CICS Transaction Server for z/OS
5.6

Developing CICS Applications

Note

Before using this information and the product it supports, read the information in Product Legal Notices.

This edition applies to the IBM® CICS® Transaction Server for z/OS®, Version 5 Release 6 (product number 5655-Y305655-BTA) and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1974, 2023.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this PDF ix

Chapter 1. CICS applications 1
- CICS programs, transactions and tasks 1
- Transactions in CICS 2
- CICS programming 8
- CICS data objects 9
- CICS programming commands 11
- EXEC interface block (EIB) 12
- Program translation 12
- Testing for CICS 13
- CICS programming roadmap 15

Chapter 2. Making asynchronous requests 17
- Benefits of using the asynchronous API 18
- About the asynchronous API 19
- Managing performance with the asynchronous API 20
- Using JCICS with the asynchronous API 22
- Example: Running a credit check application with asynchronous processing versus with traditional sequential processing 23

Chapter 3. Cloud-enabling CICS Transaction Server for z/OS 27
- How it works: platforms 31
  - Platform directory structure on zFS 43
  - Platform states 44
- How it works: applications 46
  - Application binding 49
  - Application entry points 50
  - Application context 53
  - Application states 58
- How it works: multi-versioning applications 61
  - Assigning a version to your applications 71

Chapter 4. Designing applications 73
- How tasks are started 73
- Which transaction? 74
- Separating business and presentation logic 77
- Multithreading: Reentrant, quasi-reentrant, and threadsafe programs 78
  - Quasi-reentrant application programs 79
  - Threadsafe programs 80
  - Making applications threadsafe 82
  - CONCURRENCY(REQUIRED) programs 85
  - OPENAPI programs 86
  - Using the FORCEQR system initialization parameter 88
  - Non-reentrant programs 88
- Storing data in a transaction 89
  - Transaction work area (TWA) 89
  - User storage 89
  - COMMAREA in LINK and XCTL commands 90
  - Channels in LINK and XCTL commands 90

Program storage ... 90
Temporary storage queues ... 91
Intrapartition transient data ... 92
GETMAIN SHARED command ... 93
Your own data sets ... 93
Lengths of areas passed to CICS commands ... 93
Minimizing errors ... 94
Non-terminal transaction security ... 95
Design for performance ... 96
Program size ... 96
Virtual storage ... 97
Exclusive control of resources ... 100
Operational control ... 100
Operating system waits ... 101
The NOSUSPEND option ... 101
Efficient sequential data set access ... 102
Efficient logging ... 103
Designing for asynchronous requests ... 103
Sharing data across transactions ... 104
Using the common work area (CWA) ... 104
Using the TCTTE user area (TCTUA) ... 106
Using the COMMAREA in RETURN commands ... 107
Using a channel on RETURN commands ... 108
Using the display screen to share data ... 108
Transferring data between programs using channels ... 108
Channels: quick start ... 109
Using channels: some typical scenarios ... 112
Benefits of channels ... 118
Creating a channel ... 119
The current channel ... 120
The scope of a channel ... 124
Discovering which containers were passed to a program ... 129
Discovering which containers were returned from a link ... 129
Deleting channels and containers and freeing their storage ... 129
Designing a channel: Best practices ... 131
Constructing and using a channel: an example ... 132
CICS read-only containers ... 133
Using channels with Java classes ... 134
Channels and BTS activities ... 134
Dynamic routing with channels ... 136
Data conversion ... 137
Migrating from COMMAREAs to channels ... 143
Program control ... 146
Program linking ... 147
Passing data to other programs ... 148
Using mixed addressing modes ... 152
Using LINK to pass data ... 152
Using RETURN to pass data ... 154
Affinity ... 156
Types of affinity ... 157
Programming techniques and affinity ... 158
Programming techniques that avoid affinity ... 159
Programming techniques that create affinities ... 163
Programming techniques that might create affinities ... 169
Duration and scope of inter-transaction affinities ... 179
Recovery design ... 186
Journaling ... 186
Syncpointing ... 188

Dealing with exception conditions..... 189
Default CICS exception handling..... 190
Handling exception conditions by inline code..... 190
Modifying default CICS exception handling 193
Using the HANDLE CONDITION command..... 195
Using the HANDLE CONDITION ERROR command..... 196
Using the IGNORE CONDITION command..... 197
Using the HANDLE ABEND command..... 198
Using PUSH HANDLE and POP HANDLE commands..... 198
Parsing SOAP fault messages..... 199
Avoiding the storm drain effect..... 202
Abnormal termination recovery..... 204
Creating a program-level abend program or routine..... 205
Retrying operations..... 206
Monitoring application performance..... 207
Determining a user's access to resources by using the **QUERY SECURITY** command 208
Designing applications to use the user-defined resources..... 209
CICS intercommunication..... 209
Transaction routing..... 210
Function shipping..... 210
Distributed program link (DPL)..... 211
Asynchronous processing..... 222
Distributed transaction processing (DTP)..... 222
Common Programming Interface Communications (CPI Communications)..... 223
External CICS interface (EXCI)..... 223
CICS services for application programs..... 224
Understanding file control..... 224
Terminal control..... 256
Interval control..... 287
Task control..... 289
CICS storage protection and transaction isolation 292
Transient data control..... 303
Temporary storage control..... 306
CICS documents and document templates 307
Setting up document templates..... 313
Programming with documents and document templates..... 320
Named counter servers..... 330
Printing and spool files..... 345
Basic mapping support..... 366

Chapter 5. Developing for asynchronous requests..... 443

Chapter 6. Developing Assembler language applications..... 449
Assembler language programming restrictions and requirements..... 449
Language Environment coding requirements for assembler language applications..... 453
Coding DFHEIENT for AMODE(24) and AMODE(31) programs..... 455
Coding DFHEIENT for AMODE(64) programs..... 457
Coding DFHEIRET..... 458
Calling assembler language programs..... 458
Extending dynamic storage..... 460
Developing AMODE(64) assembler language programs..... 461

Chapter 7. Developing C and C++ applications..... 467
C and C++ programming restrictions and requirements..... 468
Passing arguments in C and C++..... 471
Accessing the EIB from C and C++..... 472
Locale support for C and C++..... 473

XPLink and C and C++ programming....473
XPLink uses X8 and X9 mode TCBs....474
Passing control between XPLink and non-XPLink objects....474
Global user exits and XPLink....474
Using the CICS foundation classes....475
C++ Objects....475
Overview of the foundation classes....476
Buffer objects....481
Using CICS Services....484
Compiling, executing, and debugging....499
Conditions, errors, and exceptions....501
Polymorphic Behavior....507
Storage management....511
Parameter passing conventions....511
Scope of data in IccBuf reference returned from 'read' methods....512

Chapter 8. Developing COBOL applications....513
COBOL programming restrictions and requirements....513
Language Environment CBLPSHPOP option....516
Using the DL/I CALL interface....516
VS COBOL II programs....517
Using based addressing with COBOL....518
Calling subprograms from COBOL programs....519
Flow of control between programs and subprograms....520
Rules for calling subprograms....522
COBOL2 and COBOL3 translator options....524
CICS translator actions for COBOL programs....525
Batch compilation for COBOL programs....527
Nested COBOL programs....528

Chapter 9. Programming languages and Language Environment....533
Language Environment callable services....534
Language Environment abend and condition handling....535
Language Environment storage....537
Mixing languages in Language Environment....538
Dynamic Link Libraries (DLLs)....540
Defining runtime options for Language Environment....541
CEEBXITA and CEECSTX user exits....543
CICSVAR: CICS environment variable....544
CEEBINT exit for Language Environment....545

Chapter 10. Developing PL/I applications....547
PL/I programming restrictions and requirements....547
Language Environment coding requirements for PL/I applications....548
Fetched PL/I routines....550

Chapter 11. Developing for recovery....551
Application design considerations....551
Questions relating to recovery requirements....551
Validate the recovery requirements statement....552
Designing the user's restart procedure....552
User's standby procedures....553
Communications between application and user....553
Security....553
System definitions for recovery-related functions....553
Documentation and test plans....555
Programming for recovery....555

Designing applications for recovery ... 555
Program design ... 557
Managing transaction and system failures ... 562
Locking (enqueuing on) resources in application programs ... 566
User exits for transaction backout ... 573

Chapter 12. Translation and compilation ... 577
The integrated CICS translator ... 577
Using the integrated CICS translator ... 578
Specifying CICS translator options ... 578
The translation process ... 579
The CICS-supplied translators ... 581
Dynamic invocation of the separate translator ... 582
Using a CICS translator ... 590
Defining translator options ... 592
Using COPY statements ... 594
The CICS-supplied interface modules ... 594
Using the EXEC interface modules for AMODE(24) and AMODE(31) applications ... 595
Example assembler language program with LEASM ... 597
Using the EXEC interface modules for AMODE(64) applications ... 620
Uppercase translation ... 621
Translating national characters to uppercase ... 621
Translating TS data sharing messages to uppercase ... 622

Chapter 13. Setting up an application ... 623
Designing a CICS application for deployment to a platform ... 624
Packaging CICS applications for deployment in a cloud environment ... 625
Invoking a multi-versioned application ... 626
Examples of EXEC CICS INVOKE APPLICATION ... 627

Chapter 14. Debugging applications ... 629
Execution diagnostic facility (EDF) ... 629
Restrictions when using EDF ... 630
What does EDF display? ... 632
Testing programs using EDF ... 638
Using EDF to change information ... 643
Using EDF menu functions ... 644
Temporary storage browse (CEBR) ... 650
Using the CEBR transaction ... 650
What does the CEBR transaction display? ... 652
Using the CEBR function keys ... 653
Using the CEBR commands ... 654
Using the CEBR transaction with transient data ... 656
Command-level interpreter (CECI) ... 656
What does CECI display? ... 657
Using CECI ... 661
Defining variables ... 662
Saving commands ... 663
How CECI runs ... 664
Shared storage: ENQ commands without LENGTH option ... 665
Preparing to use debuggers with CICS applications ... 665
Debugging profiles ... 667
Using debugging profiles to select programs for debugging ... 668
Using generic parameters in debugging profiles ... 669
Debugging CICS applications from a workstation ... 670
Preparing to debug applications from a workstation ... 671
Using Debug Tool with CICS applications ... 671

Preparing to debug applications with Debug Tool... 672

Chapter 15. Deploying applications ... 673
Deploying an application to a platform... 673
Installing application programs... 674
Program installation steps... 675
Using dynamic program LIBRARY resources... 676
Defining MVS residence and addressing modes... 684
Running applications in the link pack area... 686
Running application programs in the read-only DSAs... 686
Using BMS map sets in application programs... 689
Using the CICS-supplied procedures to install application programs... 690
Including the CICS-supplied interface modules... 691
Translating, assembling, and link-editing assembler language application programs... 692
Installing COBOL application programs... 695
Installing PL/I application programs... 699
Installing C application programs... 699
Using your own job streams... 703
Installing map sets and partition sets... 706
Installing map sets... 708
Installing partition sets... 717
Defining programs, map sets, and partition sets to CICS... 718
Testing applications... 718
Preparing the application for testing... 719
Preparing the system for testing... 719
Preparing CICS Db2 programs for execution and production ... 720
The CICS Db2 test environment... 720
CICS Db2 program preparation... 721
What to bind after a program change... 723
Bind options and considerations for programs... 724
CICS Db2 program testing and debugging ... 725
Going into production: checklist for CICS Db2 applications... 725
Tuning a CICS application that accesses Db2... 727
CICS application build automation with the CICS build toolkit... 728
Preparing to use the CICS build toolkit... 729
Building a CICS bundle, application, application binding, or platform... 730
Resolving variables in a CICS bundle... 732
CICS build toolkit command line options... 733
Return codes... 735
Deployment automation with DFHDPLOY... 736
DFHDPLOY commands... 737
Script examples... 747
Return codes... 750
Receiving output messages in Japanese or Simplified Chinese... 750
Deployment with UrbanCode Deploy... 751
Deployment with Zowe CLI CICS deploy plug-in ... 751

Notices... 753

Index... 759

About this PDF

This PDF describes how to design and develop applications that use the EXEC CICS API to access CICS services and resources. Other PDFs, listed below, describe the application programming considerations for certain areas of CICS and you might need to refer to those as well as this PDF. (In IBM Documentation, all this information is under one section called "Developing Applications"). You are also likely to need the reference companion to this PDF: the *API (EXEC CICS) Reference*.

Programming information for areas of CICS is in the following PDFs:

- SOAP and JSON is in *Web Services Guide*.
- ONC/RPC interface is in the *External Interfaces Guide*.
- EXCI is in *Using EXCI with CICS*.
- Java™ and Liberty are in *Java Applications in CICS*.
- Front End Programming Interface is in the *Front End Programming Interface User's Guide*.
- Db2® is in *Db2 Guide*.
- DBCTL is in the *IMS DB Control Guide*.
- Shared data tables are in the *Shared Data Tables Guide*.
- CICSPlex® SM is in *CICSPlex SM Application Programming Guide*.
- BTS is in *Business Transaction Services*
- Connections between CICS systems is in the *Intercommunication Guide*

Information about resolving problems with CICS applications is in *Troubleshooting CICS*.

For details of the terms and notation used in this book, see Conventions and terminology used in CICS documentation in IBM Documentation.

Date of this PDF

This PDF was created on 2025-12-02 (Year-Month-Date).

© Copyright IBM Corp. 1974, 2023

Chapter 1. What is a CICS application?

An application is a collection of related programs that together perform a business operation, such as processing a product order, or preparing a company payroll. CICS applications execute under CICS control, using CICS services and interfaces to access programs and files.

CICS is a transaction processing subsystem. It provides services for you to run applications online, by request, at the same time as many other users are submitting requests to run the same applications, using the same files and programs. CICS manages the sharing of resources, integrity of data, and prioritization of execution, with fast response.

CICS applications are traditionally run by submitting a *transaction* request. Execution of the transaction consists of running one or more *application programs* that implement the required function. In CICS documentation, CICS application programs are sometimes called *programs*, and the term *transaction* is used to imply the processing that the application programs do.

The term *transaction* is now used extensively in the IT industry to describe a *unit of recovery*, which in CICS is called a *unit of work*. This is typically a complete operation that is recoverable; it can be committed or backed out as an entirety, as a result of programmed command or system failure. In many cases, the scope of a CICS transaction is also a single unit of work, but be aware of the difference in meaning when you use non-CICS documentation.

CICS programs, transactions and tasks

To develop and run CICS applications, you need to understand the relationship between CICS programs, transactions, and tasks.

The following terms are used throughout CICS documentation and in many commands:

Program

A program is used by CICS to process an entire transaction, or part of a transaction. Programs are stored in the program library. For more information about program resources and programs used in CICS bundles, see PROGRAM resources.

Transaction

A transaction is an item of processing that is initiated by a single request. This request is usually from an end-user at a terminal, but can also be made from a web page, a remote workstation program, or an application in another CICS system, or triggered automatically at a predefined time. For descriptions of different ways to run CICS transactions, see Internet, TCP/IP, and HTTP concepts and Overview of CICS external interfaces.

A single transaction consists of one or more application programs that, when run, carry out the processing needed.

However, the term *transaction* is used in CICS to mean both a single event and all other transactions of the same type. You describe each transaction type to CICS with a TRANSACTION resource definition. This definition gives the transaction type a name (the transaction identifier, or TRANSID) and provides information to CICS about the work to be done; for example, which program to invoke first, and what kind of authentication is required throughout the execution of the transaction.

You run a transaction by submitting its TRANSID to CICS. CICS uses the information recorded in the TRANSACTION definition to establish the correct execution environment, and starts the first program.

The term *transaction* is now used extensively in the IT industry to describe a *unit of recovery*, which in CICS is called a *unit of work*. This is typically a complete operation that is recoverable; it can be committed or backed out as an entirety, as a result of programmed command or system failure. In many cases, the scope of a CICS transaction is also a single unit of work, but be aware of the difference in meaning when you use non-CICS documentation.

© Copyright IBM Corp. 1974, 2023

Task

The term *task* has a specific meaning in CICS and CICS documentation. When CICS receives a request to run a transaction, it starts a new task that is associated with this one instance of the execution of the transaction type; that is, one execution of a transaction, with a specific set of data, usually on behalf of a specific user at a specific terminal. You can also consider a task as analogous to a thread. When the transaction completes, the task ends.

Related concepts

CICS programming
You write a CICS program in a similar way to any other program. You can use COBOL, C, C++, Java, PL/I, or assembler language to write CICS application programs. Most of the processing logic is expressed in standard language statements, but you use CICS commands, or the Java and C++ class libraries, to request CICS services. You can also use JavaScript to write applications that access the invoke API.

CICS data objects
CICS provides a number of different objects that CICS programs and transactions can use to exchange data with one another.

CICS programming commands
The general format of a CICS command is EXECUTE CICS (or EXEC CICS) followed by the name of the required command and possibly one or more options.

EXEC interface block (EIB)
In addition to the usual CICS control blocks, each task in a command-level environment has a control block known as the EXEC interface block (EIB) associated with it.

Program translation
Some older compilers (and assemblers) cannot process CICS commands directly. An additional step is needed to convert your program into executable code. This step is called *translation*, and consists of converting CICS commands into the language in which the rest of the program is coded, so that the compiler (or assembler) can understand them.

Related tasks

Testing for CICS
A program has two ways to determine whether it is running in a CICS environment: by using the C language iscics() function or by calling the DFH3QSS program.

CICS programming roadmap
Follow the steps in this roadmap to develop a CICS application that uses the EXEC CICS command level programming interface.

Related information

Transactions in CICS
A CICS transaction consists of an item of processing that can be run by one or more application programs. Transactions can be defined by users of CICS, and are referred to as *user transactions* in CICS documentation. CICS also supplies transactions to control CICS operations, and these are *CICS transactions*.

Transactions in CICS

A CICS transaction consists of an item of processing that can be run by one or more application programs. Transactions can be defined by users of CICS, and are referred to as *user transactions* in CICS documentation. CICS also supplies transactions to control CICS operations, and these are *CICS transactions*.

The term *transaction* is used extensively in the IT industry to describe a unit of recovery, typically a complete operation that is recoverable. It can be committed or backed out as an entirety, as a result of programmed command or system failure. In CICS, this is called a *unit of work*. In many cases, the scope of a CICS transaction is also a single unit of work but it doesn't have to be.

In CICS, unless a programmer chooses more granular control over units of work, all of the updates that are made as part of a transaction are kept in-doubt until CICS finishes processing the request. When

the processing is complete, all the updates are finalized at the same time. This implicit way of handling transactions and units of work requires no code changes and is transparent to the applications. If the program wants more control over the transaction model, they can use two CICS API calls: one is called syncpoint, used to finalize the in-doubt unit of work and to start a new one. The second is rollback, used to signify that this unit of work can no longer complete and should be backed out.

Unlike other transaction architectures, CICS can maintain a transaction over the execution of programs that are written in different programming languages.

An analogy of a transaction

To understand a CICS transaction where the transaction is also a single unit of work, think of a vending machine that contains your favorite treat. To obtain the treat you must enter a certain amount of money, enter a code to tell the machine what treat you would like and in return, the machine takes your money and dispenses the treat. The vending machine is acting as a transaction server that processes your request to purchase a treat.

At the beginning, you and the vending machine are each in a starting state. You know the processing that you want to run (you want to buy a treat), you know the transaction (the code that is used by the machine), and you have the correct change (data) to obtain your treat. The vending machine is error free, knows its stock levels, and is ready to process a request. At the end of the transaction, both you and the machine reach your end state. You no longer have your money, but you have a treat. The machine takes your money and updates its stock levels. The transaction is the complete operation of your purchase.

Transactional integrity is an important part of CICS. In our vending machine example, what if something goes wrong? The treat fails to dispense, the product code is unknown by the machine, or some other issue arises. As a result, the transaction cannot complete as requested. In this example, ideally, the vending machine returns your money. This situation is known as backing out the transaction. Because the machine cannot progress you to the end-state, it must return both actors in the transaction to the beginning state (as if the transaction never started). Consider an application that needs to make the same update to two different files. If for some reason the second file cannot be updated, CICS can roll back the transaction and back out the update from both files.

Transactions and transaction definitions

You describe each transaction type to CICS with a TRANSACTION resource definition. This definition gives the transaction type a name (the transaction identifier, or TRANSID) and provides information to CICS about the work to be done; for example, which program to start first, and what kind of authentication is required throughout the execution of the transaction.

You run a transaction by submitting its TRANSID to CICS. CICS uses the information that is recorded in the TRANSACTION definition to establish the correct execution environment, and starts the first program.

A transaction is an item of processing that is initiated by a single request. This request can be from various sources, including a terminal, a web page, an application in another CICS system, or triggered automatically at a predefined time. For descriptions of different ways to run CICS transactions, see Internet, TCP/IP, and HTTP concepts and Overview of CICS external interfaces.

User transactions

These transactions are defined by you, rather than supplied with CICS.

You tell CICS how you want your transaction to run with a TRANSACTION resource definition. This definition gives the transaction type a name (TRANSID) and provides information to CICS about the work to be done; for example, which program to start first, and what kind of authentication is required throughout the execution of the transaction. On the TRANSACTION definition, you also link the transaction with other resources, such as programs or profiles.

You can use a CICS bundle to create, edit, and install a TRANSACTION resource definition. If you create a TRANSACTION resource in this way, you must use the CICS bundle to manage the lifecycle of that

resource, and you cannot manage the resource independently. For more information, see Defining CICS bundles.

For applications that run on CICS platforms (see Cloud-enabling CICS), a TRANSACTION resource can be declared as an application entry point.

For security, define users of these transactions to specific RACF groups and give the groups access only to the transactions that apply to them. For more information, see Transaction security.

For information about defining the transaction, see TRANSACTION resources.

CICS transactions

CICS transactions have transaction identifiers that start with C and are 4 characters long; for example, CEMT. For a complete list of CICS transactions, including the transactions that do not have an operator interface, see the List of CICS transactions.

The CICS transactions are described in three categories. Each category specifies the recommended security.

Category 1

Category 1 transactions are for CICS internal use only and must not be started from a user terminal. Because these transactions are part of CICS itself, they run under the CICS region user ID. CICS does not validate that the CICS region user ID has authority to run Category 1 transactions because these transactions are supplied with CICS. This reduces security risks that can be caused due to misconfiguration because the CICS region user ID has implicit authority to run these transactions.

Some Category 1 transactions are defined by CICS during initialization rather than in the CICS system definition data set (CSD).

For a list of transactions in this category, see List of CICS Category 1 transactions.

Category 2

Category 2 transactions are similar to user transactions. They are initiated by CICS users or are associated with CICS users in different roles, such as application developer, operator, or system programmer. It is unlikely that users require access to all the transactions in this category, so you can group them in subcategories, as shown in Table 1 on page 4. The sample CLIST DFH$CAT2 (in library CICSTS56.CICS.SDFHSAMP) shows one way that you might group the Category 2 transactions. To use a different setup, adapt the CLIST, or provide your own.

Define users of these transactions to specific RACF groups and give the groups access only to the subcategory of transactions that apply to them.

Table 1. Suggested subcategories for Category 2 transactions.

Subcategory	Contains	Notes
For all users:		
ALLUSER	Transactions that are used by all users.	Add to the list of transactions your "good morning" transaction (defined on the **GMTRAN** system initialization parameter) and your "good night" transaction in this group (defined on the **GNTRAN** system initialization parameter).
For operators:		
SYSADM	Transactions that are used by system programmers who need full access to the system.	

Table 1. Suggested subcategories for Category 2 transactions. (continued)		
Subcategory	**Contains**	**Notes**
INQUIRE	Transactions that are used by operators or other users who need only to inquire on resources.	
OPERATOR	Transactions that are used by operators.	
DBCTL	Transactions that are used by operators of the DBCTL interface to IMS.	
CMCIUSER	Transactions that are used by operators who use the CICS Explorer® and other users of the CMCI.	
For application developers:		
DEVELOPER	Transactions that are used by application developers on a non-production system.	
For application users:		
JVMUSER	Transactions that are used by users of Liberty applications.	Depending on your security configuration, transactions CJSA and CJSU might require the CICS default user to have access. For more information, see Security for Java applications. As a security best practice, turn on Liberty security and avoid using the CICS default user to run application tasks. For the z/OS Connect Service, you can install a URIMAP resource that CICS uses to associate the work for the Service with a specific transaction ID in CICS and with an initial user ID. CJSA is the default transaction ID for any web request that does not have a matching URI. Consider restricting access to CJSA to prevent any arbitrary application being run.

Table 1. Suggested subcategories for Category 2 transactions. (continued)		
Subcategory	**Contains**	**Notes**
INTERCOM	Transactions that are used by application users who run transactions on multiple regions.	If you are using function shipping, the mirror transactions must be available to remote users in a function shipping environment. When a database or file is on another CICS region, CICS function ships the request to access the data. The request runs under one of the CICS-supplied mirror transactions. In this situation, the following conditions apply: • The terminal user that runs the application must be authorized to use the mirror transaction. See Transaction security. • The terminal user must also be authorized to use the data that the mirror transaction accesses. The mirror transactions are supplied with RESSEC(YES) defined; so, even if the user's transaction specifies RESSEC(NO), the mirror transaction fails if the user is not authorized to access the data. If you do not use resource security checking, change the mirror transaction definitions to specify RESSEC(NO). Because the mirror transactions are an IBM-protected resource, first copy these definitions into your own groups and then change them. For a deferred **START** request, if the user transaction to be started is eligible for dynamic routing, system transaction CDFS will run and start the user transaction at the specified time. Ensure that security for CDFS is correctly configured.
WEBUSER	Transactions that are used by application users who run transactions through the CICS web interface.	The CICS default user requires access to the CWBA transaction initially, even if an analyzer program is then used to assign another user ID to the task. Make sure that the CICS default user that is specified in the **DFLTUSER** system initialization parameter has access to this transaction. If you use the supplied CLIST DFH$CAT2 to create a WEBUSER RACF® profile, the default user must have access to this profile.

Table 1. Suggested subcategories for Category 2 transactions. (continued)		
Subcategory	**Contains**	**Notes**
RPCUSER	Transactions that are used by application users who run transactions through ONC RPC.	
PIPEUSER	Transactions that are used by application users who run web services transactions.	
EVENTUSER	Transactions that are used by EP adapters.	If the RESSEC and CMDSEC options for these transactions are not the ones you want, you can specify your own transaction IDs in the adapter tab **Advanced Options** section of the Event binding editor. For more information, see Specifying EP adapter and dispatcher information.
For users of IBM MQ:		
MQADMIN	CKAM CKCN CKDL CKRS CKSD CKSQ	
MQBRIDGE	CKBC CKBP CKBR	
MQMONITOR	CKTI	
MQSTATUS	CKQC CKBM CKRT CKDP	

For a list of transactions in this category, see List of CICS Category 2 transactions.

Category 3

Category 3 transactions are available to all users, whether signed-on or not. These transactions are not subject to security checking.

For a list of transactions in this category, see List of CICS Category 3 transactions.

For more information, see Transaction security.

Most of the CICS transactions are generated in the designated groups when you initialize your CICS system definition data set (CSD). Do not modify the CSD definitions of the Category 1 transaction. This initialization does not include the CICS sample transactions (the transactions that are in Category 2 and in groups that start with DFH$). Some Category 1 transactions are not in the CSD. They are defined by CICS during installation.

Related concepts

CICS programs, transactions and tasks

To develop and run CICS applications, you need to understand the relationship between CICS programs, transactions, and tasks.

CICS programming
You write a CICS program in a similar way to any other program. You can use COBOL, C, C++, Java, PL/I, or assembler language to write CICS application programs. Most of the processing logic is expressed in standard language statements, but you use CICS commands, or the Java and C++ class libraries, to request CICS services. You can also use JavaScript to write applications that access the invoke API.

CICS data objects
CICS provides a number of different objects that CICS programs and transactions can use to exchange data with one another.

CICS programming commands
The general format of a CICS command is EXECUTE CICS (or EXEC CICS) followed by the name of the required command and possibly one or more options.

EXEC interface block (EIB)
In addition to the usual CICS control blocks, each task in a command-level environment has a control block known as the EXEC interface block (EIB) associated with it.

Program translation
Some older compilers (and assemblers) cannot process CICS commands directly. An additional step is needed to convert your program into executable code. This step is called *translation*, and consists of converting CICS commands into the language in which the rest of the program is coded, so that the compiler (or assembler) can understand them.

Related tasks

Testing for CICS
A program has two ways to determine whether it is running in a CICS environment: by using the C language iscics() function or by calling the DFH3QSS program.

CICS programming roadmap
Follow the steps in this roadmap to develop a CICS application that uses the EXEC CICS command level programming interface.

CICS programming

You write a CICS program in a similar way to any other program. You can use COBOL, C, C++, Java, PL/I, or assembler language to write CICS application programs. Most of the processing logic is expressed in standard language statements, but you use CICS commands, or the Java and C++ class libraries, to request CICS services. You can also use JavaScript to write applications that access the invoke API.

This information describes the use of the CICS command level programming interface, **EXEC CICS**, that can be used in COBOL, C, C++, PL/I or assembler language programs. These commands are defined in detail in CICS Application development reference.

The following programming information is also available:

- Programming in Java with the JCICS class library is described in Java development using JCICS.
- Programming in C++ with the CICS C++ classes is described in Using the CICS foundation classes.
- Programming in JavaScript to access the invoke API is described in Calling CICS services
- For information about writing web applications to process HTTP requests and responses, see Developing HTTP applications.

For further guidance on language use with CICS, see Programming languages and Language Environment.

As well as CICS commands, you can use SQL statements, DLI requests, CPI statements, and the CICS Front End Programming Interface (FEPI) commands in your program. See the following information for the relevant details:

- To use SQL, see SQL: The language of Db2 in Db2 for z/OS product documentation and Programming for Db2 for z/OS in Db2 for z/OS product documentation.

- To use DL/I, see Application programming for EXEC DLI in IMS product documentation and Application programming design in IMS product documentation.
- To use CPI, see z/VM: CPI Communications User's Guide.
- To use FEPI, see Developing with the FEPI API.
- To use IBM MQ, see IBM MQ product documentation.

Related concepts

CICS programs, transactions and tasks
To develop and run CICS applications, you need to understand the relationship between CICS programs, transactions, and tasks.

CICS data objects
CICS provides a number of different objects that CICS programs and transactions can use to exchange data with one another.

CICS programming commands
The general format of a CICS command is EXECUTE CICS (or EXEC CICS) followed by the name of the required command and possibly one or more options.

EXEC interface block (EIB)
In addition to the usual CICS control blocks, each task in a command-level environment has a control block known as the EXEC interface block (EIB) associated with it.

Program translation
Some older compilers (and assemblers) cannot process CICS commands directly. An additional step is needed to convert your program into executable code. This step is called *translation*, and consists of converting CICS commands into the language in which the rest of the program is coded, so that the compiler (or assembler) can understand them.

Related tasks

Testing for CICS
A program has two ways to determine whether it is running in a CICS environment: by using the C language iscics() function or by calling the DFH3QSS program.

CICS programming roadmap
Follow the steps in this roadmap to develop a CICS application that uses the EXEC CICS command level programming interface.

Related information

Transactions in CICS
A CICS transaction consists of an item of processing that can be run by one or more application programs. Transactions can be defined by users of CICS, and are referred to as *user transactions* in CICS documentation. CICS also supplies transactions to control CICS operations, and these are *CICS transactions*.

CICS data objects

CICS provides a number of different objects that CICS programs and transactions can use to exchange data with one another.

CICS provides these data objects:

Communication areas
A *communication area* (COMMAREA) is a single block of storage that is used to pass data between programs. A program can pass just one COMMAREA to another program. The recommended maximum size of a communication areas is 24 KB, although the absolute maximum size is 32 KB.

Containers and channels
A container is a named block of data that is used for passing information between programs. A program can pass any number of containers to another program using a logical grouping of containers known as a *channel*. Containers are not limited to a maximum size of 32 KB.

Containers that are associated with a business transaction services (BTS) process are *recoverable*; they can be rolled back to their state at the start of a transaction.

Temporary storage queues

A temporary storage queue is a named queue of data items that can be written, rewritten, read, and reread, in any sequence. The recommended maximum size of a data item is 24 KB, although the absolute maximum size is 32 KB.

You can define temporary storage queues as recoverable.

Transient data queues

A transient data queue is a named queue of data items that can be written and read once only. The maximum size of a data item can be up to 32 KB.

Transient data queues must be read sequentially, and each item can be read only once; after a transaction reads an item, that item is removed from the queue and is not available to any other transaction.

CICS provides two types of transient data queues:

- *Intrapartition transient data queues* are used primarily for communication between CICS programs.
- *Extrapartition transient data queues* are used primarily for communication between a CICS program and a sequential device, such as a QSAM data set or a printer.

You can define intrapartition transient data queues as recoverable.

CICS also provides a variety of facilities to store data in and between transactions. For more information, see Storing data in a transaction.

Related concepts

CICS programs, transactions and tasks
To develop and run CICS applications, you need to understand the relationship between CICS programs, transactions, and tasks.

CICS programming
You write a CICS program in a similar way to any other program. You can use COBOL, C, C++, Java, PL/I, or assembler language to write CICS application programs. Most of the processing logic is expressed in standard language statements, but you use CICS commands, or the Java and C++ class libraries, to request CICS services. You can also use JavaScript to write applications that access the invoke API.

CICS programming commands
The general format of a CICS command is EXECUTE CICS (or EXEC CICS) followed by the name of the required command and possibly one or more options.

EXEC interface block (EIB)
In addition to the usual CICS control blocks, each task in a command-level environment has a control block known as the EXEC interface block (EIB) associated with it.

Program translation
Some older compilers (and assemblers) cannot process CICS commands directly. An additional step is needed to convert your program into executable code. This step is called *translation*, and consists of converting CICS commands into the language in which the rest of the program is coded, so that the compiler (or assembler) can understand them.

Related tasks

Testing for CICS
A program has two ways to determine whether it is running in a CICS environment: by using the C language iscics() function or by calling the DFH3QSS program.

CICS programming roadmap
Follow the steps in this roadmap to develop a CICS application that uses the EXEC CICS command level programming interface.

Related information

Transactions in CICS

A CICS transaction consists of an item of processing that can be run by one or more application programs. Transactions can be defined by users of CICS, and are referred to as *user transactions* in CICS documentation. CICS also supplies transactions to control CICS operations, and these are *CICS transactions*.

CICS programming commands

The general format of a CICS command is EXECUTE CICS (or EXEC CICS) followed by the name of the required command and possibly one or more options.

You can write many application programs using the CICS command-level interface without any knowledge of, or reference to, the fields in the CICS control blocks and storage areas. However, you might need to get information that is valid outside the local environment of your application program. You can use ADDRESS and ASSIGN commands to access such information.

When you use the ADDRESS and ASSIGN commands, various fields can be read but should not be set or used in any other way. Do not use any of the CICS fields as arguments in CICS commands, because these fields might be altered by the EXEC interface modules.

You can use the INQUIRE, SET, and PERFORM commands for application programs to access information about CICS resources. These commands are known as system programming commands. The application program can retrieve and modify information for CICS data sets, terminals, system entries, mode names, system attributes, programs, and transactions. These commands, plus the spool commands of the CICS interface to the Job Entry Subsystem (JES), are primarily for system programmers to use.

For reference information about CICS programming commands, see CICS API commands.

Related concepts

CICS programs, transactions and tasks
To develop and run CICS applications, you need to understand the relationship between CICS programs, transactions, and tasks.

CICS programming
You write a CICS program in a similar way to any other program. You can use COBOL, C, C++, Java, PL/I, or assembler language to write CICS application programs. Most of the processing logic is expressed in standard language statements, but you use CICS commands, or the Java and C++ class libraries, to request CICS services. You can also use JavaScript to write applications that access the invoke API.

CICS data objects
CICS provides a number of different objects that CICS programs and transactions can use to exchange data with one another.

EXEC interface block (EIB)
In addition to the usual CICS control blocks, each task in a command-level environment has a control block known as the EXEC interface block (EIB) associated with it.

Program translation
Some older compilers (and assemblers) cannot process CICS commands directly. An additional step is needed to convert your program into executable code. This step is called *translation*, and consists of converting CICS commands into the language in which the rest of the program is coded, so that the compiler (or assembler) can understand them.

Related tasks

Testing for CICS
A program has two ways to determine whether it is running in a CICS environment: by using the C language iscics() function or by calling the DFH3QSS program.

CICS programming roadmap
Follow the steps in this roadmap to develop a CICS application that uses the EXEC CICS command level programming interface.

Related information

Transactions in CICS

A CICS transaction consists of an item of processing that can be run by one or more application programs. Transactions can be defined by users of CICS, and are referred to as *user transactions* in CICS documentation. CICS also supplies transactions to control CICS operations, and these are *CICS transactions*.

EXEC interface block (EIB)

In addition to the usual CICS control blocks, each task in a command-level environment has a control block known as the EXEC interface block (EIB) associated with it.

An application program can access all of the fields in the EIB by name. The EIB contains information that is useful during the execution of an application program, such as the transaction identifier, the time and date (initially when the task is started, and subsequently, if updated by the application program using ASKTIME), and the cursor position on a display device. The EIB also contains information that is helpful when a dump is used to debug a program. For programming information about EIB fields, see EIB fields.

Related concepts

CICS programs, transactions and tasks
To develop and run CICS applications, you need to understand the relationship between CICS programs, transactions, and tasks.

CICS programming
You write a CICS program in a similar way to any other program. You can use COBOL, C, C++, Java, PL/I, or assembler language to write CICS application programs. Most of the processing logic is expressed in standard language statements, but you use CICS commands, or the Java and C++ class libraries, to request CICS services. You can also use JavaScript to write applications that access the invoke API.

CICS data objects
CICS provides a number of different objects that CICS programs and transactions can use to exchange data with one another.

CICS programming commands
The general format of a CICS command is EXECUTE CICS (or EXEC CICS) followed by the name of the required command and possibly one or more options.

Program translation
Some older compilers (and assemblers) cannot process CICS commands directly. An additional step is needed to convert your program into executable code. This step is called *translation*, and consists of converting CICS commands into the language in which the rest of the program is coded, so that the compiler (or assembler) can understand them.

Related tasks

Testing for CICS
A program has two ways to determine whether it is running in a CICS environment: by using the C language iscics() function or by calling the DFH3QSS program.

CICS programming roadmap
Follow the steps in this roadmap to develop a CICS application that uses the EXEC CICS command level programming interface.

Related information

Transactions in CICS
A CICS transaction consists of an item of processing that can be run by one or more application programs. Transactions can be defined by users of CICS, and are referred to as *user transactions* in CICS documentation. CICS also supplies transactions to control CICS operations, and these are *CICS transactions*.

Program translation

Some older compilers (and assemblers) cannot process CICS commands directly. An additional step is needed to convert your program into executable code. This step is called *translation*, and consists of

converting CICS commands into the language in which the rest of the program is coded, so that the compiler (or assembler) can understand them.

Most compilers use the integrated CICS translator approach, where the compiler interfaces with CICS at compile time to interpret CICS commands and convert them automatically to calls to CICS service routines. If you use the integrated CICS translator approach, many of the translation tasks are done at compile time and you do not need to complete the additional translation step. For more information about the task in the translation step, see The translation process.

Related concepts

CICS programs, transactions and tasks
To develop and run CICS applications, you need to understand the relationship between CICS programs, transactions, and tasks.

CICS programming
You write a CICS program in a similar way to any other program. You can use COBOL, C, C++, Java, PL/I, or assembler language to write CICS application programs. Most of the processing logic is expressed in standard language statements, but you use CICS commands, or the Java and C++ class libraries, to request CICS services. You can also use JavaScript to write applications that access the invoke API.

CICS data objects
CICS provides a number of different objects that CICS programs and transactions can use to exchange data with one another.

CICS programming commands
The general format of a CICS command is EXECUTE CICS (or EXEC CICS) followed by the name of the required command and possibly one or more options.

EXEC interface block (EIB)
In addition to the usual CICS control blocks, each task in a command-level environment has a control block known as the EXEC interface block (EIB) associated with it.

Related tasks

Testing for CICS
A program has two ways to determine whether it is running in a CICS environment: by using the C language iscics() function or by calling the DFH3QSS program.

CICS programming roadmap
Follow the steps in this roadmap to develop a CICS application that uses the EXEC CICS command level programming interface.

Related information

Transactions in CICS
A CICS transaction consists of an item of processing that can be run by one or more application programs. Transactions can be defined by users of CICS, and are referred to as *user transactions* in CICS documentation. CICS also supplies transactions to control CICS operations, and these are *CICS transactions*.

Testing for CICS

A program has two ways to determine whether it is running in a CICS environment: by using the C language iscics() function or by calling the DFH3QSS program.

About this task

iscics

If you are adapting an existing C language program or writing a new program that is designed to run outside CICS as well as under CICS, the C language iscics() function can prove useful. It returns a non-zero value if your program is currently running under CICS, or zero otherwise. This function is an extension to the C library.

DFH3QSS

Your program can call the DFH3QSS program to query the CICS environment and API capability. Link DFH3QSS statically into your own application. On return, register 15 addresses a result structure that consists of a halfword length (that includes itself) followed by a reserved halfword (currently zero) followed by a bit string:

Bit 0

When set to 1, this means that the caller is running in a CICS environment (on a CICS-managed TCB or one of its descendants).

Bit 1

When set to 1, this means that the CICS API is available to the caller (in the current PSW key, ASC-mode, AMODE, and cross-memory environment).

The output structure remains accessible as long as the TCB under which the request was issued has not terminated and DFH3QSS itself is still present in virtual storage. Any change of execution state (such as PSW key, ASC-mode, AMODE, or cross-memory environment) might affect the availability of the CICS API. Registers are preserved.

Environment

This function must be called in problem state.

Related concepts

CICS programs, transactions and tasks

To develop and run CICS applications, you need to understand the relationship between CICS programs, transactions, and tasks.

CICS programming

You write a CICS program in a similar way to any other program. You can use COBOL, C, C++, Java, PL/I, or assembler language to write CICS application programs. Most of the processing logic is expressed in standard language statements, but you use CICS commands, or the Java and C++ class libraries, to request CICS services. You can also use JavaScript to write applications that access the invoke API.

CICS data objects

CICS provides a number of different objects that CICS programs and transactions can use to exchange data with one another.

CICS programming commands

The general format of a CICS command is EXECUTE CICS (or EXEC CICS) followed by the name of the required command and possibly one or more options.

EXEC interface block (EIB)

In addition to the usual CICS control blocks, each task in a command-level environment has a control block known as the EXEC interface block (EIB) associated with it.

Program translation

Some older compilers (and assemblers) cannot process CICS commands directly. An additional step is needed to convert your program into executable code. This step is called *translation*, and consists of converting CICS commands into the language in which the rest of the program is coded, so that the compiler (or assembler) can understand them.

Related tasks

CICS programming roadmap

Follow the steps in this roadmap to develop a CICS application that uses the EXEC CICS command level programming interface.

Related information

Transactions in CICS

A CICS transaction consists of an item of processing that can be run by one or more application programs. Transactions can be defined by users of CICS, and are referred to as *user transactions* in

CICS documentation. CICS also supplies transactions to control CICS operations, and these are *CICS transactions*.

CICS programming roadmap

Follow the steps in this roadmap to develop a CICS application that uses the EXEC CICS command level programming interface.

Procedure

1. Design your application, identifying the CICS resources and services you will use.

   See Designing applications and Design for performance for guidance on designing CICS applications.
2. Write your program in the language of your choice, including **EXEC CICS** commands to request CICS services.

   See CICS command summary for a list of CICS commands.
3. Translate and compile your program.

   If you are using a compiler that incorporates The integrated CICS translator, you will only need to compile your program, and then install it in CICS, using the process described in Installing application programs. Otherwise, you will need to define translator options for your program, using the process described in Using a CICS translator, and then translate and compile your program, and install it in CICS, using the process described in Installing application programs.
4. Define your program and related transaction to CICS.

   Use PROGRAM resources and TRANSACTION resources.
5. Define and install any CICS resources that your program uses, such as files, queues or terminals.
6. Run your program.

   Enter the transaction identifier at a CICS terminal, or use any of the methods described in Interfaces to CICS transactions and programs.

Related concepts

CICS programs, transactions and tasks
To develop and run CICS applications, you need to understand the relationship between CICS programs, transactions, and tasks.

CICS programming
You write a CICS program in a similar way to any other program. You can use COBOL, C, C++, Java, PL/I, or assembler language to write CICS application programs. Most of the processing logic is expressed in standard language statements, but you use CICS commands, or the Java and C++ class libraries, to request CICS services. You can also use JavaScript to write applications that access the invoke API.

CICS data objects
CICS provides a number of different objects that CICS programs and transactions can use to exchange data with one another.

CICS programming commands
The general format of a CICS command is EXECUTE CICS (or EXEC CICS) followed by the name of the required command and possibly one or more options.

EXEC interface block (EIB)
In addition to the usual CICS control blocks, each task in a command-level environment has a control block known as the EXEC interface block (EIB) associated with it.

Program translation
Some older compilers (and assemblers) cannot process CICS commands directly. An additional step is needed to convert your program into executable code. This step is called *translation*, and consists of converting CICS commands into the language in which the rest of the program is coded, so that the compiler (or assembler) can understand them.

Related tasks

Testing for CICS

A program has two ways to determine whether it is running in a CICS environment: by using the C language iscics() function or by calling the DFH3QSS program.

Related information

Transactions in CICS

A CICS transaction consists of an item of processing that can be run by one or more application programs. Transactions can be defined by users of CICS, and are referred to as *user transactions* in CICS documentation. CICS also supplies transactions to control CICS operations, and these are *CICS transactions*.

Chapter 2. Making asynchronous requests

Using the asynchronous API commands provided with CICS is a simple way to issue requests to external services and have them run asynchronously. Instead of calling each service sequentially and waiting for a response, the asynchronous API commands provide a simple and powerful way to write asynchronous applications which save wait time and frees up your program to continue with other processing.

How asynchronous processing affects CICS

Using the asynchronous API commands in CICS can enable multiple child transactions to run asynchronously, reducing the overall response time of the application. A parent can run one or more local child transactions that can run asynchronously, and will run in different CICS tasks. It is possible to pass data from a parent to a child and receive back data using CICS channels and containers. The framework of running and fetching child transactions is all done through the use of the supported asynchronous API.

The CICS asynchronous API uses **EXEC CICS** commands to control the creation, retrieval, and control of child tasks by parent tasks. These commands include:

- RUN TRANSID
- FETCH CHILD
- FETCH ANY
- FREE CHILD

Comparing sequential transaction processing and asynchronous transaction processing

In a sequential transaction processing scenario, CICS call an external service and waits for a response. CICS may then also call multiple other external services, and wait for responses from them. This sequence can continue until the user receives all the data that is needed. There can be many wait times, and the program execution is blocked until the response is received. Chapter 2, “Making asynchronous requests,” on page 17 illustrates how CICS calls an external service and waits for a response.

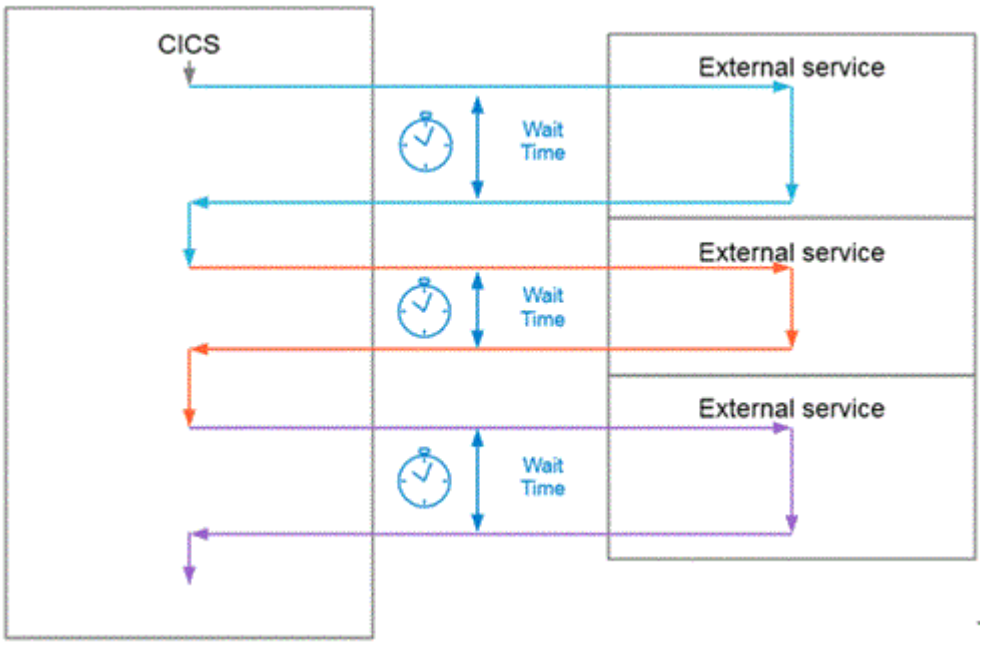


Figure 1. Calling services sequentially

© Copyright IBM Corp. 1974, 2023

Unlike sequential transaction processing, asynchronous transaction processing provides a way of issuing multiple requests simultaneously. Calling external services asynchronously can reduce the overall response time of the application: the parent is freed up to continue with its own logic separate to its child transactions Figure 2 on page 18 illustrates that CICS has called several external services at the same time. The wait time is reduced, and the program can continue processing until responses are received.

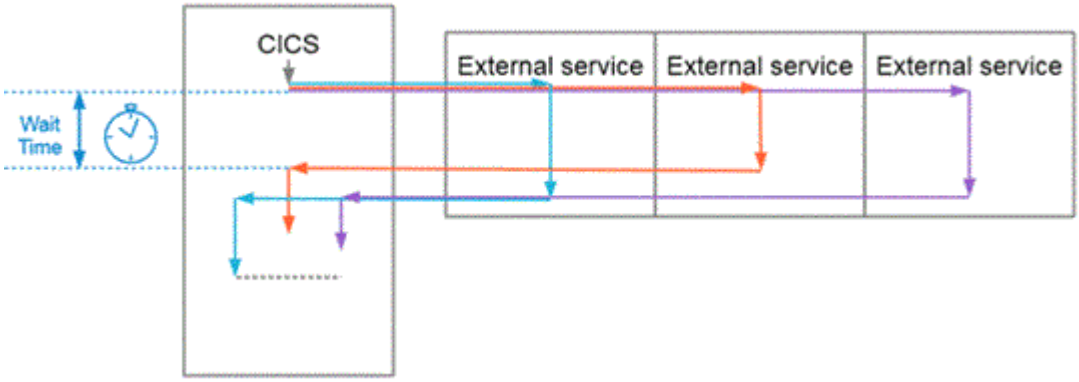


Figure 2. Calling services in parallel (asychronous processing)

Where is asynchronous processing used?

Asynchronous processing is applicable to any situation in which you don't want to halt processing while an external request is being processed.

Related links:

- Developing for asynchronous requests
- Troubleshooting the Asynchronous API

Benefits of using the asynchronous API

Asynchronous request processing was possible before CICS TS 5.4, but the new asynchronous API commands can simplify designing and implementing asynchronous processing.

Using the **EXEC CICS RUN TRANSID** and **EXEC CICS FETCH** commands, applications can process the requests, responses, and exceptions involved in asynchronous processing, and still make use of standard CICS commands and coordination, minimizing complexity in both management and execution of programs.

There are three key challenges involved in asynchronous processing, all of which are addressed as part of the new asynchronous API commands:

- Executing work independently following an asynchronous request.
- Tracking the completion status of processes and services running asynchronously.
- Communicating data between asynchronous processes.

Previously, these challenges were addressed by using a combination of other CICS functions, services, or supporting products, which weren't designed to support asynchronous processing on a large, integrated scale. Using the **EXEC CICS START** and **EXEC CICS DELAY** commands, Business Transaction Services (BTS), Event Control Blocks (ECB), or WebSphere® MQ, it was possible to create a functioning asynchronous model, but in an unsupported and typically complex manner.

BTS, for example, works better for long-term units of work, or those that require human intervention to provide meaningful results, while using external products such as WebSphere MQ relieves pressure on CICS system programmers and application programmers, but increases the burden of managing supporting infrastructure.

The **EXEC CICS RUN TRANSID** and **EXEC CICS FETCH** commands are designed to integrate seamlessly with programs and applications that use normal CICS logic – and, in conjunction with the **EXEC CICS PUT CONTAINER** and **EXEC CICS GET CONTAINER**, to simplify data management between asynchronously running processes.

About the asynchronous API

CICS TS provides a set of new asynchronous API commands which allow application developers to use asynchronous programming model that can run child tasks asynchronously.

Elements of the asynchronous API

The asynchronous API consists of the following elements:

- API commands to send requests and receive responses.
- Parent tasks that issue **EXEC CICS RUN TRANSID** and **EXEC CICS FETCH** commands to start and interact with child tasks.
- A **FREE CHILD** command to release the resources associated with a child prior to the parent program completing.
- Child tasks that run asynchronously to the parent task as a separate unit of work. Each parent can have one or more children.
- Channels that can be optionally used to pass data between parent and child tasks if required.

The asynchronous API manages the state of parent and child tasks, cleans up all associated resources, and manages data in channels as standard. Use of the API can have major benefits for historically long-running tasks, such as web services applications, who can instead create and free child tasks as and when required to minimize their system footprint.

How the asynchronous API works

A parent task is any task running a program which uses the **EXEC CICS RUN TRANSID** and **EXEC CICS FETCH** commands to start and interact with child tasks. This example parent program uses the **EXEC CICS PUT CONTAINER** command to create a container and a channel to communicate with the child program, and then creates a child task with the **EXEC CICS RUN TRANSID** command, which will run the child program defined by *transid*. The child program receives a copy of the channel specified and begins processing, leaving the parent program available to continue with any other logic. The **EXEC CICS FETCH CHILD** command can be issued by the parent when it requires the result or completion status of the child task. If the child task has successfully completed and passed a channel back, then the **EXEC CICS GET CONTAINER** command is used to retrieve the output.

```
EXEC CICS PUT CONTAINER(container) CHANNEL(channel)
                    FROM(struct)  FLENGTH(len_struct) BIT
                    RESP(reason) RESP2(response)

EXEC CICS RUN TRANSID(transid) CHILD(child)
            CHANNEL(channel) RESP(reason) RESP2(response)

…

EXEC CICS FETCH CHILD(child)
                  ABCODE(abcode)
                  COMPSTATUS(child_status)
                  CHANNEL(fetch_channel)
                  RESP(reason) RESP2(response)

EXEC CICS GET CONTAINER(container) CHANNEL(fetch_channel)
                      INTO(struct) FLENGTH(len_struct)
                      RESP(reason) RESP2(response)
```

A child program can use CICS API commands as normal, but will typically use the **EXEC CICS GET CONTAINER** and **EXEC CICS PUT CONTAINER** commands to interact with the parent program. This example child program uses the **EXEC CICS GET CONTAINER** command to retrieve the container

passed by the parent program, runs some other logic to modify the contents (`...`), and then uses the **EXEC CICS PUT CONTAINER** command to send the result back to be consumed by the parent.

```
EXEC CICS GET CONTAINER(container) CHANNEL(channel)
              INTO(struct) FLENGTH(len_struct)
              RESP(reason) RESP2(response)

 ...

  EXEC CICS PUT CONTAINER(container) CHANNEL(channel)
              FROM(struct) FLENGTH(len_struct) BIT
              RESP(reason) RESP2(response)
```

Using channels to communicate between parent and child programs is optional. If a parent program only needs to know whether a child task has completed successfully, for example, then the **EXEC CICS FETCH** command is sufficient.

Managing performance with the asynchronous API

The asynchronous API can result in a large number of concurrent tasks within a CICS system. CICS will automatically begin workload management in case a region reaches **MXT**, and you can regulate performance yourself using TRANCLASS. Transaction tracking and statistics are also available to monitor the performance of regions running asynchronous workloads.

Regulating performance

Using the TRANCLASS resource

By specifying the TRANCLASS of parent transactions, you can control the maximum number of parent tasks that will run in a system at any given time, and by extension the number of child tasks that will be created by those parents.

Use the MAXACTIVE attribute of TRANCLASS to ensure that the combined number of parent and child transactions is less than the MXT for your system. The MAXACTIVE value for your child tasks should be higher than the MAXACTIVE value for parent tasks, assuming that a given parent task will create multiple children.

Important: If you do choose to set TRANCLASS for your child transactions, don't use the same TRANCLASS for child transactions and parent transactions, otherwise you can end up with a system full of parent tasks and no space for child tasks to attach

Automatic regulation by CICS system management

If a region becomes overloaded, CICS will automatically start regulating workflow to prevent too many child tasks being created. Parent tasks issuing a **RUN TRANSID** command will be suspended and using the ASPARENT wait type put in a queue, and resumed when workload levels in the region drop.

When parent tasks are resumed, workload in the region may fluctuate, and parent tasks may be suspended and resumed in quick succession. This is the expected behavior, and indicates that automatic system management is properly protecting CICS from excessive asynchronous requests.

If your asynchronous workload is regularly causing problems for your region, consider regulating workload using TRANCLASS as described above.

Monitoring performance

Using previous transaction tracking

Previous transaction data characteristics allow you to track the relationships between tasks in your region. For example, if one of your child tasks is hanging, you can use this data to see which parent task is waiting for a response from that child, and make a decision about how to resolve the problem.

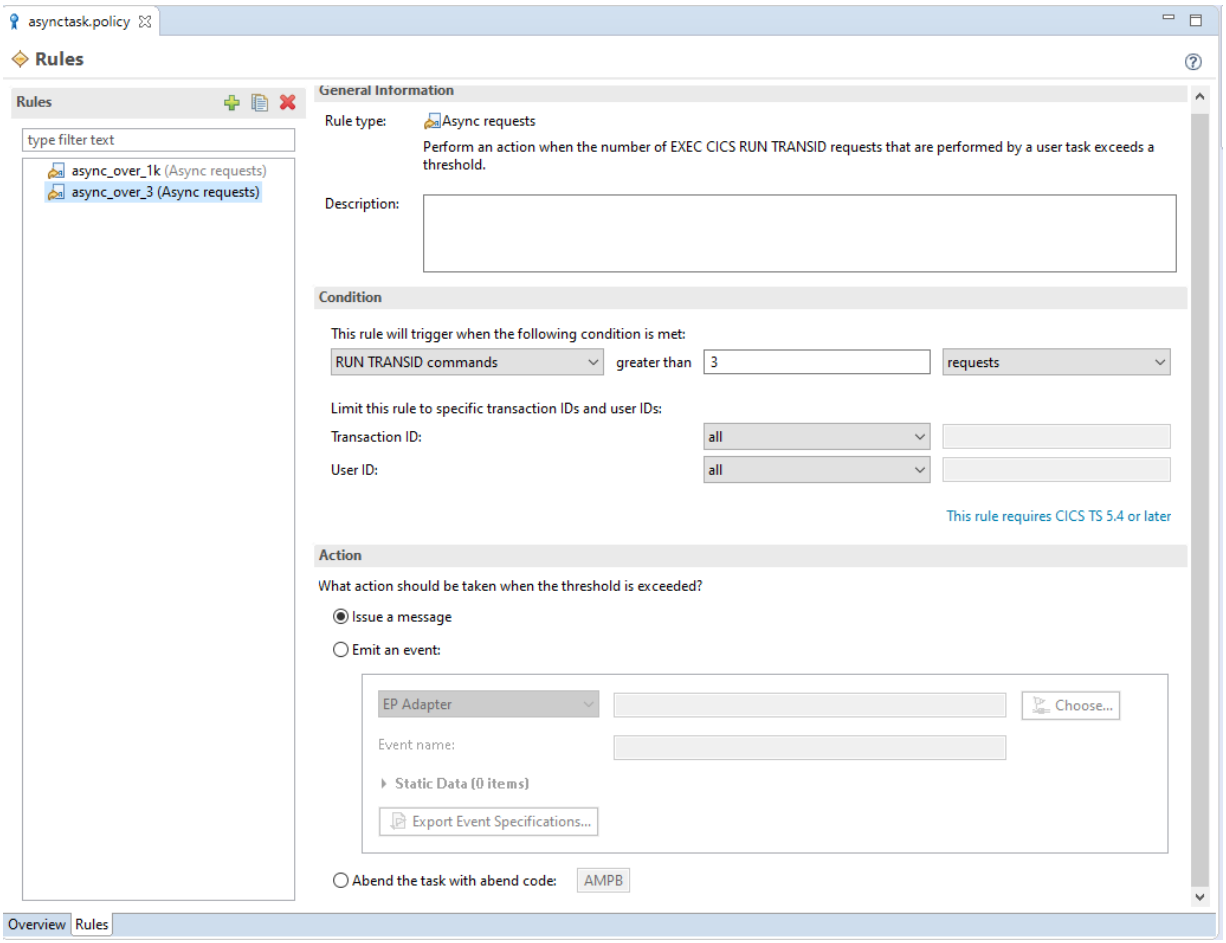
Using statistics and monitoring data

Asynchronous services statistics and Monitoring field data are available to help you monitor and diagnose asynchronous workloads. You can find out how many **RUN TRANSID** and **FETCH** commands were issued by an application, how much time was spent waiting for child tasks to complete, and other useful information to help monitor and improve asynchronous performance.

Using CICS policies

You can also use CICS policies to manage asynchronous workload. Using the `Async requests` rule type, you can choose a threshold for how many **RUN TRANSID** commands can be issued before the policy action is initiated. Using CICS policies, you can gain even finer control over the parent and child tasks in your system.

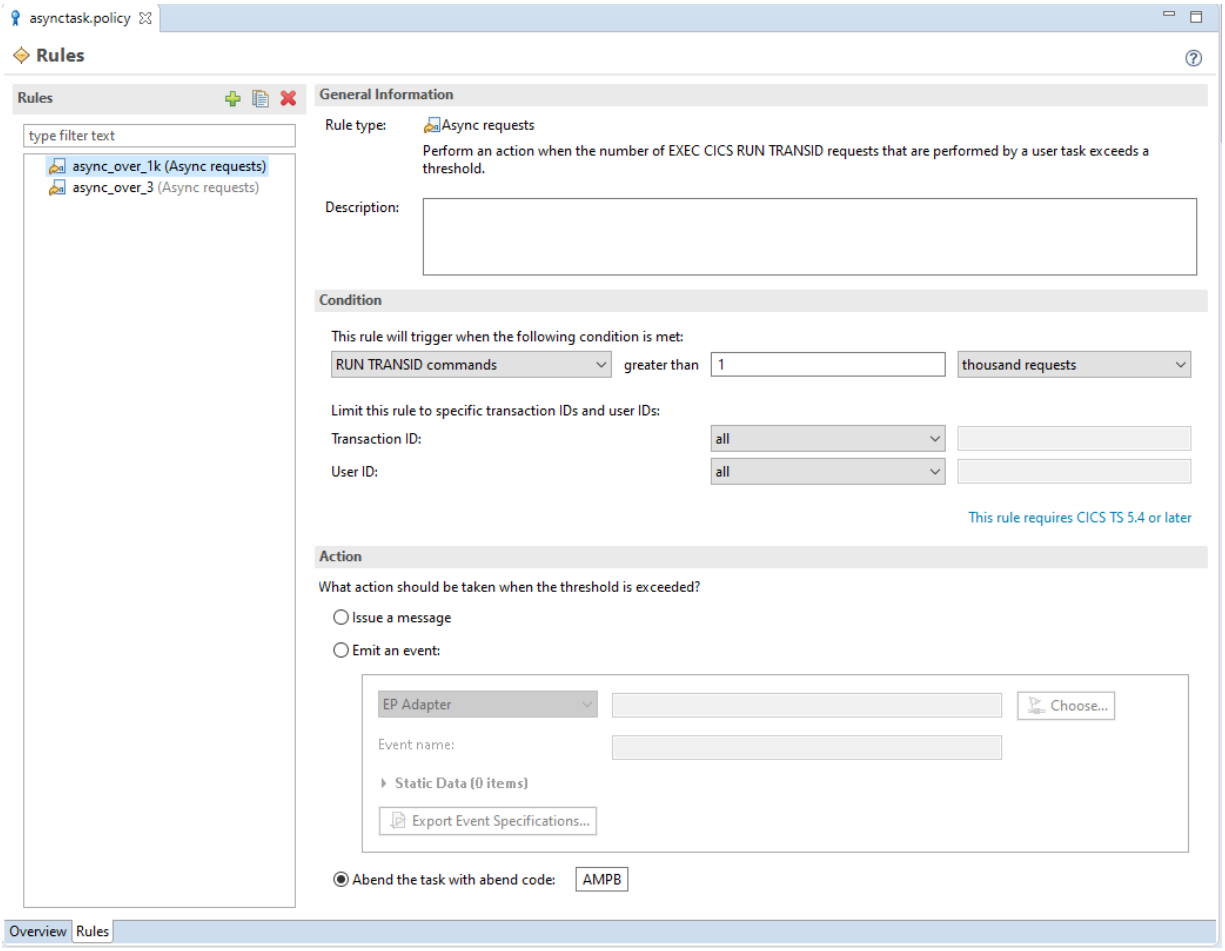
Example: Two Async `requests` task rules



In this example, the `async_over_3` rule specifies that if a task issues more than three **RUN TRANSID** commands, CICS will issue message DFHMP3001.

```
DFHMP3001 13/03/2017 16:11:36 IYCWZCAB Task 00238(PAS2) exceeded a policy threshold.
BundleId=asyncPolicy, PolicyName=asynctask, RuleName=async_over_3, RuleType=asyncrequest,
Category=runtransid, Threshold=3 (Value=3, Unit=), CurrentCount=4.
```

Figure 3. Definition of the rule `async_over_3`



The `async_over_1k` rule, on the other hand, will abend a task if the task issues more than a thousand **`RUN TRANSID`** commands. CICS will issue message DFHMP3002 along with the abend code AMPB. Abends caused as a result of policy rules can be handled just like any other abend.

```
DFHMP3002 13/03/2017 16:11:36 IYCWZCAB Task 00238(PAS2) exceeded a policy threshold and is
abended with abend code AMPB.
BundleId=asyncPolicy, PolicyName=asynctask, RuleName=async_over_1k, RuleType=asyncrequest,
Category=runtransid,
Threshold=1000 (Value=1, Unit=K), CurrentCount=1001.
```

Figure 4. Definition of the rule `async_over_1k`

Using JCICS with the asynchronous API

The JCICS variant of the CICS asynchronous API is provided as a set of classes, and an implementation of the `java.util.concurrent.Future` interface.

Java variants of all the **`EXEC CICS`** asynchronous API commands are provided in the following classes:

- `AsyncService`
- `AsyncServiceImpl`
- `ChildResponse`

To start a child transaction from a Java program, you must first get a new instance of `AsyncService` before starting the child transaction. No state is persisted in the `AsyncServiceImpl` class, so Future objects are not scoped to their instance of `AsyncService`.

```
AsyncService asService = new AsyncServiceImpl();
Future<ChildResponse> child = asService.runTransactionId("ABCD");
```

The resulting `Future` object can be used as normal, although methods that relate to the canceling of child tasks are not supported.

`runTransactionId()` is non-blocking, but `get()` is designed to comply with the Future interface and will block.

Unlike the CICS API implementation, the `get()` or `getAny()` invocation always attempts to retrieve a channel from the child, and ownership of the channel will be immediately transferred to the parent task.

To receive information back from a child task, you can use the `get()` method. For example:

```
ChildResponse response = child.get();
```

`ChildResponse` provides the following methods:

- `getAbendCode()`
- `getChannel()`
- `getCompletionStatus()`

`ChildResponse` also includes an enum, `CompletionStatus`, to enumerate the possible CVDA values returned by the child task.

Table 2.

Java Class	Method	Equivalent CICS API
AsyncService	runTransactionId(String transactionId) runTransactionId(String transactionId, Channel channel)	**RUN TRANSID** **RUN TRANSID CHANNEL**
AsyncService	freeChild(ChildResponse child) freeChild(Future<ChildResponse> child)	**FREE CHILD** **FREE CHILD**
ChildResponse	ChildResponse.getAny() ChildResponse.getAny(BlockingAction blockingAction) ChildResponse getAny(long timeout, TimeUnit timeUnit)	**FETCH ANY CHANNEL COMPSTATUS ABCODE** **FETCH ANY NOSUSPEND CHANNEL COMPSTATUS ABCODE** **FETCH ANY TIMEOUT CHANNEL COMPSTATUS ABCODE**
Future<ChildResponse>	get() get(long timeout, TimeUnit timeUnit)	**FETCH CHILD CHANNEL COMPSTATUS ABCODE** **FETCH CHILD CHANNEL COMPSTATUS ABCODE TIMEOUT**
Future<ChildResponse>	isDone()	**FETCH CHILD NOSUSPEND COMPSTATUS**

Example

```
final String childTransaction = "ABCD";
AsyncService asService = new AsyncServiceImpl();

Future<ChildResponse> child = asService.runTransactionId(childTransaction);

// Logic here to be processed while the child runs asynchronously

ChildResponse response = child.get();
if (response.getCompletionStatus().equals(CompletionStatus.NORMAL)) {
  System.out.println("Child task completed normally");
}
```

Example: Running a credit check application with asynchronous processing versus with traditional sequential processing

This example simulates a real-world scenario: a customer applies for a credit card, and a credit check application kicks off to perform checks on the customer. The programs involved in the application will

be running both sequentially and asynchronously. This example illustrates the potential of asynchronous processing and its benefits compared with traditional sequential processing.

Before you begin

The source code for this example is available in full on GitHub, where you'll find two parent programs, ASYNCPNT and SEQPNT, and instructions for running them from a CICS terminal.

About the example

The credit check application checks the customer's name, details and history. Each check has its own program and associated transaction in CICS.

- SEQPNT calls the involved programs using `EXEC CICS LINK`.

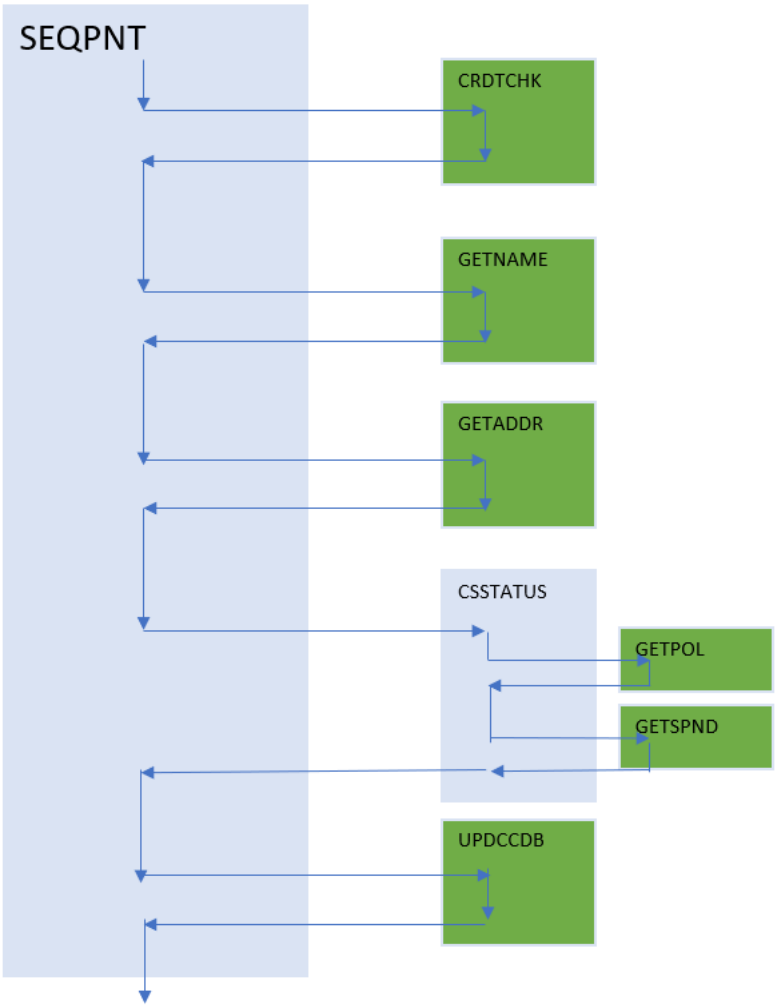


Figure 5. SEQPNT - Sequential processing flow

- ASYNCPNT calls the involved programs using `EXEC CICS RUN TRANSID`.

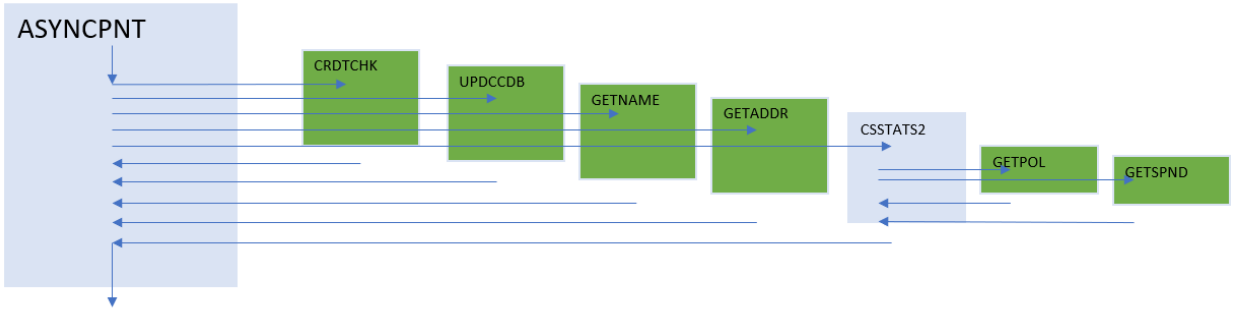


Figure 6. ASYNCPNT - Asynchronous processing flow

Running the example

To run the example, follow the instructions provided with the source code.

Result: The child programs take the same amount of time to complete in both ASYNCPNT and SEQPNT, but ASYNCPNT finishes in roughly half the time thanks to the asynchronous API.

How they differ

Examine in details how asynchronous processing differs from traditional sequential processing:

Getting it done: all at once or bit by bit

The fundamental difference between asynchronous and sequential processing is, simply put, how much you can do at once. Traditional sequential processing only allows you to perform one action at a time; then you have to wait for it to complete (successfully or otherwise) before you can do something else. If one of your actions hangs, or the external service it's calling is being slow, then the delays stack up quickly. The main body of the sequential SEQPNT program works this way, calling programs one by one using `EXEC CICS LINK` and waiting for them to complete before retrieving data from them using `EXEC CICS GET CONTAINER` (the application is using containers to store and retrieve data):

```
EXEC CICS LINK PROGRAM ( CREDIT-CHECK )
               CHANNEL ( MYCHANNEL )
               RESP    ( COMMAND-RESP )
               RESP2   ( COMMAND-RESP2 )
END-EXEC

EXEC CICS GET CONTAINER (CRDTCHK-CONTAINER)
                INTO    (CREDIT-CHECK-RESULT)
                CHANNEL (MYCHANNEL)
                RESP    (COMMAND-RESP)
                RESP2   (COMMAND-RESP2)
END-EXEC
```

And so on for the other programs that need to run, calling and waiting, calling and waiting.

The ASYNCPNT program, however, looks more like this:

```
EXEC CICS RUN TRANSID      (CREDIT-CHECK-TRAN)
              CHANNEL      (MYCHANNEL)
              CHILD        (CREDIT-CHECK-TKN)
END-EXEC

EXEC CICS RUN TRANSID      (GET-ADDR-TRAN)
              CHANNEL      (MYCHANNEL)
              CHILD        (GET-ADDR-TKN)
END-EXEC

EXEC CICS RUN TRANSID      (CSSTATUS-TRAN)
              CHANNEL      (MYCHANNEL)
```

```
                    CHILD       (CSSTATUS-TKN)
          END-EXEC
```

Using the EXEC CICS RUN TRANSID command, multiple programs are kicked off at once, and they'll run as child transactions of the parent program. Child transactions run separately to the parent program, much like real children, allowing the parent program to continue with anything else it wants to do in the meantime. Child tasks will always run locally to the parent task (that is, in the same CICS region).

Getting results

Getting results in a sequential processing model is as simple as waiting, and waiting, and waiting, because your parent program is blocked until its child program completes. In the asynchronous model, the parent program logic continues until it's time to find out what the child programs are up to. Use the **EXEC CICS FETCH CHILD** command to check on the completion status and abend code of any child transaction:

```
          EXEC CICS FETCH CHILD       (CREDIT-CHECK-TKN)
                          CHANNEL     (CREDIT-CHECK-CHAN)
                          COMPSTATUS  (CHILD-RETURN-STATUS)
                          ABCODE      (CHILD-RETURN-ABCODE)
          END-EXEC
```

The credit check application doesn't have any logic that checks if a child transaction completed successfully, which should have been done in real cases, but the response received from a **FETCH CHILD** command includes completion status and abend code, so you can (and should) ensure that your child transaction was happy and successful.

Once the **FETCH CHILD** command has completed, retrieve the results of the child transaction using an **EXEC CICS GET CONTAINER** command, just the same as what would be done in sequential processing:

```
          EXEC CICS GET CONTAINER (CRDTCHK-CONTAINER)
                          INTO    (CREDIT-CHECK-RESULT)
                          CHANNEL (CREDIT-CHECK-CHAN)
                          RESP    (COMMAND-RESP)
                          RESP2   (COMMAND-RESP2)
          END-EXEC
```

Parent and child communication

Passing data between parent and child tasks is handled using channels and their containers. When a child task is started using **EXEC CICS RUN TRANSID**, it will be passed a copy of the channel specified in the command, as well as a copy of all the channel's containers (and the data within them). As it runs, the child task will update the data in its copy of the channel, while the parent program can continue processing using the original channel; this is what allows the two (or more) tasks to run asynchronously.

Sometimes you won't even need to use containers: if all you need to know is the completion status of your child tasks, then the **EXEC CICS FETCH CHILD** command will give you all the information you need.

Chapter 3. Cloud-enabling CICS Transaction Server for z/OS

Cloud is a conceptual shift in how businesses offer services. It drives increased operational efficiency over the management and operation of services and increases agility in developing and deploying them. CICS Transaction Server for z/OS provides three key capabilities as building blocks to transform existing CICS TS topologies and applications into cloud-style platforms and services. These are: *platforms*, *applications* as single entities, and *policies* to control operations. CICS *bundles* also play a part as a way of grouping and managing related resources for use by applications or platforms.

This section introduces the key capabilities of cloud-enabling CICS TS and gives a high-level view of what's involved in building a cloud solution.

What is a platform?

Platforms enable the creation of agile service delivery run times. CICS regions can be grouped as platforms for rapid application deployment, decoupling applications from the underlying topology and increasing flexibility. When you start regions in a platform, applications are deployed to them without any further interaction from a system administrator. Reliability is increased through automatic resource validation, provisioning, and de-provisioning. Platforms can be managed dynamically by applying policies during run time.

Platforms build on the foundations of CICSPlex SM topology definitions, such as CICS system groups (CSYSGRP). The operations to deploy and manage both platforms and applications are built on the single-point-of-control capabilities of CICSplex. You must deploy CICSPlex SM to get full use of CICS cloud enablement features.

For more information, see “How it works: platforms” on page 31.

What is an application?

In the context of cloud, an application combines disparate application resources. As an application, these resources can be managed as a single entity. This entity can be versioned and rapidly moved through the development, test, and production lifecycle. Using applications improves the management of dependencies, and entire applications can be measured for resource usage and internal billing. Applications can be managed dynamically by applying policies during run time.

For more information, see “How it works: applications” on page 46.

What are policies?

You can have automated control over critical system resources with CICS policy task and system rules. For example, task rules can be defined to set thresholds such as the number of file requests performed by a task, storage used by a task, or processor time used by a task. When the threshold is exceeded, one of a number of automated actions can be performed: issue a message, abend the task, or emit an event that can trigger further actions. Policies can be applied dynamically during run time.

For more information, see CICS policies.

What are CICS bundles?

A CICS bundle is a directory that contains artifacts and a manifest that describes the bundle and its dependencies. CICS bundles provide a way of grouping and managing related resources.

CICS bundles also provide versioning for the management of resource updates, and can declare dependencies on other resources outside the bundle. Application developers can use CICS bundles for

© Copyright IBM Corp. 1974, 2023

application packaging and deployment, business events, and services. System programmers can use CICS bundles to define CICS policies.

For more information, see Defining CICS bundles.

Building a cloud solution

Let's look at who is involved in building a CICS cloud solution and the resulting topology. Figure 7 on page 29 shows how different roles work together to build up the solution. Figure 8 on page 30 shows the system topology and the main artefacts that are involved.

- The software architect selects, or designs, the CICS system group to be used for the platform and assesses how to package the components of the application for deployment on the platform.
- The developer works on the application, then packages the application into an application bundle and exports it to the platform home directory that the system administrator sets up on zFS.
- The system administrator configures the platform, creating and securing directories on zFS and also creates and installs any resources that are required but are not packaged with the application. Either the developer or the system administrator packages the application into a CICS bundle, then the system administrator deploys the application to the platform.
- The system administrator can secure a platform and its deployed applications by setting up RACF security profiles in a similar way to security for other CICSPlex SM components.

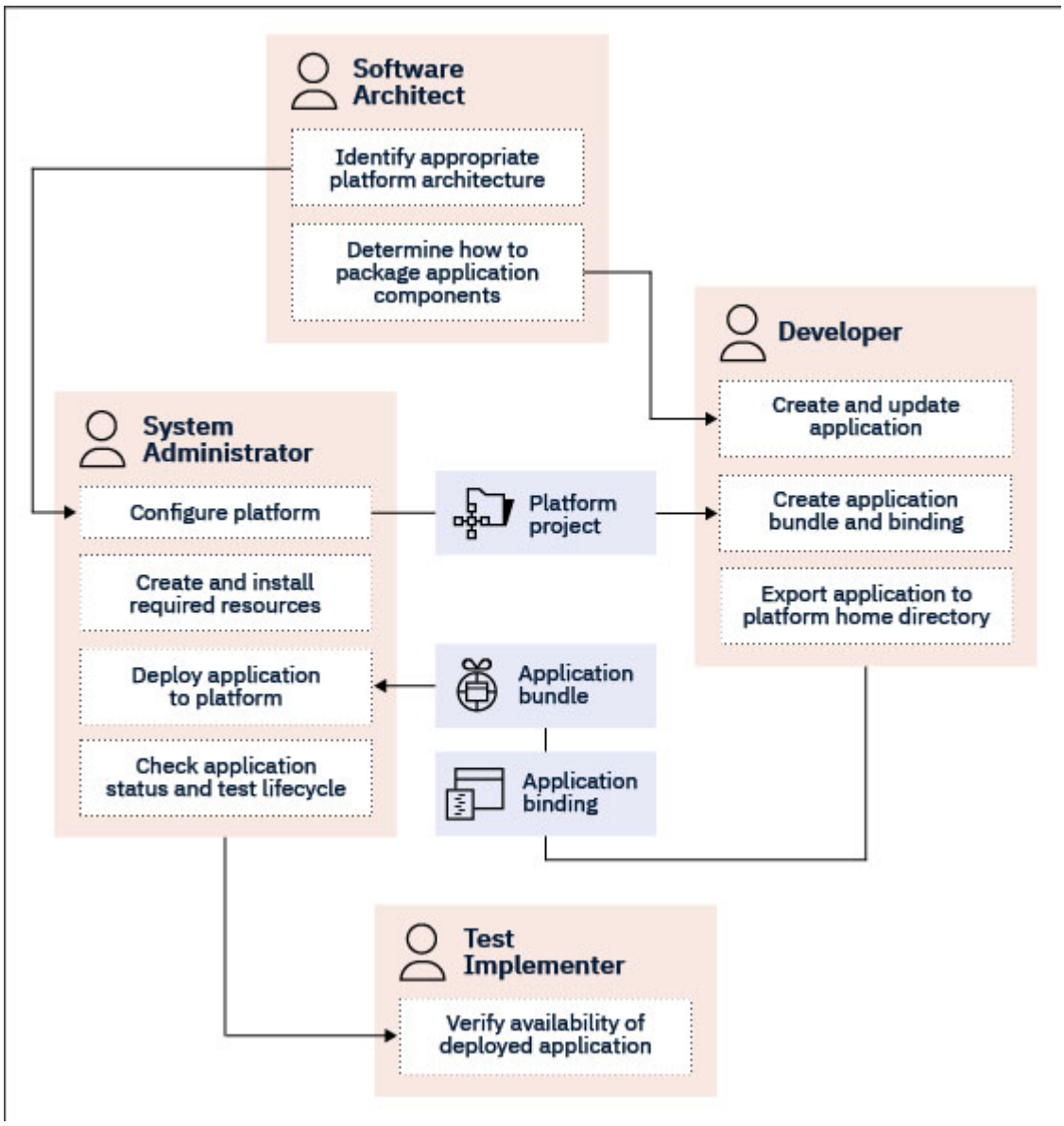


Figure 7. Who is involved in setting up a cloud solution in CICS TS?

The topology below is for one version of an application. If you use multi-versioning, see an equivalent diagram in How it works: multi-versioning applications.

The system administrator used projects in CICS Explorer to define the platform and package the application. The projects are exported to the zFS platform home directory. The system administrator also created PLATDEF and APPLDEF definitions for the platform and application in the CICSPlex SM data repository. The PLATDEF definition is installed in the CICSplex to create a platform with a region type that contains the target CICS regions. The APPLDEF definition is installed in the platform to create CICS bundles for the application in the CICS regions, and the resources that are defined in the bundles are dynamically created in the CICS regions.

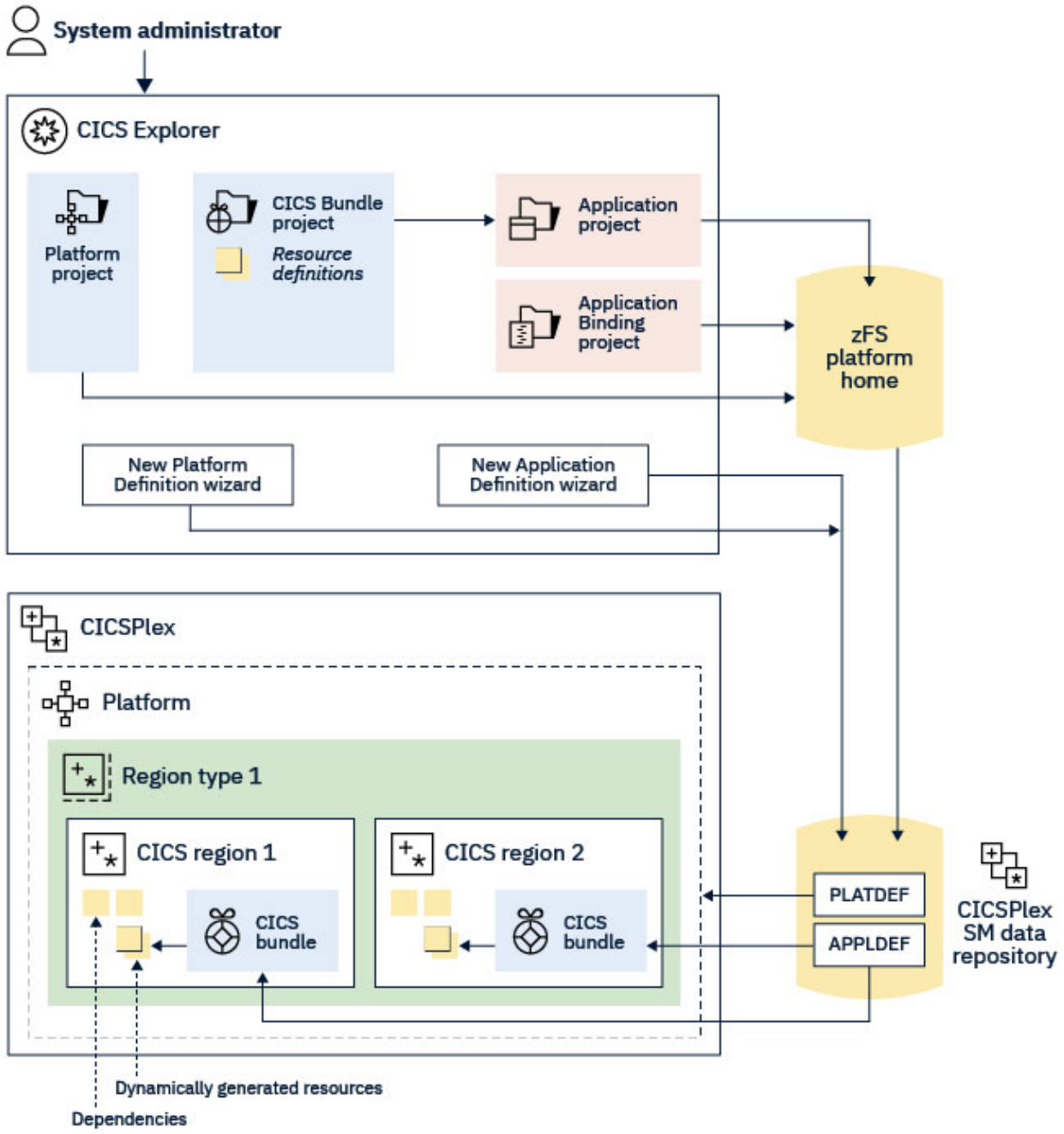


Figure 8. Cloud topology

Platforms and applications: find the information that you'll need

To work with CICS platforms and applications, you'll need the following parts of the IBM Documentation. You might also like to read the IBM Redbooks: Cloud Enabling IBM CICS. It uses the CICS sample General Insurance Application (GENAPP) in examples of moving an existing COBOL-VSAM application to CICS cloud capabilities.

Table 3. Where's the information about CICS platforms and applications?	
Topic	**Where to look**
Find out how platforms and applications work	• How it works: applications • How it works: platforms • How it works: multi-versioning applications
Set up a platform	Setting up a platform
Set up an application	Setting up an application
Secure your platform or application	Security for platforms and applications
Deploy a platform	Deploying a CICS platform
Deploy an application	Deploying an application to a platform
Manage a platform	Managing platforms
Manage an application	Managing applications
Troubleshoot issues with platforms or applications	Troubleshooting platforms, applications, and policies
Find an example	IBM Redbooks: Cloud Enabling IBM CICS

How it works: platforms

A platform is a CICS resource that provides a layer of abstraction to decouple applications from the underlying region topology. A platform contains one or more *region types*, each of which contains one or more CICS regions. Platforms can both create and reuse existing region definitions. An application can be dynamically deployed to the relevant region types within a platform, and removed again when no longer needed. Platforms can be used to deploy policies, to limit and control resources, and to provide application separation.

The region is the basic unit of server deployment in a CICS TS installation. CICS TS regions are IBM z/OS address spaces that are defined to run as jobs or started tasks. They are identified by various attributes, including their job names, their IBM VTAM® applid, or their CICS TS SysID, according to context. Regions require various dedicated resources, such as data sets.

As workloads grow and the number of regions increases, the functions of a single region must be provided by groups of regions for scale and availability. These regions are clones of the original single region. It makes sense to operate on these clones as a single group. Each action that is performed against the group is performed against all regions in that group. Inside a platform, a group of cloned regions that provides the same capability is called a *region type*.

Using platforms, you can:

- Manage system characteristics, such as the status of platform services and declared dependencies, from a single point
- Host applications and platform services
- Provision policies to all applications that are installed in the platform
- Dynamically add and remove services and dependencies while a platform is installed
- Share systems between region types and platforms in a controlled way
- Scale hosted applications while the platform is already installed and without changes.

This section introduces “Components of a platform” on page 32 and “Platform examples” on page 33 and “How do I set up a platform?” on page 41

How is a platform different to a CICSplex?

Before CICS TS V5.1, you could use only a CICSplex or a CICS system group to mark the boundaries of your infrastructure. Platforms allow you to specify a more descriptive and more distinct boundary.

A CICSplex supports separation of environments, such as production and test or different business units. Within a CICSplex, you can define multiple platforms. Compared to a CICSplex, the platform follows a more descriptive route, providing a set of region types to encapsulate the regions within the scope of the platform. Also, platforms support sharing of regions between region types. You can use platforms to separate your concerns within a CICSplex. For example, in a development CICSplex, you can choose to allocate a platform per developer. In a test and production CICSplex, you can choose to have a separate application for each line of business. If you want to consolidate regions, you can take advantage of a platform's ability to share regions with other platforms. For example, you can choose to combine your routing regions for a subset of platforms within the same CICSplex.

Consider the following when you choose a CICSplex or a platform to scope your regions:

- If the regions must be isolated from other regions because they are for a different part of the business or, for example, for development and test regions, a CICSplex is usually a good option. Using a separate CICSplex means that changes made in one scope do not affect the other.
- If the regions don't have to be isolated, platforms might be a better choice. Platforms can also share resources and regions with other platforms, but only where requested.

Components of a platform

A platform is composed of:

- The platform resource itself
- Region types
- A PLATDEF resource definition
- Optionally, CICS bundles to define resources that will be supplied to the applications that run on the platform
- Optionally, policies to control the execution of tasks that run in the context of the application in that platform.

The deployed CICS platform artefacts are located in a directory in zFS. See "Platform directory structure on zFS" on page 43 for details.

You use CICS Explorer to create and manage the components of the platform. Figure 13 on page 42 shows the relationship between these components. You can secure a platform and its deployed applications by setting up RACF security profiles in a similar way to security for other CICSPlex SM components. For information, see Security for platforms and applications.

Platform

A platform bundle, defined in CICS Explorer, describes the platform and its region types and references the CICS bundles that are deployed at the platform level. When you install a platform definition, a PLATFORM resource is created to represent the platform bundle and you can use this resource to remove a platform as a single entity.

The platform resource reflects the current state of the platform. It determines both the overall activity of the regions within the region types, and the overall state of installed platform services. For example, a platform is ACTIVE if at least one region in each region type is active and connected to CICSPlex SM.

The platform also tracks the services that it manages, and any declared dependencies. Platform services and dependencies are installed as CICS bundles, and the platform reflects the health of these bundles with its property ENABLESTATUS. For example, when all bundles are enabled across the platform region types, the platform states that it is ENABLED.

Applications require protection from environment change, and this protection is reflected in the lifecycle of the platform. After a platform is installed, it cannot be discarded until all applications

installed within it are also discarded. While the platform is installed, the region types cannot be modified or discarded but regions can be added and removed from region types. Platform services also follow the lifecycle of the platform. When platforms are installed, enabled, disabled, or discarded, all defined services take the same action.

Region types

A platform includes one or more region types. Region types are used to classify and contain CICS regions according to their type. For example, all CICS regions that handle connections to Db2 could belong to the same group.

Region types define the interface to the platform that applications must be bound to before deployment. When a platform is installed and the region type is created, it creates the region definitions for all defined regions. Alternatively, a region type can ruse an existing environment by adopting an existing CICS system group (CSYSGRP) and the regions in it.

You can clone certain region attribute values for all the CICS regions in a created region type by specifying the attributes at a region type level. Only CICS regions whose definitions have the same values specified for those attributes, or have no values specified for those attributes, can be part of that region type.

Region types can share regions between each other in the same or different platforms. This ability is valuable for consolidation scenarios where you might have a group of routing regions that service several different platforms, or a file-owning region that hosts files for each platform.

Platform (PLATDEF) resource definition

The platform definition, which is a PLATDEF resource definition in the data repository for the CICSplex, identifies the target CICSplex for the platform. On installation of the PLATDEF, the platform is created with a region type that contains the target CICS regions.

CICS bundles

Platforms can install and manage resources to be supplied to the application. For example, a platform might provide a TCP/IP service that the application can consume by using a URIMAP. Platform services and dependencies are installed in CICS bundles. All services that are installed in this manner can be added, enabled, disabled, and removed from the platform through the ADDBUNDLE and REMOVEBUNDLE actions.

After an ADDBUNDLE action is issued, the bundle is installed into all regions within the region type. This bundle is then enabled if the platform was previously enabled. The REMOVEBUNDLE action reverses this process by disabling and discarding the bundle in all regions within the region type.

The relationship between the platform and each installed CICS bundle is saved in a management part. The management part is a MGMTPART record that is created automatically for each CICS bundle during the platform installation process. The management part records the CICS regions where the bundle is installed, and tracks the status of the bundle in the CICS regions.

Platform services don't have to be provided by the platform itself. Platforms can import resources into bundles. Bundle imports declare dependencies on services that are provided outside the platform. For example, if multiple platforms need to share a file, one platform could create the file definition in a bundle, while the other platforms import that file definition into their bundles.

Policies

A platform can also install CICS TS policies. Policies provide a contract for the execution of tasks that run in the context of any application within that specific platform. For example, a policy can force an abend if the task exceeds the contracted processor consumption. Like resource definitions, policies and their associated rules are installed in CICS bundles.

For more information, see CICS policies.

Platform examples

All managed regions within a CICSplex can be defined to a single platform, or managed regions within a CICSplex can be spread across multiple, detached, or overlapping platforms (as a result of the same CICS region being present in more than one platform).

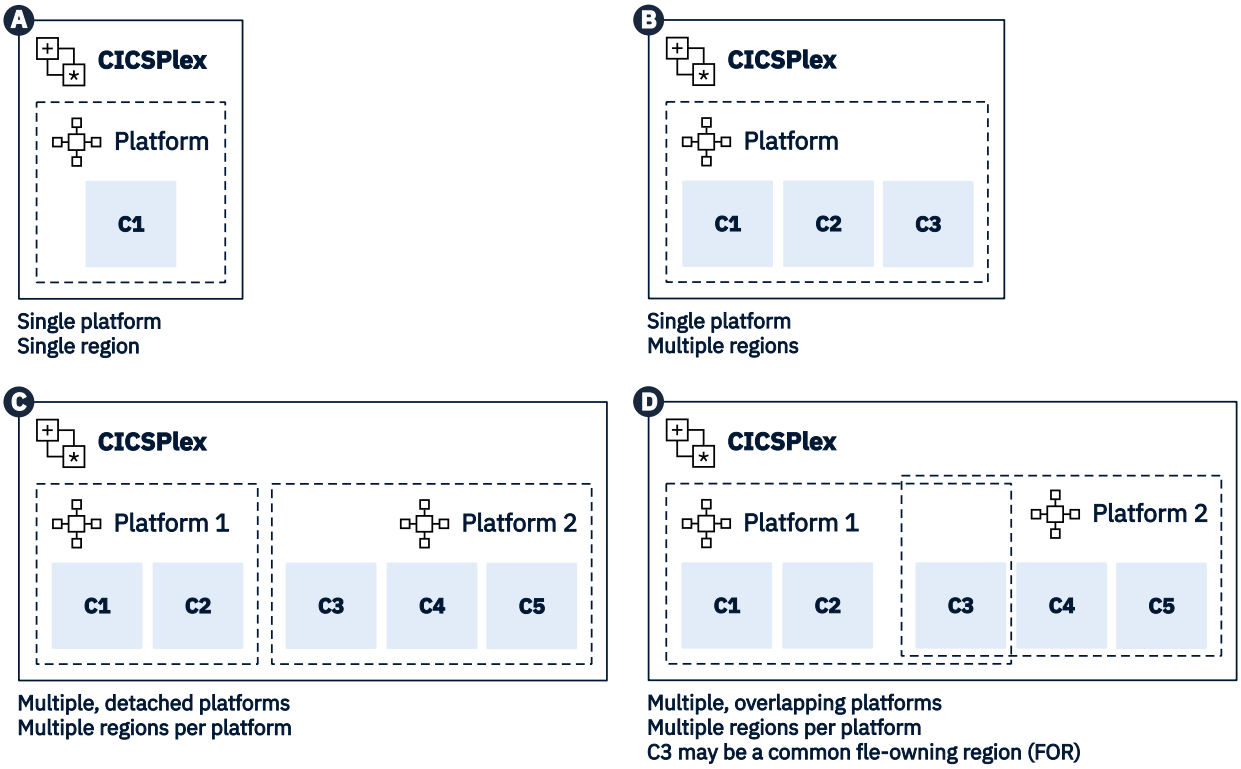


Figure 9. CICS platform architecture examples

In the next example, the CICS regions that are Managed Address Spaces (MASs) in the platform are a web-owning region (WOR), an application-owning region (AOR), and a file-owning region (FOR). The CMAS is a CICSPlex SM MAS that manages not just the WOR, AOR, and FOR, but also the platform.

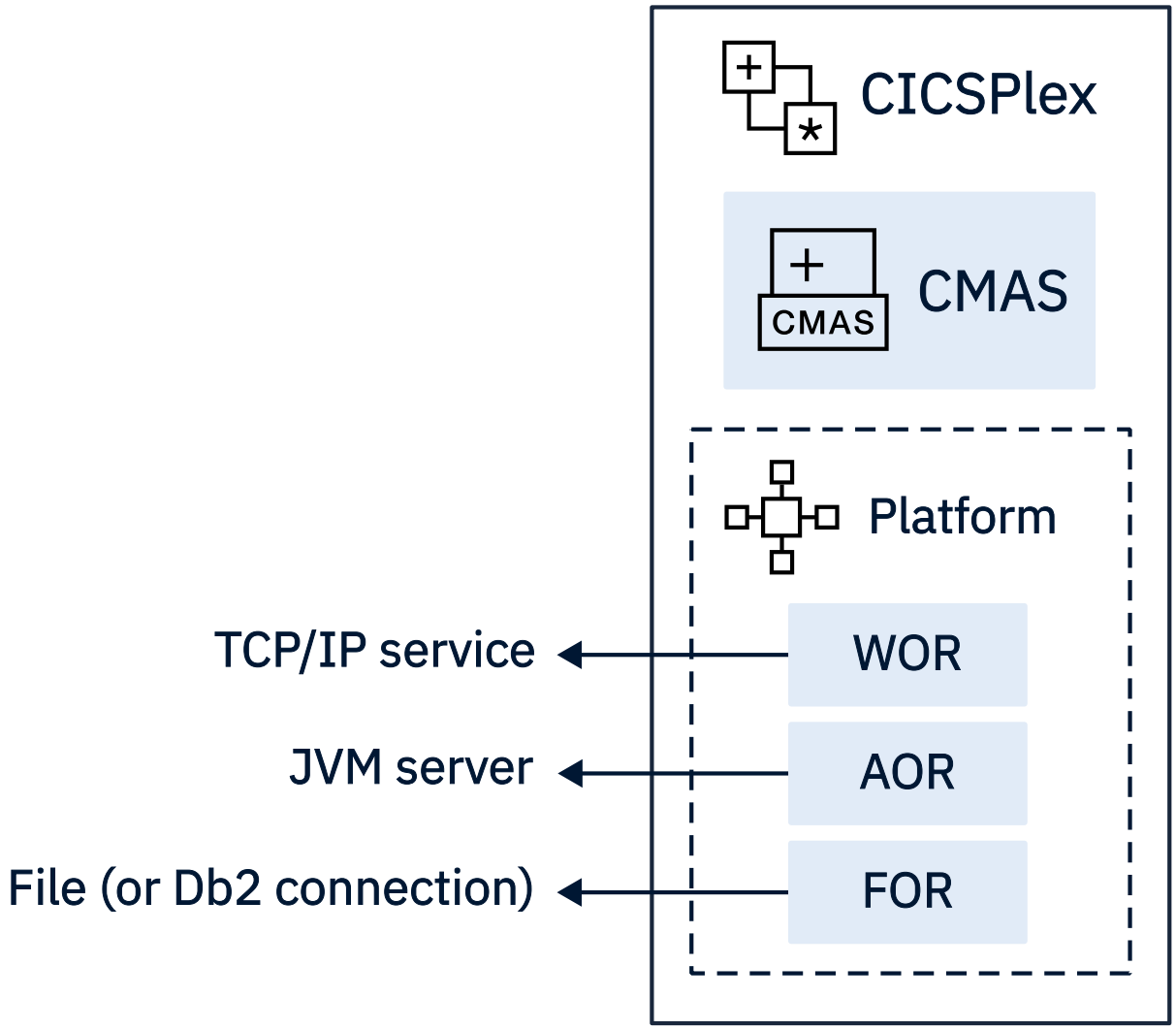
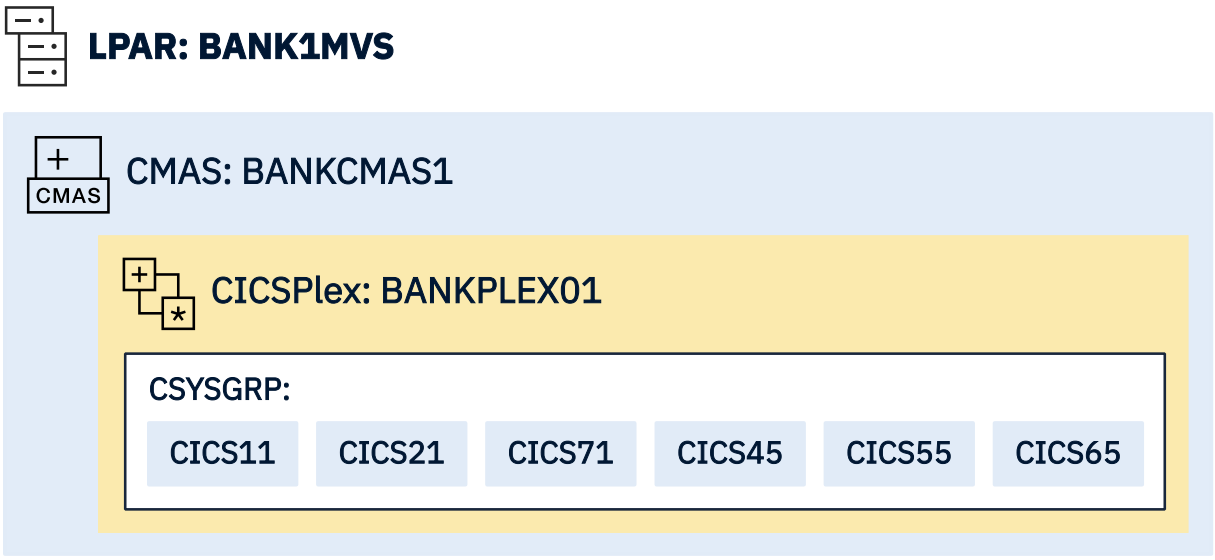


Figure 10. Platform capabilities

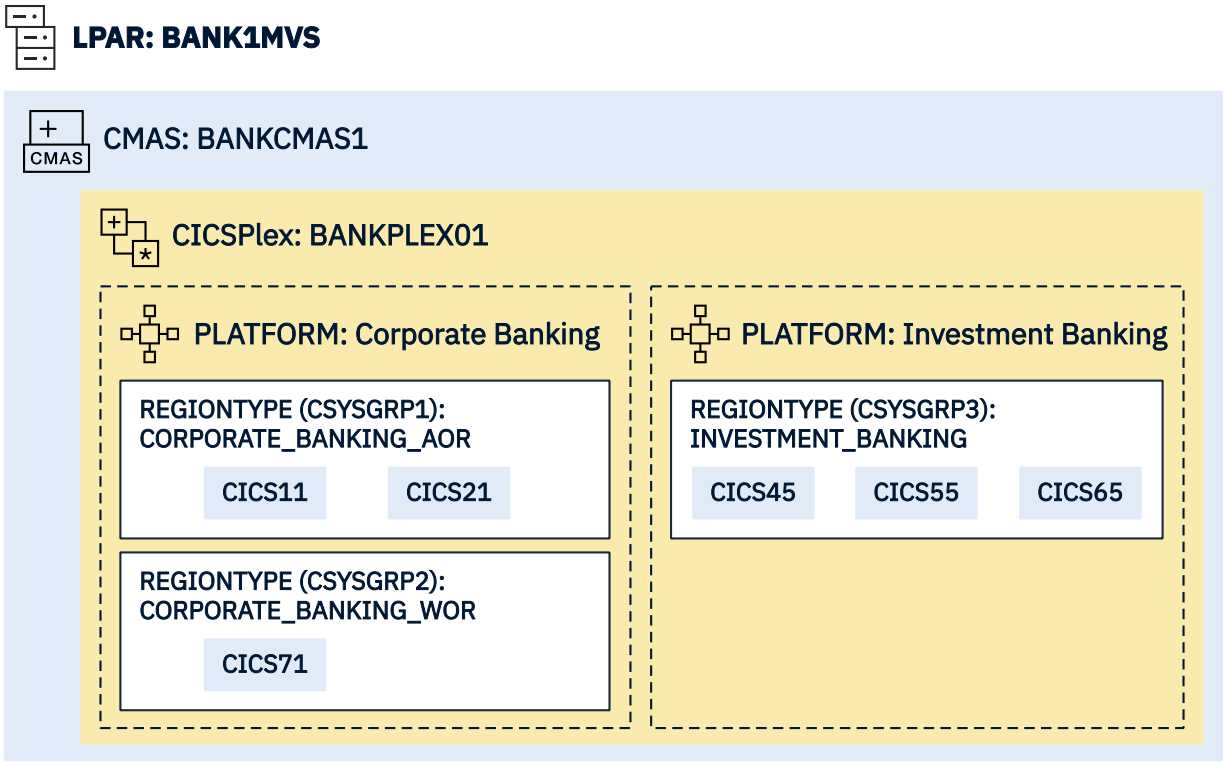
You can arrange platforms as logical entities to achieve application separation. Consider this simplified view of a banking CICSplex:

Figure 11. An example banking CICSplex


LPAR: BANK1MVS
CMAS
CMAS: BANKCMAS1
CICSPlex: BANKPLEX01
CSYSGRP:
CICS11
CICS21
CICS71
CICS45
CICS55
CICS65

This banking CICSplex could be divided into multiple platforms to provide environments for hosting corporate banking and investment banking applications:

Figure 12. An example banking CICSplex divided into multiple platforms


LPAR: BANK1MVS
CMAS
CMAS: BANKCMAS1
CICSPlex: BANKPLEX01
PLATFORM: Corporate Banking
REGIONTYPE (CSYSGRP1):
CORPORATE_BANKING_AOR
CICS11
CICS21
REGIONTYPE (CSYSGRP2):
CORPORATE_BANKING_WOR
CICS71
PLATFORM: Investment Banking
REGIONTYPE (CSYSGRP3):
INVESTMENT_BANKING
CICS45
CICS55
CICS65

How do I set up a platform?

At a high-level, you'll go through the steps listed below. Figure 13 on page 42 shows what you'll end up with.

1. Design the platform. You consider what applications, policies, and resources that you want to deploy on the platform and whether you are creating new CICS regions and region types or adopting existing CICS regions and system group definitions. For more information, see Designing a CICS platform.
2. In zFS, configure your platform home directory. For more information, see Preparing zFS for platforms.
3. In CICS Explorer, create a platform project. To this project, you add regions types, specify any CICS bundles to deploy and the regions types where they will be deployed. For more information, see Setting up a platform.
4. From CICS Explorer, export the platform project from to zFS. The export process packages the CICS bundles that are referenced in the CICS platform project, then exports all the files for the platform bundle and the CICS bundles to the platform home directory in zFS. For more information, see Deploying a CICS platform.
5. In CICS Explorer, create a platform definition. The platform definition is a CICSPlex SM PLATDEF resource definition, which points to the platform bundle in the platform home directory in zFS, and identifies the target CICSplex for the platform. For more information, see Deploying a CICS platform.
6. For each CICS region definition that you created in a region type in your platform project, set up an actual CICS region. For more information, see Deploying a CICS platform.
7. In CICS Explorer, install the platform definition into the CICSplex where you want to run the platform. CICSPlex SM uses the information in the platform bundle to install the platform in the target CICSplex, along with any CICS bundles that are installed with the platform. For more information, see Deploying a CICS platform.
8. Start your CICS regions.
9. If you have any CICS bundles deployed with the platform, in CICS Explorer, enable them so that they are available to the platform.

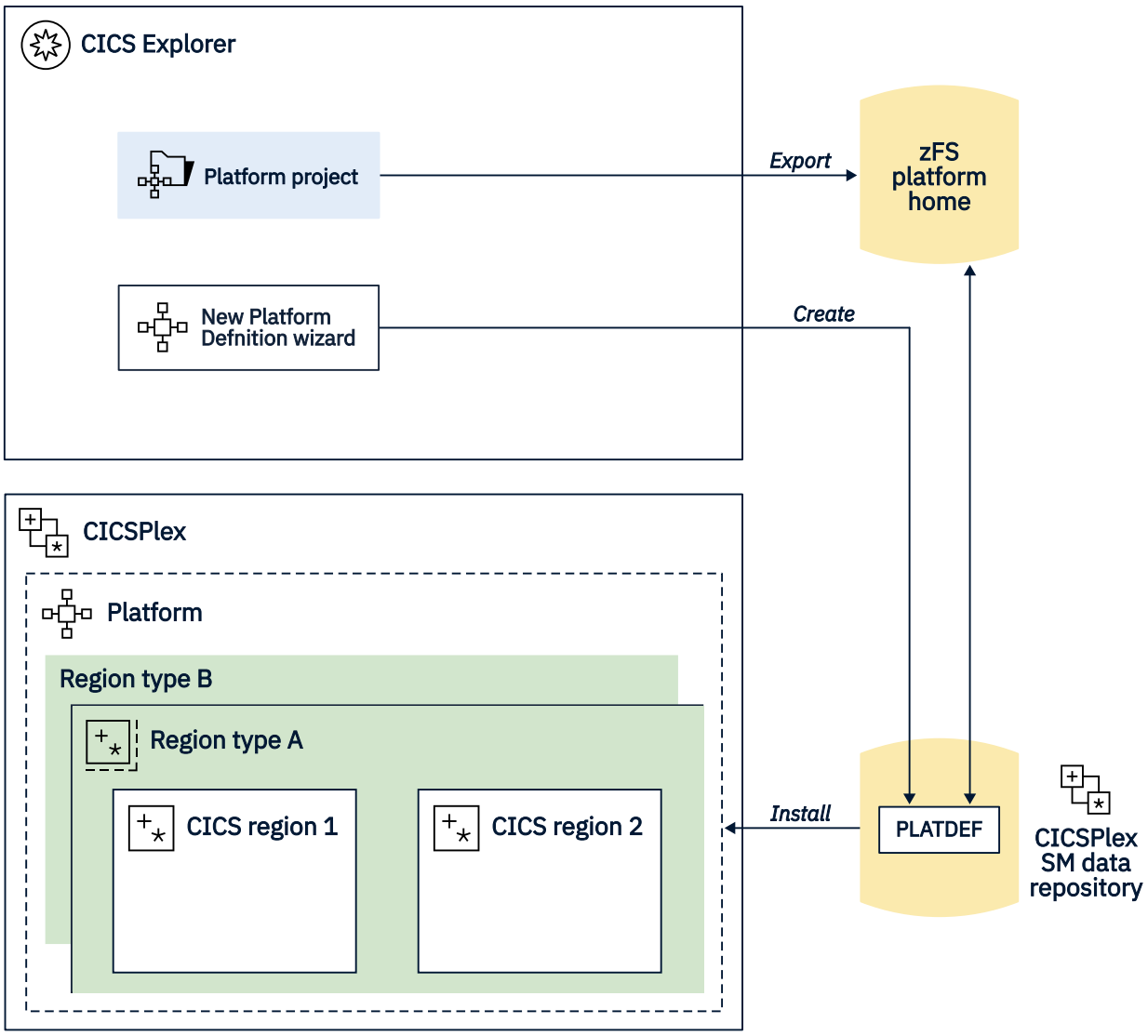


Figure 13. Creating the elements of a platform

Platform directory structure on zFS

The default location for deployed CICS platform artefacts is the directory `/var/cicsts/`*`CICSplex`*`/`*`platform1`*`/`, where *`CICSplex`* is the name of the CICSplex where the platform is installed, and *`platform1`* is the name of your platform. This location is called the platform home directory.

The platform home directory contains a standard directory structure, with directories for the platform bundle itself, and for the applications, application bindings, and CICS bundles that are associated with the platform. For instructions to create a platform home directory, see Preparing zFS for platforms.

The following example shows the standard structure for the platform home directory.

```
/var/cicsts/CICSPLX1/
                    /platform1/applications/..
                              /bindings/..
                              /bundles/..
                              /platform/platform1
                    /platform2/..
           /CICSPLX2/..
```

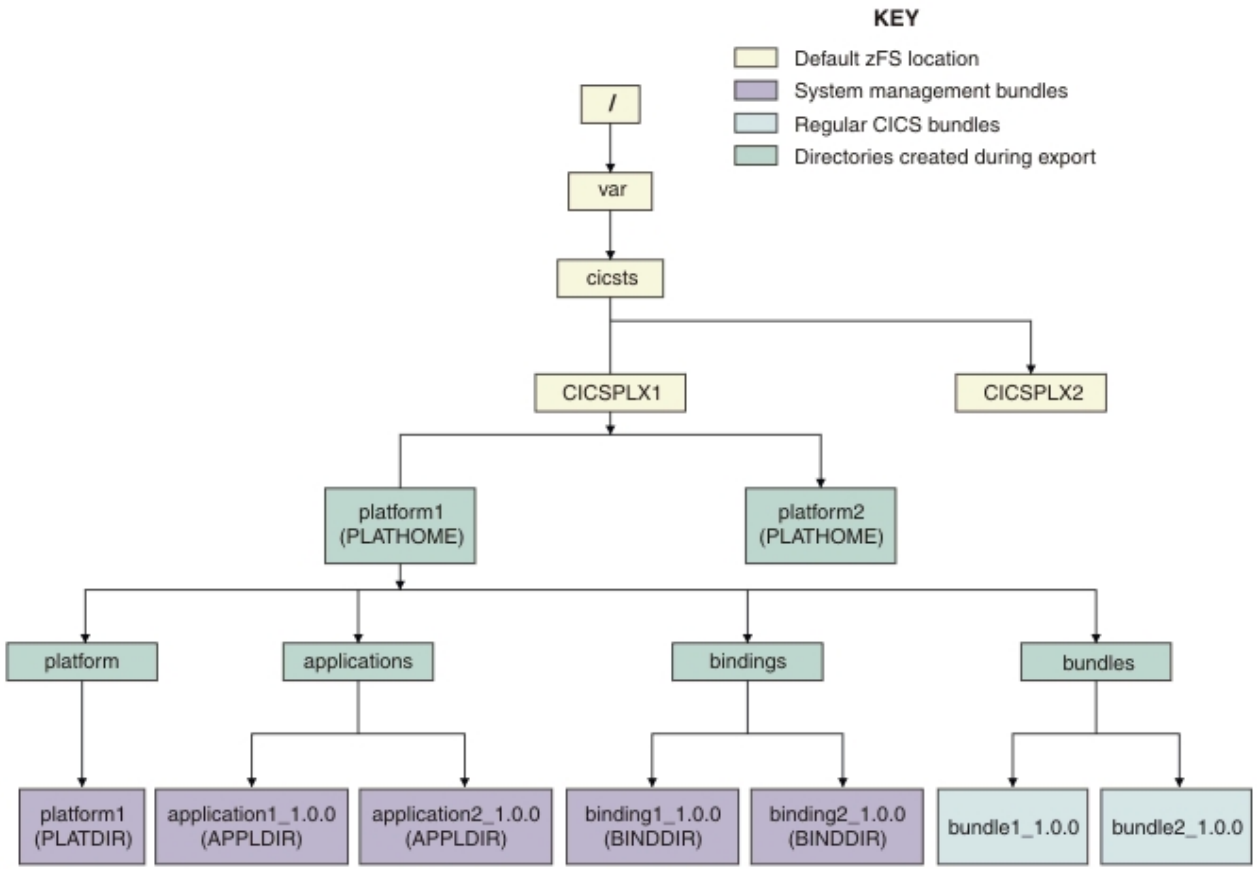


Figure 14. Standard platform home directory structure

Platform home directory (PLATHOME)

The platform home directory (PLATHOME) is the directory in zFS where application bundles, application binding bundles, and CICS bundles are stored in subdirectories, ready to be deployed. The platform bundle is also stored in a subdirectory of this directory. Each platform has its own home directory.

The default platform home directory is `/var/cicsts/`*`CICSplex`*`/`*`platform1`*`/`, where *`CICSplex`* is the name of the CICSplex where the platform is installed, and *`platform1`* is the name of your platform. Keep this default as a best practice. If you need to use a different directory as the platform home directory, you must change the platform bundle to specify the alternative directory name using the platform descriptor editor after you create the CICS Platform project.

In the platform home directory, a number of different directories hold the different resources. These directories are exported from CICS Explorer:

- `applications` contains application bundles.
- `bindings` contains application bindings for this platform.
- `bundles` contains CICS bundles. These bundles might be used in applications or as part of the platform.
- `platform` contains the platform bundle.

Platform directory (PLATDIR)
The platform directory (PLATDIR) is the location of the platform bundle. It contains the exported platform project. The exported platform project contains the `META-INF` directory and subdirectories containing the `bundles.xml`, `deployment.xml`, `manifest.xml`, `platform.xml`, `regions.xml`, `regionTypes.xml`, and `regionTypeLinks.xml` files.

The platform directory is found at `/var/cicsts/`*`CICSplex`*`/`*`platform1`*`/platform/`*`platform1`*.

Any bundles deployed as part of the platform are copied to the `/bundles` subdirectory.

Applications directory
The applications directory is the location of the CICS application bundles. It is created during platform export from CICS Explorer. It contains one directory for each deployed application (APPLDIR). This contains the `META-INF` directory and subdirectories containing `applications.xml`, `bundles.xml`, and `manifest.xml` files.

The applications directory is found at `/var/cicsts/`*`CICSplex`*`/`*`platform1`*`/applications/`.

Any bundles deployed as part of an application are copied to the `/bundles` subdirectory.

Bindings directory
The bindings directory is the location of the application bindings. It is created during platform export from CICS Explorer. It contains one directory for each deployed application binding (BINDDIR). This contains the `META-INF` directory and subdirectories containing `appbinding.xml`, `bundles.xml`, `deployment.xml` and `manifest.xml` files.

The bindings directory is found at `/var/cicsts/`*`CICSplex`*`/`*`platform1`*`/bindings/`.

Any bundles deployed as part of an application binding are copied to the `/bundles` subdirectory.

Bundles directory
The bundles directory is the location of the CICS bundles. It is created during platform export from CICS Explorer. It contains CICS bundles exported from CICS Explorer. These bundles might be used in applications or as part of the platform.

The bundles directory is found at `/var/cicsts/`*`CICSplex`*`/`*`platform1`*`/bundles/`.

Platform states

CICS Explorer shows the state information for a platform. The STATUS setting for a platform is derived from the status of the CICS regions in the region types. This value shows whether the platform is active, partially active, or inactive, depending on whether or not the CICS regions are active. The ENABLESTATUS setting for a platform is derived from the status of the management parts for the CICS bundles that are installed with the platform. This value shows whether the CICS bundles that were installed with the platform are present and enabled in the CICS regions for the platform.

A management part is a MGMTPART record that is created automatically during the platform installation process. Management parts record the relationship between the platform and each installed CICS bundle, and the region type where each CICS bundle is installed in the platform.

Follow the CICS Explorer instructions in Checking the status of a platform in the CICS Explorer product documentation. The tables below show the possible values for the status of the platform and the associated management part.

Table 4. Platform states that reflect the states of the CICS regions associated with the platform

STATUS value	**Meaning**
ACTIVE	There is at least one active CICS region within every region type associated with the platform.
PARTIAL	At least one but not all region types have no active CICS regions or regions that are able to manage platform resources.
INACTIVE	All region types have no active CICS regions that support the PLATFORM object.

Table 5. Platform states that reflect the enable status of the platform

ENABLESTATUS value	**Meaning**
ENABLING	All the management parts for the platform are in the process of being enabled.
ENABLED	All the management parts for the platform are enabled.
DISABLING	All the management parts for the platform are in the process of being disabled.
DISABLED	All the management parts for the platform are disabled.
SOMEDISABLED	Some of the management parts for the platform are disabled.
INSTALLING	The platform is being installed, and it cannot be enabled or disabled at this time.
DISCARDING	The platform is being discarded, and it cannot be enabled or disabled at this time.
FAILED	A problem occurred during install or discard of the platform.
INCOMPLETE	Some of the management parts for the platform are empty or have an invalid scope.
EMPTY	None of the management parts for the platform are installed.

Table 6. States for the management parts for the CICS bundles that are installed with the platform

Status value	**Meaning**
ENABLING	The CICS bundle is in the process of being enabled.
ENABLED	The CICS bundle is installed and enabled in all of the active CICS regions.
DISABLING	The CICS bundle is in the process of being disabled.
DISABLED	The CICS bundle is disabled in all of the CICS regions.
SOMEDISABLED	The CICS bundle is disabled in some of the CICS regions.
IMPORTONLY	The CICS bundle is installed and enabled in all of the CICS regions, but it only contains import statements, so does not affect the status of the platform.
INCOMPLETE	The CICS bundle is installed in some, but not all, of the CICS regions.
INVALIDSCOPE	The CICS system group specified for installing the CICS bundles does not exist, so no bundles are installed.
EMPTY	The CICS bundle is not installed in any of the CICS regions.

How it works: applications

The large set of resources that make up a business application in CICS® can be logically defined as a single entity and deployed on a platform as a single resource. An application that is defined in this way can then be managed as a single entity throughout its lifecycle, making CICS application management faster, easier, and less prone to error.

Applications provide a powerful and capable container around CICS TS bundled resources. These resources track the application lifecycle, from installation to eventual discard. They can host resources, application dependencies, and policies. Their overall health can be tracked through the application's status. Additionally, the application can provide entry points, whose declared operations, for example querying customer or authorizing payment, can be used for simple and accurate resource monitoring and billing through System Management Facilities (SMF) records. CICS policy that defines task rules can also be deployed as part of an application to take effect only on those tasks that are related to specific operations.

Applications support multi-versioning, enabling you to run multiple versions of an application (for example, the current version and a newly-patched version) at the same time on the same platform instance. Applications can contain private program and library resources so that each application version can run different versions of programs, even when the program resources have the same name. For more information, see How it works: multi-versioning applications.

You can get the following benefits from applications:

- Measure, at the application level, the cost of existing applications.
- Protect, by using policies, your CICS regions against problem applications that use too many resources.
- Manage application resources as a single entity throughout the application lifecycle and across every CICS region in which your application runs.
- Deploy multiple versions of the same application to the same regions, and deploy different applications to the same CICS regions even if they have clashing program names.
- Move applications from development, into test, and into production without changing them.

This section introduces "Components of an application" on page 46 and "How do I set up an application?" on page 48

Components of an application

A deployed CICS application is composed of:

- The CICS application itself
- One or more CICS bundles to define or import resources for use by the application
- A CICS application binding
- An APPLDEF resource definition
- One or more application entry points, which also set up an application context
- Optionally, CICS policy that defines task rules to ensure that the application runs within defined limits.

The deployed CICS application artefacts are located in a directory in zFS that is associated with the platform on which the application runs. See "Platform directory structure on zFS" on page 43 for details.

You use CICS Explorer to create and manage the components of the application. Figure 15 on page 49 shows the relationship between these components. You can secure a platform and its deployed applications by setting up RACF security profiles in a similar way to security for other CICSPlex SM components. For information, see Security for platforms and applications.

Application
In CICS Explorer, a CICS Application project defines an application bundle. An application bundle is a type of management bundle that describes a CICS application, including its name and version information. This management bundle references the CICS bundles that contain the dependencies

and resources for the application. All the application resources are installed and managed together. When the application is ready for testing, it is exported to zFS, where it is used by CICS TS at run time.

CICS bundles

The CICS TS application contains one or more CICS bundles, each of which define or import a set of resources for use by the application. A CICS bundle is a container for a set of CICS TS resources. When a CICS bundle is installed into a CICS TS region, the set of resources that it contains also get installed.

Imported resources in the bundle enable the application to declare a dependency on a specific resource that is not defined by this application, for example a file that is shared by multiple applications. The dependency is checked by CICS TS at installation time and, if it is not available, the application will not enable.

A CICS TS application might span several different types of CICS TS regions. For example, the application might process requests in a web-owning region (WOR) and then run business logic in an application-owning region (AOR). If this is the case, the CICS TS application can contain a CICS bundle of resources for the WOR, and a second CICS bundle of resources for the AOR. The CICS bundles must be available on zFS when the application is installed.

Application binding

The CICS application binding is the link between a CICS TS application and a CICS TS platform. A CICS TS application is deployed into a platform. The application should remain unchanged when it is deployed into different platforms (for example, a development platform and a test platform). To enable this, a CICS application binding is used. The application binding maps the bundles in the application (its needs) to the specific platform into which it is to be deployed. The application binding can add more CICS bundles to the application as an alternative to deploying them with the application or adding them to the platform. These bundles might include policies or resource definitions that control or customize the behavior of the application for the target platform. The application binding is exported to zFS. For more information, see Application binding

Application (APPLDEF) resource definition

The application definition, which is an APPLDEF resource definition in the data repository for the CICSplex, tells CICS TS where to find the application and application binding on zFS. On installation of the APPLDEF, the application is installed.

Application entry points

An application entry point identifies a resource that is an access point to an application. Application entry points are used to control users' access to different versions of an application that is deployed on a platform. PROGRAM, URIMAP, and TRANSACTION resources can be declared as application entry points.

You can declare one or more application entry points for a CICS bundle that is to be packaged and deployed as part of an application. Each application entry point is declared on a resource and also names the operation that is performed by the resource, such as a create, read, update, or delete operation. For more information, see Application entry points.

They are also used to create an application context to monitor the resource usage for applications and to identify an application that is running. When a task passes through an application entry point, application context data is associated with the task. The application context data is propagated forwards to other CICS TS regions that the task uses. The application context data can be made available in monitoring records to provide a way of measuring how much system resources an application is using. For more information, see Application context.

Optional policies

Policies that define task rules can be defined in CICS bundles that are part of the CICS application. These policies enforce that the application meets particular requirements. For example, it might need to complete within a specified amount of time.

For more information, see CICS policies.

How do I set up an application?

At a high-level, you'll go through the steps below. Figure 15 on page 49 shows what you'll end up with.

Applications are deployed into platforms.

1. Design the application, considering what application entry points, policies, and resources that you want to deploy as part of the application. For more information, see Designing a CICS platform.
2. In CICS Explorer, create CICS bundles to contain the application resources, application entry points dependencies, and any CICS policies relating to the application. For more information, see Defining CICS bundles.
3. In CICS Explorer, create an application project to define the application bundle. The application bundle references the required CICS bundles. For more information, see Packaging CICS applications for deployment in a cloud environment.
4. In CICS Explorer, create an application binding project to describe the deployment rules for the application to a region type in a platform.
5. From CICS Explorer, export the application project, the application binding project, and the CICS bundles to zFS. The export process exports the files to the platform home directory in zFS in the applications, bindings, and bundles subdirectories. For more information, see Deploying an application to a platform.
6. In CICS Explorer, create an application definition. The application definition is a CICSPlex SM APPLDEF resource definition, which points to the application bundle and the application binding in the platform home directory for the platform where the application runs. For more information, see Deploying an application to a platform.
7. In CICS Explorer, install the application to a platform. CICSPlex SM uses the information in the application bundle and the application binding to install the CICS bundles that make up the application into all of the target CICS regions in the platform. In each region, the resources that are specified in the bundles are dynamically created and CICS checks that any resources that are specified as dependencies are present. For more information, see Deploying an application to a platform.
8. In CICS Explorer, make the application available for users to start through the available application entry points. For more information, see Managing applications.

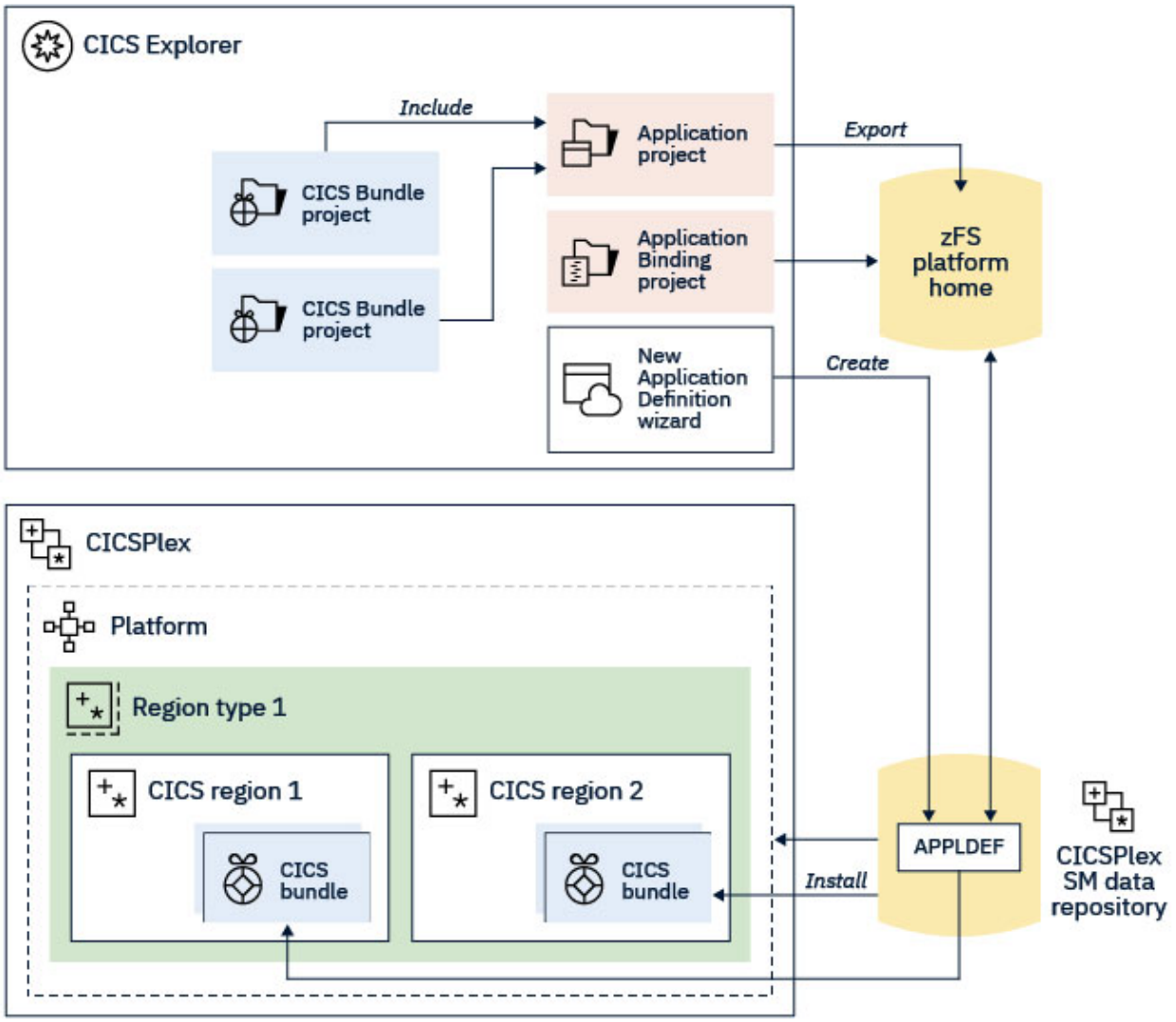


Figure 15. Creating elements of an application

Application binding

Before an application can be deployed, the application must be bound, or mapped, to the target platform. Each CICS bundle in the application must be bound to one or more CICS region types in the platform. The application binding specifies how applications are bound to platforms and ensures that CICS bundles are deployed correctly on the target platform.

A simple example CICS topology might require three CICS bundles to be installed into a single CICS region. A more sophisticated example topology might require one CICS bundle to be deployed to a region type in the platform, and the other two CICS bundles to both be deployed to a different region type in the platform. The application binding, rather than the application bundle, controls the deployment of the CICS bundles, so you do not include deployment information in the application itself. An application binding that maps CICS bundles to region types is required to install an application bundle.

Application bindings also support more complex scenarios in which different CICS resources are deployed on different platforms and region types. For example, URIMAP resources containing different URIs can be deployed on different platforms and region types. This type of customization can be achieved by including additional CICS bundles in the application binding. Each bundle, in this case containing a different URIMAP resource, is specified in the application binding, but the application itself does not require modifying. This type of application binding is optional.

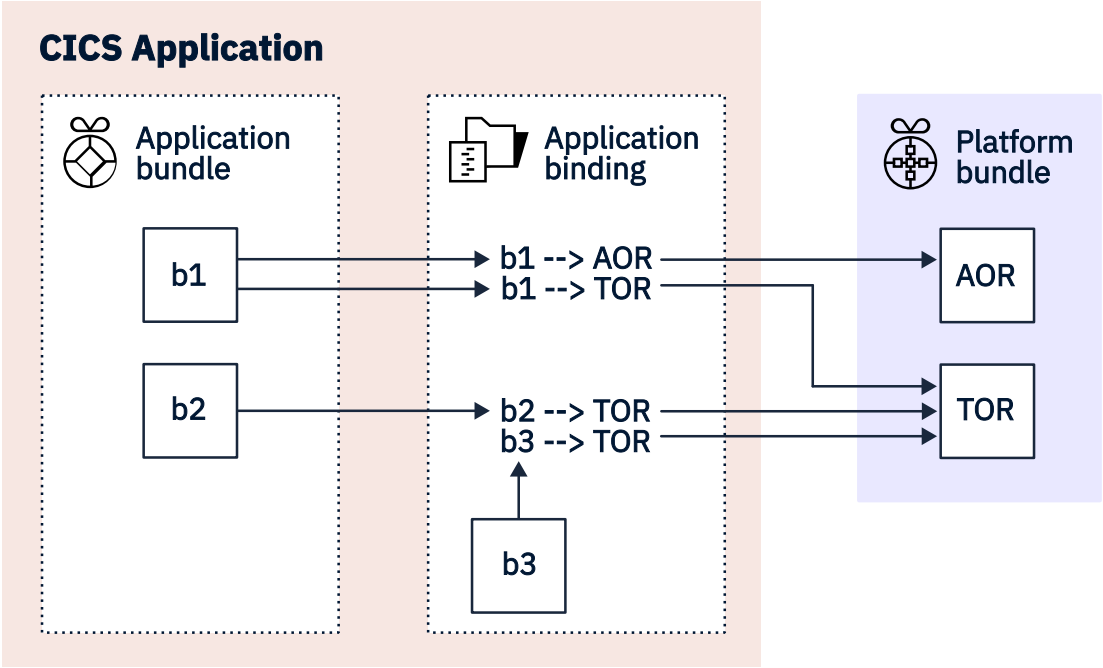


Figure 16. Example of binding an application to a platform

You create an application binding in CICS Explorer.The CICS Application Binding Project generates an application binding bundle, which is a type of management bundle, and updates the application definition with details of the bind directory. XML files are created:

- `bundles.xml` lists any CICS bundles that are to be deployed with this application binding.
- `appbinding.xml` associates the application binding with a specific application and a specific platform.
- `deployment.xml` lists the CICS bundles from both the application and the application binding, and associates them with region types from the platform.

After you have created an application binding, you can use the application binding descriptor editor in CICS Explorer to change how the CICS bundles from both the application and the application binding are deployed to region types in a platform. You can also add or remove CICS bundles that are deployed with the application binding.

When you make changes to the CICS bundles for an application, you use CICS Explorer to change the application binding to specify the new version of the application, and also to update the version number of the application binding to match the application version. You cannot use an existing version of an application binding with a new version of an application bundle. You must update the versions of the application bundle and application binding whenever you update the CICS bundles for the application.

For instructions on using CICS Explorer to create, deploy, and manage an application binding, see Creating a CICS Application Binding project in the CICS Explorer product documentation.

Application entry points

An application entry point identifies a resource that is an access point to an application. Application entry points are used to control users' access to different versions of an application that is deployed on a platform. They are also used to create an application context to monitor the resource usage for applications and to identify an application being run. PROGRAM, URIMAP, and TRANSACTION resources can be declared as application entry points.

For applications that are deployed on a platform, application entry points control users' access to the different versions of the application. Application entry points can be set as available or unavailable to users. You can install the application and its resources in the CICS regions in the platform at any

convenient time, then enable the CICS bundles to verify the installation. When you choose to provide the application version to users, you make the application entry points, and therefore the resources that they control for the application, available to callers.

You can declare one or more application entry points for a CICS bundle that is to be packaged and deployed as part of an application. Each application entry point is declared on a resource and also names the operation performed by the resource, such as a create, read, update, or delete operation.

- A resource for an application can only be declared once as an application entry point, naming one operation. You cannot declare multiple application entry points on the same resource.
- An operation name must be unique within an application.
- Operation names are case sensitive, so you may use operation names that are differentiated only by case.

The resource for an application entry point does not have to be defined in the same CICS bundle as the application entry point. CICS adds the application operation to the specified resource when the application is made available. When a resource for an application entry point and the entry point are both defined as part of an application, the entry point controls access to the service provided by the resource. For example, a TRANSACTION defined as an application entry point cannot be invoked until the application is made available.

You can declare an application entry point for a resource that is defined outside of the application. The resource is not part of the application, so the application entry point will not alter the availability of the service provided by a resource. You can also declare an application entry point for a PROGRAM resource that can be autoinstalled in the CICS regions where the application will be deployed. When you install an application, if the resource targeted by an application entry point is not present and cannot be autoinstalled, the CICS bundle containing the declaration of the application entry point will not be enabled.

CICS bundles that are installed as part of platform bundles, or added to a running platform, must not contain declarations of application entry points. Application entry points are not supported for CICS bundles installed directly on platforms, and CICS does not enable the application entry points in this situation, although the CICS bundle and its resources are installed. Stand-alone CICS bundles that are installed directly in CICS regions can contain declarations of application entry points to enable scoping of region level policies.

User access can be controlled on resources defined as application entry points. If an application includes any public resources that are not named as application entry points, when the application is installed and enabled, these resources can be accessed by other applications installed on the platform or in the CICS region regardless of the availability status of the application. Private resources for an application version cannot be accessed by other applications.

Application entry points are also used to create an application context for tasks. When a task calls a resource that has an application entry point, CICS creates an application context and associates it with the task. The application context is used to identify, load, and browse private resources for applications deployed on platforms, to apply policies to tasks, and to monitor the resources used by applications across multiple CICS regions and tasks. For information on the application context, see “Application context” on page 53.

Use CICS Explorer to declare application entry points for your applications, and to make available or unavailable, enable or disable, and install or discard applications. For instructions to set up application entry points, see Defining application entry points in the CICS Explorer product documentation.

PROGRAM resources as application entry points

Programs that are declared as an application entry point must have a unique PROGRAM resource name in your environment. To enable these programs to be called from outside the application, they must be public resources. When you make an application available that contains an application entry point for a private PROGRAM resource, the PROGRAM resource that is named as the application entry point changes from a private resource to a public resource. Only one instance of a public resource with a particular name can exist in a CICS region. The private PROGRAM resource therefore cannot have the same name as a

public program that is installed in the CICS region, or the same name as a public program that is defined as an application entry point by a different installed application. However, multiple versions of the same private PROGRAM resource defined as an application entry point can be installed for multiple versions of the same application, because CICS manages the promotion of private PROGRAM resources to public status for the versions of an application.

When you enable an application that contains an application entry point for a PROGRAM resource, CICS carries out the following checks to ensure that the resource is valid for use by the application:

- If the PROGRAM resource that you name as an application entry point is a private PROGRAM resource that is defined in one of the CICS bundles packaged with the application, CICS checks that the PROGRAM resource does not have the same name as a public program that is installed in the CICS region, or as a public program that is defined as an application entry point by another installed and enabled application.
- If the PROGRAM resource that you name as an application entry point is not defined in one of the CICS bundles packaged with the application, but is not yet installed as a public program, CICS attempts to autoinstall the program, then to reserve it for the application and enable an application entry point for it. If this action takes place, the autoinstalled program becomes a private program when you make the application version unavailable, so the same action can be taken for future versions of the application.
- If the PROGRAM resource that you name as an application entry point is already installed as a public program, CICS checks that the PROGRAM resource has not already been defined as an application entry point by another installed and enabled application, then reserves the public program for the application, and enables an application entry point for it. If this action takes place, CICS cannot automatically manage the application entry points for future versions of the application, because a public program that was installed before the application cannot become a private program. To update the application to a new version, you will need to disable and discard the existing version. To avoid this situation, you can arrange that the public program is autoinstalled by the application installation process, in which case it can become a private program and allow future application versions to be installed at the same time. Alternatively, you can define the program in one of the CICS bundles deployed with the application version, ensuring that it has a unique name, and so make it a private program.

When the enabling process has completed successfully, the application entry point is in an ENABLED state, but the program is not yet available to callers through its application entry points. When you make the application available using CICS Explorer, CICS allows callers to access the application through its application entry points, and makes the private PROGRAM resources for application entry points into public resources. Callers can either use the **EXEC CICS LINK** command to access the highest available application version, or use the **EXEC CICS INVOKE APPLICATION** command to specify any available application version. Where multiple versions of the same application are available, the PROGRAM resource for the highest application version is public, and the others are private.

When you make an application version unavailable, the PROGRAM resources for the application entry points remain reserved for the application, but they are no longer available to callers. Any PROGRAM resources that began as private resources defined as part of the application, or were autoinstalled during the enabling process, are changed back from public to private programs. When you disable an application version, the PROGRAM resources for the application entry points are no longer reserved for the application. Any PROGRAM resources that began as public resources are made available to other applications and to callers.

Programs that are declared as application entry points are identified and reported in both the public and private resource statistics for program definitions and JVM programs produced by CICS, because, while the entry point is publicly accessible, it is also part of the application. For the program statistics that are produced by the loader domain, application entry points are not identified, and only one private program statistics record is written.

You can also define application entry points in stand-alone bundles that are installed directly in CICS regions, to enable scoping of region level policies to CICS tasks for a particular initial PROGRAM. However, in this situation, the service provided by the PROGRAM becomes available as soon as you install and enable the PROGRAM resource and is not controlled by the availability of the bundle that defines the application entry point.

TRANSACTION resources as application entry points

Transactions that are declared as an application entry point must have a unique TRANSACTION resource name in your environment. A TRANSACTION resource can be defined in a CICS bundle and packaged as part of an application, but is not supported as a private resource for the application, so cannot be installed in multiple application versions. If you intend to deploy multiple version of the application concurrently, when you update applications that include TRANSACTION resources defined in CICS bundles, you must rename the TRANSACTION resources for each version of the application.

When you define a TRANSACTION resource in a CICS application bundle, you can use an application entry point declaration to control users' access to the service provided by the TRANSACTION resource. When you install and enable the application, the service provided by the TRANSACTION resource is not yet available to callers. To make the application entry point and service provided by the TRANSACTION resource available to callers, use CICS Explorer to make the application available and therefore make the application entry point and TRANSACTION resource available.

You can also declare a TRANSACTION resource as an application entry point if it is defined outside the application. In this situation, the service becomes available to users as soon as you install and enable the TRANSACTION resource.

You can also define application entry points in stand-alone bundles (if they are installed directly in CICS regions) to enable scoping of region level policies to CICS tasks for a particular TRANSACTION. If you want the application entry point to also control access to the service provided by the TRANSACTION resource, declare the application entry point and the TRANSACTION in the same CICS bundle. In this situation, the service provided by the TRANSACTION is not available to callers until the bundle is made available. If the application entry point and the TRANSACTION definition are not in the same stand-alone CICS bundle, then the service provided by the TRANSACTION becomes callable as soon as you enable the TRANSACTION.

CICS system transactions must not be defined as application entry points.

URIMAP resources as application entry points

URIMAP resources that are declared as an application entry point must have a unique name in your environment. A URIMAP resource can be defined in a CICS bundle and packaged as part of an application, but it is not supported as a private resource for the application, so it cannot be installed in multiple application versions. If you intend to deploy multiple version of the application concurrently, when you update applications that include URIMAP resources defined in CICS bundles, you must rename the URIMAP resources for each version of the application.

You can also define application entry points in stand-alone bundles (if they are installed directly in CICS regions) to enable scoping of region level policies to CICS tasks for a specific URIMAP resource. If you want the application entry point to also control access to the service provided by the URIMAP resource, declare the application entry point and the URIMAP in the same CICS bundle. In this situation, the service provided by the URIMAP is not available to callers until the bundle is made available. If the application entry point and the URIMAP are not defined in the same bundle, the service provided by the URIMAP becomes available as soon as you install and enable the URIMAP resource.

Application context

An application context is a set of data that identifies tasks that are running in the context of your application and platform. When an application is accessed through its application entry points, the CICS regions generate application context data for the tasks related to the application. This task application context is available in the performance records that the CICS monitoring facility writes to SMF and includes data fields about the platform, the application, the application version and the operation. You can use the information to measure resource consumption by applications (or by particular routes into applications), to use policy-based management for applications, and to relate tasks to specific applications to help with problem diagnosis.

A task can pass through one or more applications as it executes. Each task can have up to two application contexts associated with it at any time:

The *initial* application context of a task
Used for monitoring and measuring how much resource an application or a particular application operation is using across CICS regions and multiple tasks. The initial application context can be used when applying a policy to tasks that are part of an application, to define threshold conditions to manage the behavior of the tasks. The initial application context can be inherited from an invoking task, or set when the task first passes through an application entry point.

The *current* application context of a task
Used for loading private libraries and WLM user exits. The current application context can be queried using XPI, SPI, and API calls. The current application context changes each time the task passes through an application entry point.

Both the initial or current application context can be used with the transaction tracking capability in CICS Explorer to quickly identify and diagnose application-related problems. Both the initial and the current application contexts are propagated from task to task.

Creating an application context

To create an application context, a CICS resource must be declared as an application entry point for an application that is deployed on a platform. For more information about application entry points, see “Application entry points” on page 50.

Note: Using the **EXEC CICS XCTL** command or COBOL call (whether static or dynamic) to create an application context is not supported.

When a task that does not have an application context calls a resource that is declared as an application entry point, CICS creates an application context that becomes the initial application context associated with the task. This application context is also associated with any subsequent programs that the task calls and tasks that it starts. If a task already has an application context, the new application context becomes the current application context for the task. The task's initial application context continues to be used for monitoring and scoping of policies.

The application context contains the following data:

- Application name
- Major version number for the application
- Minor version number for the application
- Micro version number for the application
- Platform name for the platform where the application is deployed
- Operation name for the application entry point

You can specify the application context to browse private resources by using the **EXEC CICS INQUIRE** system programming command. By default, CICS searches for the private and public resources that are available to the program where the **EXEC CICS INQUIRE** command is issued. You can specify a different application context to browse the private and public resources that are available for another application. When you use the application context for browsing the resources that are used by an application, you do not specify the operation name.

Application context data is passed in the DFHDYPDS parameter list and the EYURWCOM parameter list, and can be used in custom dynamic routing algorithms. For more information about dynamic routing, see Dynamic routing with CICSPlex SM.

Viewing the current application context

To view the current application context, you can:

- Use the CICS Explorer Task Association view.
- Use the **EXEC CICS INQUIRE ASSOCIATION** command.
- Use the **EXEC CICS ASSIGN** command.

- Query the current application context from within a global user exit using the monitoring XPI function INQUIRE_APP_CONTEXT.
- Query the current application context using the JCICS `Task.getApplicationContext()` method.

When application context isn't propagated

By default, the initial and current application contexts are propagated from task to task. However, some CICS commands, interfaces, connection types, and other processes do not support application context propagation between tasks.

The following scenarios do not support application context propagation:

- A task is attached by a START command that specifies the TERMID option.
- A task is attached by a DTP or CPIC request.
- A task is attached over an APPC connection.
- A task is attached using the transaction start EP adapter.
- A task is attached by the JVM server when a `ThreadExecutor` service creates a new thread.
- A web service pipeline handler transaction is routed over an MRO connection.
- An outbound HTTP request is made to CICS using CICS Web support.
- A task is created using a URIMAP resource as an application entry point for Liberty that calls `CICSExecutorService` to start a thread in a Java program.

Conditions for setting application context on to a task

PROGRAM resource

When a PROGRAM resource is defined as an application entry point, the application context is set onto a task when any of the following conditions occur:

- A CICS transaction that specifies the name of the PROGRAM resource runs (the context is set before the nominated resource runs).
- A program issues an **`EXEC CICS LINK`** command to the PROGRAM resource.
- A program issues an **`EXEC CICS INVOKE APPLICATION`** command naming an operation which equates to the PROGRAM resource.
- A program issues an **`EXEC CICS RUN TRANSID`** command to the PROGRAM resource.
- A Java program issues a `Program.link()` command to the PROGRAM resource.
- A Java program issues an `Application.invoke()` command naming an operation which equates to the PROGRAM resource.

Note:

- The application context is set on a task for a program defined as an application entry point, only if that program runs locally. If the program runs remotely, the request is passed to the remote CICS region, and it is on the remote CICS region that the context is set, if the resource is also flagged as an application entry point on that CICS region.
- The application context on a task is unchanged for either a COBOL program that makes a dynamic call to another program or a PL/I program that issues a FETCH and CALL to call another program (that is, if the **`EXEC CICS LINK`** call is bypassed).

TRANSACTION resource

When a TRANSACTION resource is defined as an application entry point, the application context is set onto a task when any of the following conditions occur:

- The TRANSACTION resource runs (the context is set before the nominated resource runs).
- A program issues an **`EXEC CICS START`** command for the TRANSACTION resource.

URIMAP resource

When a URIMAP resource is defined as an application entry point, the application context is set onto tasks as follows:

- If the HTTP/HTTPS request URL matches a URIMAP resource with USAGE(JVMSERVER), the application context is set on the task that runs the CICS transaction specified in the URIMAP resource.
- If the HTTP/HTTPS request URL matches a URIMAP resource with USAGE(SERVER) and a static response is to be provided, the application context is set on the CWXN task. A static response is provided by specifying the HFSFILE or TEMPLATENAME values.
- If the HTTP/HTTPS request URL matches a URIMAP resource with USAGE(SERVER) and a static response is not to be provided, the application context is set on the CWXN task and this application context is also propagated to the task that runs the CICS transaction that is specified in the URIMAP resource. For the requests that are eligible to directly attach the transaction, the application context is set on the CICS transaction that is specified on the URIMAP resource. For more information, see Processing HTTP requests by using directly attached user transactions. The default alias transaction is the CWBA transaction.
- If the HTTP/HTTPS request URL matches a URIMAP resource with USAGE(PIPELINE), the application context is set on the CWXN task and also propagated to the task that runs the CICS transaction that is specified in the URIMAP resource. For the requests that are eligible to directly attach the transaction, the application context is set on the CICS transaction that is specified on the URIMAP resource. For more information, see Processing HTTP requests by using directly attached user transactions. The default transaction is the CPIH transaction.

Table 7. Application context data (ACD) settings of a task when a transaction and its initial program are both defined as entry points (EP).

Transaction is defined as an application entry point	Initial program is defined as an application entry point	Initial program is public or private	Initial ACD for task	Current ACD for task
Yes	Yes	Not applicable	Set by the transaction entry point	Set by the program entry point
Yes	No	Public	Set by the transaction entry point	Empty
Yes	No	Private	Set by the transaction entry point	Same as initial ACD
No	Yes	Not applicable	Set by the program entry point	Same as initial ACD
No	No	Not applicable	Not set	Not set

Table 8. Application context data (ACD) settings of a task when a URIMAP and its alias transaction are both defined as entry points (EP).

URIMAP is defined as an application entry point	Alias transaction is defined as an application entry point	Initial program is public or private	Initial ACD for task	Current ACD for task
Yes	Yes	Public	Set by the transaction entry point	Empty
Yes	Yes	Private	Set by the transaction entry point	Same as initial ACD

Table 8. Application context data (ACD) settings of a task when a URIMAP and its alias transaction are both defined as entry points (EP). (continued)

URIMAP is defined as an application entry point	Alias transaction is defined as an application entry point	Initial program is public or private	Initial ACD for task	Current ACD for task
Yes	No	Public	Set by the URIMAP entry point	Empty
Yes	No	Private	Set by the URIMAP entry point	Same as initial ACD
No	Yes	Public	Set by the TRANSACTION entry point	Empty
No	Yes	Private	Set by the TRANSACTION entry point	Same as initial ACD
No	No	Not applicable	Not set	Not set

Note: The initial program of an alias transaction for a URIMAP cannot be set as an application entry point because its initial program starts with DFH, which cannot be defined as an entry point. Therefore, all programs in Table 8 on page 56 are not defined as entry points.

Table 9. Application context data (ACD) settings of a task when transactions are attached by EXEC CICS START or EXEC CICS RUN TRANSID.

Started transaction is defined as an application entry point	Initial program of started transaction is defined as an application entry point	Initial program is public or private	Starting task has ACD set	Initial ACD for task	Current ACD for task
Yes	Yes	Not applicable	Not applicable	Set by the TRANSACTION entry point	Set by the PROGRAM entry point
Yes	No	Public	Not applicable	Set by the TRANSACTION entry point	Empty
Yes	No	Private	Not applicable	Set by the TRANSACTION entry point	Same as initial ACD
No	Yes	Not applicable	Not applicable	Set by the PROGRAM entry point	Same as initial ACD
No	No	Not applicable	Yes	Inherits ACD from starting task	Empty
No	No	Not applicable	Not applicable	Not set	Not set

Table 10. Application context data (ACD) settings of a task when a transaction is defined as DYNAMIC.		
Transaction is defined as an APPLICATION entry point	**Starting task has ACD set**	**ACD passed into routing program COMMAREA**
Yes	Yes	Transaction's entry point
Yes	No	Transaction's entry point
No	Yes	Starting task's current ACD
No	No	No ACD

Application states

CICS Explorer shows the state information for an application. The status information for an application version shows whether the CICS bundles for the application are present, enabled, and available in the CICS regions associated with the region type in the platform. The status information for a version of an application is derived from the status of the individual management parts for that version of the application. A management part is a MGMTPART record that is created automatically during the application install process. Management parts record the relationship between the application and each installed CICS bundle, and the region type where each CICS bundle is installed in the platform.

Follow the CICS Explorer instructions in Checking the status of an application in the CICS Explorer product documentation. The following diagrams show the possible values for the status of the application and the associated management parts.

Figure 17 on page 58 shows the actions that you take during the provisioning stage of an application's lifecycle, and the appropriate temporary status, enablement status, and availability status in each situation, together with the possible error states. For information about handling error states, see Diagnosing platform errors

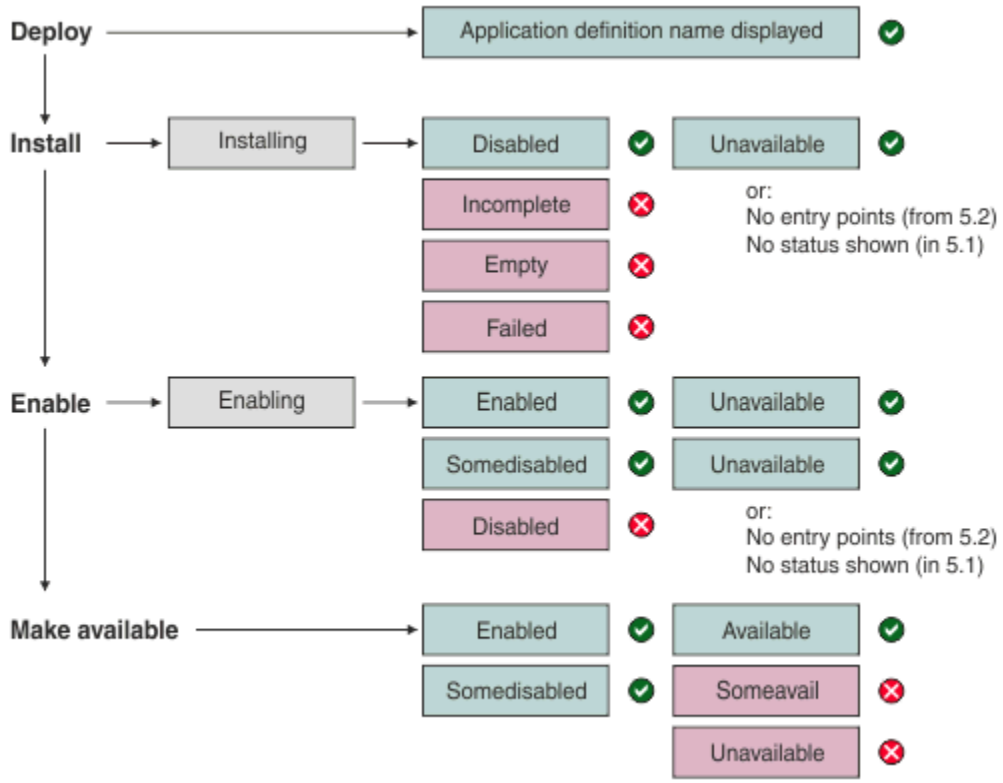


Figure 17. Application lifecycle: Provisioning

- You deploy a CICS Application project by exporting it to the home directory for the platform on zFS, and creating an APPLDEF resource definition for the application version in the data repository of the CMAS. When you have completed this process, the Cloud Explorer view displays the name of the application definition for the application version.
- When you install an application, the status INSTALLING is displayed in the Cloud Explorer view while installation is in progress. When installation is complete, the expected status for the application is DISABLED, UNAVAILABLE. If the application has no application entry points, the Cloud Explorer view displays this as the availability status. If the application is installed in a CICS TS 5.1 region, no availability status is displayed.
- When you enable an application, the status ENABLING is displayed in the Cloud Explorer view while enabling is in progress. When enabling is complete, the expected status for the application is ENABLED, UNAVAILABLE.
- An application in the SOMEDISABLED state can be made available if necessary, however, application entry points of bundles not already enabled will not become available during this action.
- You can make an application available with an enabled status of SOMEDISABLED. There are two factors you must be aware of however. When the enabled status is SOMEDISABLED, the availability status of your application is not restored if you start or restart a region. Also, you cannot automate the deployment of an application to a target state of AVAILABLE using the DFHDPLOY utility. For more information, see Automate the deployment and undeployment of CICS applications with the DFHDPLOY utility
- When you make an application available, the status for the application is expected to change to ENABLED, AVAILABLE. If the status for the application shows that it has no application entry points or is installed in a CICS TS 5.1 region, the Make Available action is not required.

Figure 18 on page 59 shows the actions that you take during the deprovisioning stage of an application's lifecycle, and the appropriate temporary status, enablement status, and availability status in each situation, together with the possible error states.

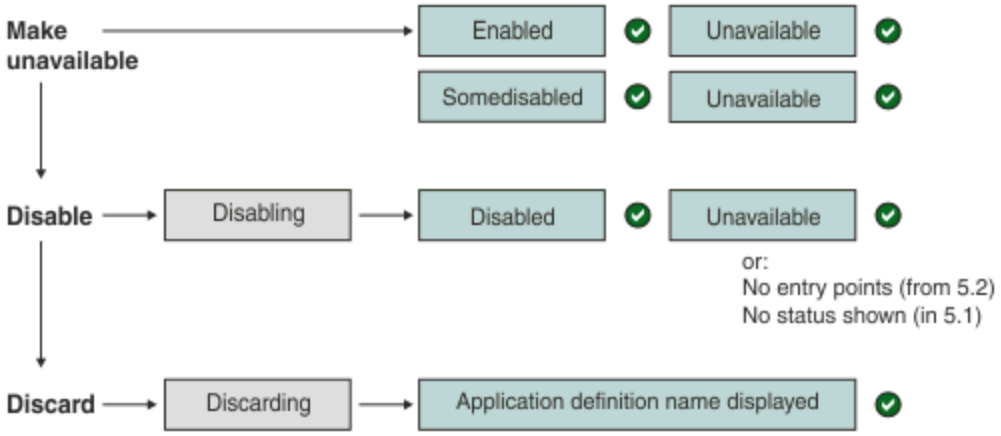


Figure 18. Application lifecycle: Deprovisioning

- When you make an application unavailable, the status for the application changes to ENABLED, UNAVAILABLE. If the status for the application shows that it has no application entry points, or that it is installed in a CICS TS 5.1 region, the Make Unavailable action is not required.
- When you disable an application, the status DISABLING is displayed in the Cloud Explorer view while disabling is in progress. When disabling is complete, the expected status for the application is DISABLED, UNAVAILABLE. Alternatively, the status can show that the application has no application entry points, or that it is installed in a CICS TS 5.1 region.
- When you discard an application, the status DISCARDING is displayed in the Cloud Explorer view while the discard is in progress. When discarding is complete, the application name is no longer shown in the Cloud Explorer view. Instead, the name of the application definition for the application version is displayed.

Table 11. Application states	
Status value	**Meaning**
AVAILABLE	The application version has been made available to callers through its application entry points.
DISABLED	All the management parts for the application version are disabled.
DISABLING	All the management parts for the application version are in the process of being disabled.
DISCARDING	The application version is being discarded, and it cannot be enabled or disabled at this time.
EMPTY	None of the management parts for the application version are installed.
ENABLED	All the management parts for the application version are enabled.
ENABLING	All the management parts for the application version are in the process of being enabled.
FAILED	A problem occurred during install or discard of the application version.
INCOMPLETE	Some of the management parts for the application version are empty or have an invalid scope.
INSTALLING	The application version is being installed, and it cannot be enabled or disabled at this time.
NONE (No entry points)	The application version does not have any application entry points.
SOMEAVAIL	The **Make Available** or **Make Unavailable** action has been performed for the application version, but some application entry points are available and some are unavailable.
SOMEDISABLED	Some of the management parts for the application version are disabled.
UNAVAILABLE	The application version is set as unavailable to callers.

Table 12. Management part status values	
Status value	**Meaning**
AVAILABLE	The application entry points declared in the CICS bundle have been made available to callers.
DISABLED	The CICS bundle is disabled in all of the CICS regions.
DISABLING	The CICS bundle is in the process of being disabled.
EMPTY	The CICS bundle is not installed in any of the CICS regions.
ENABLED	The CICS bundle is installed and enabled in all of the CICS regions.
ENABLING	The CICS bundle is in the process of being enabled.
IMPORTONLY	The CICS bundle is installed and enabled in all of the CICS regions, but it only contains import statements, so does not affect the status of the application.
INCOMPLETE	The CICS bundle is installed in some, but not all, of the CICS regions.
INVALIDSCOPE	The CICS system group specified for installing the CICS bundles does not exist, so no CICS bundles are installed.

Table 12. Management part status values (continued)	
Status value	**Meaning**
NONE (No entry points)	The CICS bundle does not contain any statements of application entry points.
SOMEAVAIL	The **Make Available** or **Make Unavailable** action has been performed for the CICS bundle, but some application entry points are available and some are unavailable.
SOMEDISABLED	The CICS bundle is disabled in some of the CICS regions. This status can also occur when the CICS bundle is disabled in all of the CICS regions, if the ENABLEDCOUNT value for any of the installed BUNDLE resources is greater than 0, indicating that one or more resources, application entry points, or policy scopes that were created by the CICS bundle are currently enabled in the CICS region.
UNAVAILABLE	The application entry points declared in the CICS bundle are set as unavailable to callers.

How it works: multi-versioning applications

You can install and manage multiple versions of an application at the same time on the same platform instance. With multi-versioning, new versions of an application can be deployed to the platform without the need to disable or remove the previous version and made available to users without service interruption. Multi-versioning enables different users to start different versions of the application, and it makes it simple to switch back to a stable version of an application if a new version turns out to have errors. At runtime, programs can invoke any available version of the application or, for programs that are not aware of multi-versioning, they automatically link to the highest version of the application that you made available on the platform.

You can host two versions of the same application concurrently for use by different groups of user, or you can replace one version of an application with another version for use by all users. For an overview, see “Example: hosting two versions of an application concurrently” on page 67 and “Example: replacing one version of an application with another” on page 63. Resources can be made private to each version of the application.

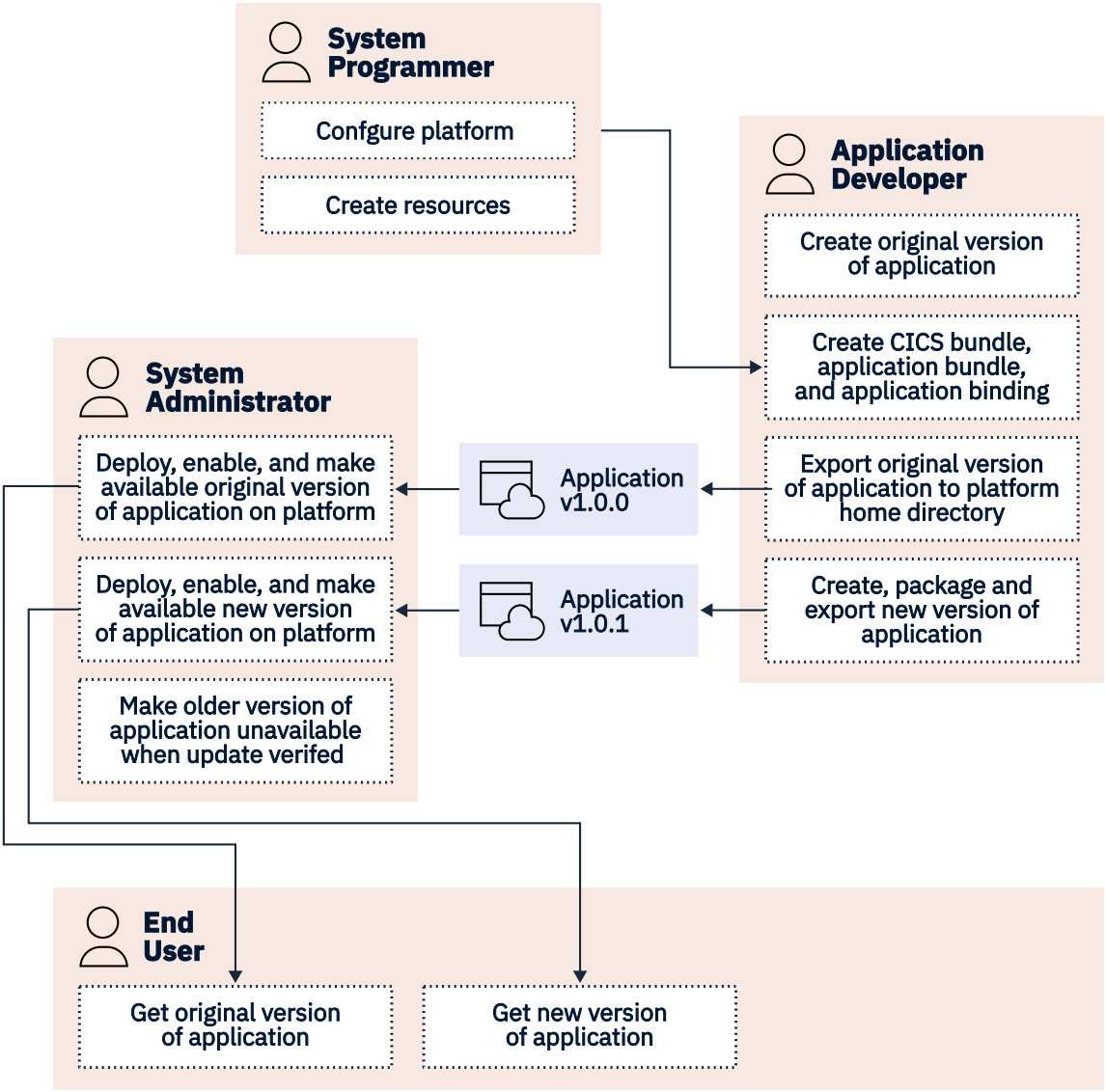


Figure 19. Using multiple versions of an application in a cloud solution

Resources can be treated as private to each version of the application

In applications, you package key resources into the application for lifecycle management. These key resources include the programs that are identified as application entry points and application version-specific load libraries. By bringing CICS TS resource definitions into a CICS TS application, you provide better management for the application throughout its lifecycle. During application development, it is beneficial to describe the resources in a single development environment, and editable by the application developers. When the application is in source code management, it can encompass and track all related artifacts through application deployment. The application can be provisioned and deprovisioned through a single control point, while the states of all of the related CICS TS resources are managed by the system.

Traditionally, applications rely on key CICS TS resources to be provided outside the application, for example, by the CICS system definition data set (CSD). But application entry points for different versions of an application cannot share resource names if the resource is provided by the CSD or CICSPlex SM Business Application Services (BAS). Multi-versioning enables multiple versions of an application to declare certain CICS TS resources with the same name. Each resource is treated as unique and private to the version of the application.

For supported resource types, a CICS resource is private if the resource is defined in a CICS bundle that is packaged and installed as part of an application, either as part of the application bundle, or as part of the application binding. When you create a CICS resource in this way, the resource is not available to any other application or version that is installed on the platform, and it is not available to other applications in the CICS region. It can be used only by the version of the application where the resource is defined. These resources are known as *private resources*. For more information, see Private resources for application versions.

The following resources are supported as part of multi-versioned applications:

- PROGRAM resources that are defined in CICS bundles that are part of the application.
- LIBRARY resources that are defined in CICS bundles that are part of the application.
- PACKAGESET resources that are defined in CICS bundles that are part of the application.
- POLICY resources
- Statements of application entry points
- Any resource that is defined as a dependency, or bundle import, for the application.

Other resources can be involved with multi-versioned applications if you manage the resources to avoid resource name clashes between different versions of the application. For example, a URIMAP resource that is part of an application can be renamed when you package and install a new version of the application. Or an application can be designed so that a resource that is not supported for multi-versioning is managed outside the application, but declared as a dependency, or bundle import, for the application. For resources that are, or could be, used by different applications, such as JVMSERVER resources, deploy and manage the resource at the level of the platform, where it can be used by any version of any application that is deployed on the platform.

As you do with a single version of an application, you control users' access to the resources in a multi-versioned application by declaring application entry points.

You can invoke a specific version of a multi-versioned application by using the **EXEC CICS INVOKE APPLICATION** command. For more information about invoking a specific version of an application, see Invoking a multi-versioned application.

Example: replacing one version of an application with another

Imagine a scenario where a company has a CICS application that users access by using a CICS transaction. A system administrator needs to deploy a new version of the application to fix a bug in the application and expects all users to move at the same time to using the new version.

The system administrator, or the application developer, uses CICS Explorer to package and export the new version of the application. The system administrator deploys the new version of the application to the platform, then enables the installed application. When the application goes into enabled status, the system administrator knows that the installation was successful. At this point, the older version of the application is still provided to users when they enter the CICS transaction. When the system administrator makes the new version of the application available, CICS immediately provides the users with that version in place of the older version. The users enter the same CICS transaction as before, and there is no need for the users to take any action.

If there is a problem with the new version of the application, users can roll back quickly to the older version of the application. By default, CICS provides callers (such as CICS transactions or other linking applications) with the highest version of the application that you make available on the platform. If the system administrator makes, or keeps, the older version available, and then makes the new version unavailable, users are automatically provided with the older version again. User access to applications is controlled by the application entry points that are declared for the resources in your applications, which can set the resources to be available or unavailable. The system administrator does not need to disable or discard alternative versions of the application to prevent users from accessing them.

In this scenario, because the older version of the application is defective, it is appropriate to disable or discard it after the new version is verified. However, multiple versions of applications that are deployed on platforms can remain available concurrently to users. Programs that are coded to be aware of multi-

versioning can access specific major and minor versions of an application by using the **EXEC CICS INVOKE APPLICATION** command.

Figure 20 on page 65 shows that projects in CICS Explorer are used to define a platform and package the application. The example application includes a CICS bundle with definitions for a LIBRARY resource and two PROGRAM resources, called PROGA and PROGB. The TRANSACTION resource for the application is defined in a CICS bundle, but it is not deployed with the application. The Platform project, Application project, Application Binding project, and CICS Bundle projects are exported to subdirectories in the zFS platform home directory. A platform definition (PLATDEF) and application definition (APPLDEF) are created for the platform and application, and stored in the CICSPlex SM data repository. The platform definition is installed in the CICSplex to create a platform with a region type that contains the target CICS region. The application definition is installed on the platform to create the CICS bundle for version 1.0.0 of the application in the CICS region. With the information from the CICS bundle, the LIBRARY and PROGRAM resources for the application are dynamically generated in the CICS region. The LIBRARY resource points to the data set on z/OS with the programs for version 1.0.0 of the application. The CICS bundle that contains the definition for the TRANSACTION resource is also added to the platform, and the TRANSACTION resource is dynamically generated in the CICS region.

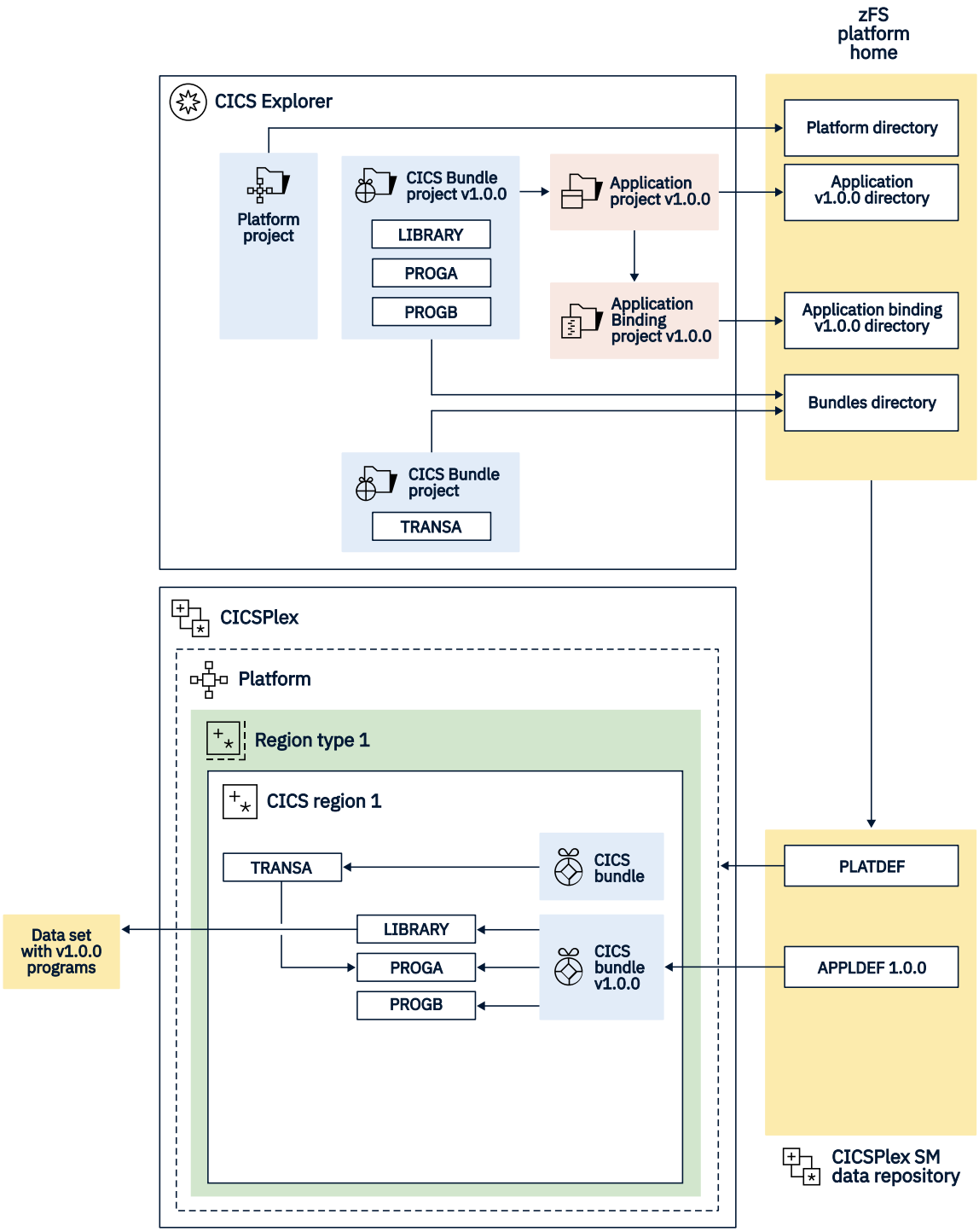


Figure 20. Example topology for Version 1.0.0 of the application

Figure 21 on page 67 shows that, for Version 1.0.1 of the application, the LIBRARY resource definition in the CICS bundle is updated to point to the data set with the new versions of the programs for the application. The other resource definitions for the application are not changed. The CICS bundle project that contains the resource for the CICS transaction also does not need to be updated because it is deployed separately.

CICS Explorer is used to package the new version of the application. The new application version includes version 1.0.1 of the CICS Bundle project that defines the resources for the application. New subdirectories are created for the new versions of the application and application binding, but the CICS bundle is stored in the same bundles subdirectory. An application definition (APPLDEF) is created for

version 1.0.1 of the application, and stored in the CICSPlex SM data repository. The name of the APPLDEF resource is the same name that was used for the older version's APPLDEF resource, but the version number is changed to match the new version of the application. The new application definition is installed on the platform to create the CICS bundle and resources for version 1.0.1 of the application in the CICS region. The CICS bundles and resources for version 1.0.0 of the application are still installed in the CICS region, but the TRANSACTION resource, which has not been updated, now points automatically to Version 1.0.1 of the application.

The CICS region now hosts the new version of the application. Private LIBRARY and PROGRAM resources for the new version of the application are dynamically created in the CICS regions when the CICS bundle that contains the resources for the new version of the application is installed. The new versions of the programs for the application are stored in a different Partitioned Data Set (PDS) or Partitioned Data Set Extended (PDSE) on z/OS. The updated LIBRARY resource for the new version of the application references the new data set. The CICS transaction that runs the application is not involved with the update process, and it is unchanged. When users run the transaction, CICS automatically provides users with the highest available version of the application, so the program for version 1.0.1 of the application now runs.

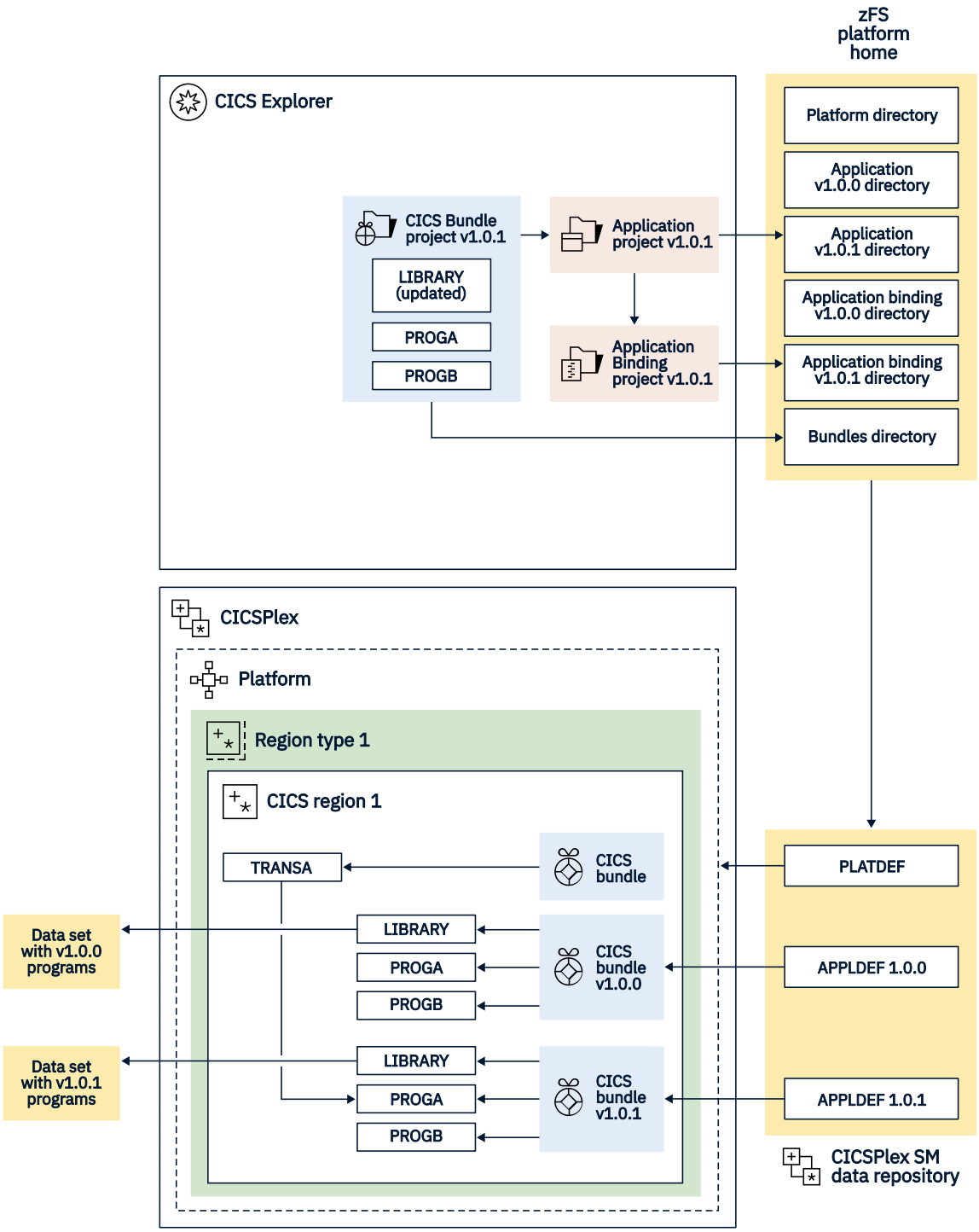


Figure 21. Example topology for a multi-versioned application, where one version of the application replaces a previous version for all users

Example: hosting two versions of an application concurrently

Imagine a scenario where a company has a CICS application that is presented to users through a browser interface as a web application, with the presentation logic hosted in a Liberty JVM server. The company plans to introduce a new version of the application. The change to the application is significant and, to avoid disruption to the business, the company doesn't want all of the users to move to the new version at the same time.

The system administrator deploys and makes available both versions of the application in the same CICS regions that are part of a platform. The users are directed to either the new version or the older version of the application by using different URLs.

Figure 22 on page 69 shows that, for Version 1.0.0 of the application, projects in CICS Explorer are used to define a platform and package the application. The application includes a CICS Bundle project that packages the web application as an OSGi Application Project contained in an enterprise bundle archive (EBA). The URIMAP resource for the application, called INDEX, is created in a CICS bundle and deployed with the application binding. A Liberty JVM server is also defined using a CICS Bundle project. The projects are exported to subdirectories in the zFS platform home directory. A platform definition (PLATDEF) and application definition (APPLDEF) are created for the platform and application, and stored in the CICSPlex SM data repository. The platform definition is installed in the CICSplex to create a platform with a region type that contains the target CICS region. The application definition is installed on the platform to create the CICS bundles for version 1.0.0 of the application in the CICS region. With the information from the CICS bundles, the URIMAP resource for version 1.0.0 of the application, named INDEX, is dynamically generated in the CICS region, and the resources for the web application are deployed in the Liberty JVM server.

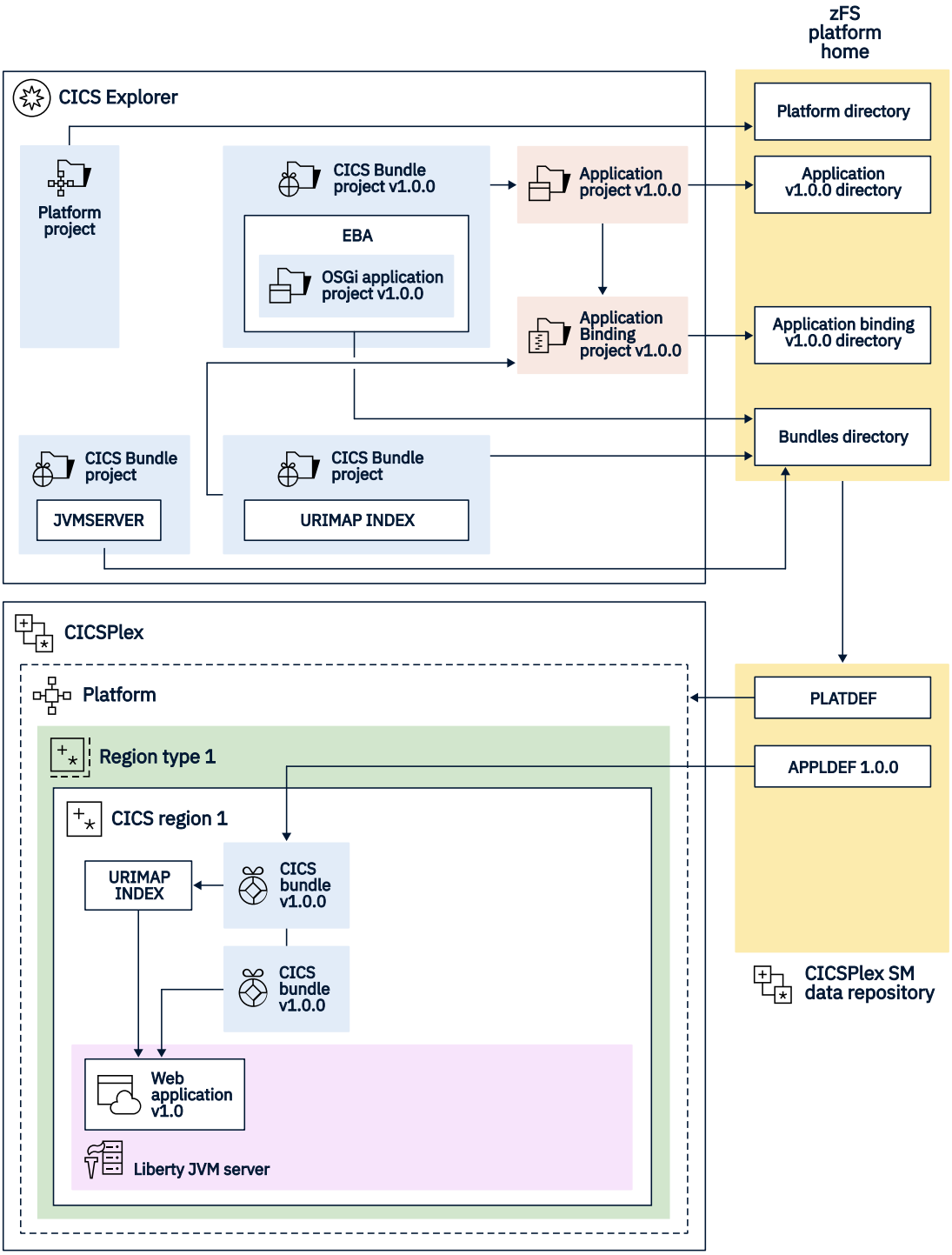


Figure 22. Example topology for Version 1.0.0 of the application

Figure 23 on page 71 shows that, for Version 1.1.0 of the application, CICS Explorer is used to package the application. The new application version includes version 1.1.0 of the CICS Bundle project that packages the web application, and the OSGi Application Project is also re-versioned to Version 1.1.0. A new URIMAP resource for the new version of the application is created in a CICS bundle and deployed with the new version of the application binding. The projects are exported to subdirectories in the zFS platform home directory. New subdirectories are created for the new versions of the application and application binding, but the CICS bundles are stored in the same `bundles` subdirectory. An application definition (APPLDEF) is created for version 1.1.0 of the application, and stored in the CICSPlex SM data repository. The name of the APPLDEF resource is the same name that was used for the older version's

APPLDEF resource, but the version number is changed to match the new version of the application. The new application definition is installed on the platform to create the CICS bundles and resources for version 1.1.0 of the application in the CICS region, alongside the CICS bundles and resources for version 1.0.0 of the application.

The CICS region handles the workload for the new version of the application as well as the workload for the older version of the application. The URIMAP resource for Version 1.1.0 of the application, called INDEX11, is dynamically generated in the CICS region and the resources for Version 1.1.0 of the web application are deployed in the Liberty JVM server. The CICS bundles for the original version of the application are still installed in the CICS region and the URIMAP resource called INDEX and the resources for Version 1.0.0 of the web application are still available. Therefore, both versions of the application are present and available in the CICS region, and are accessed by the different URLs that are specified on the different URIMAP resources.

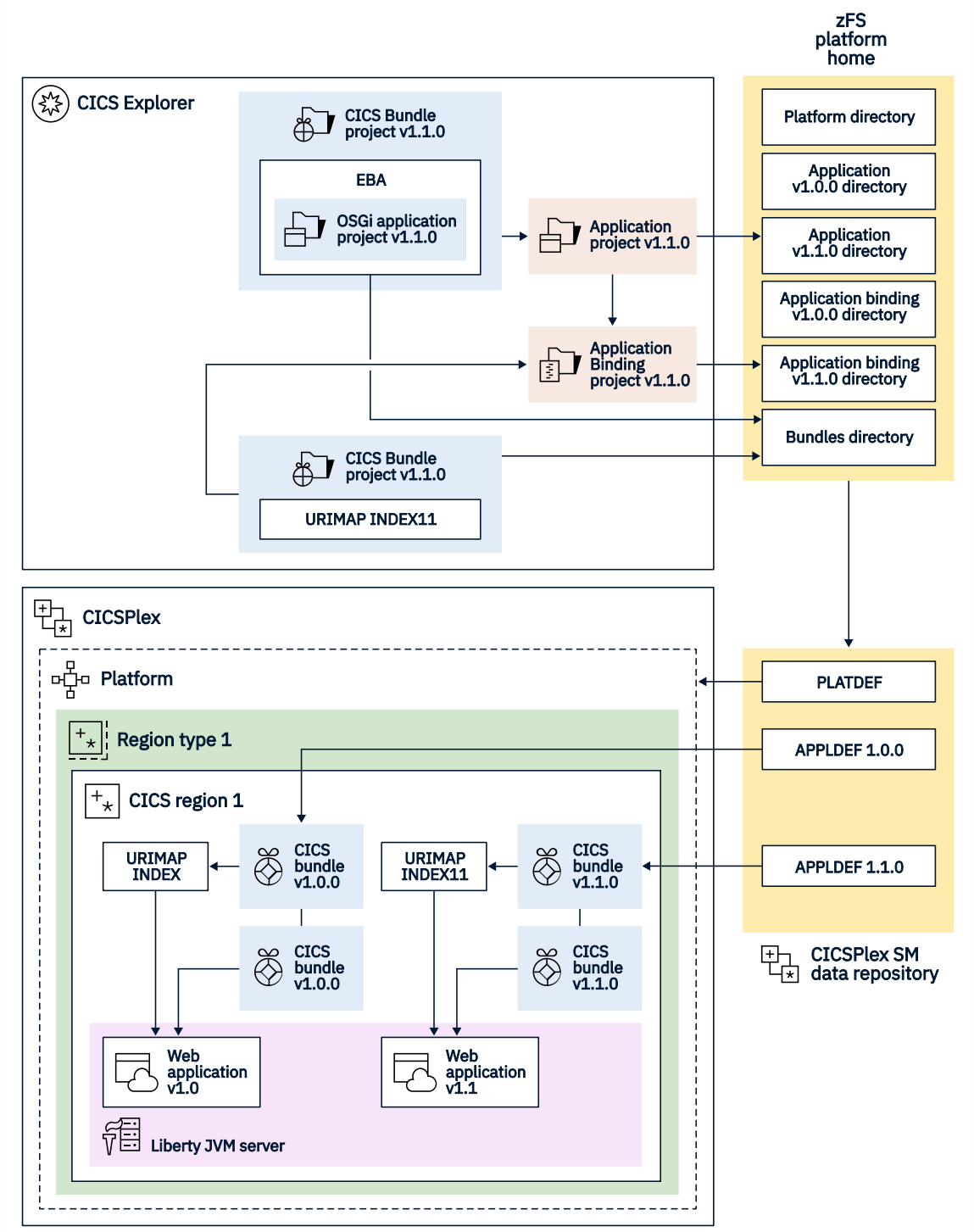


Figure 23. Example topology for a multi-versioned application, where one version of the application runs concurrently with another version

Assigning a version to your applications

In CICS TS, certain resources have versions associated with them. For example, an application in the context of cloud enablement requires a version and a name. A version must also be specified for a CICS bundle that is to be installed as part of either a CICS application or a CICS platform. CICS application bindings have a version too. These versions are different to the CICS product version, such as Version 5. Why is this version needed, and how should you use it?

You must apply a versioning policy to your CICS bundles, application bundles, and application bindings to deploy and manage updates in the CICS environment. You cannot use an existing version of an application bundle to install new versions of the CICS bundles for the application, and you cannot use an existing version of an application binding with a new version of an application bundle. You must update the versions of the application bundle and application binding whenever you update the CICS bundles for the application.

For cloud enablement, CICS suggests a versioning system based on *semantic versioning*, described in the technical paper Semantic Versioning. This versioning system is widely used in OSGi-based projects, such as Eclipse, and was first seen in CICS V4.2 with the Java class library for CICS (JCICS) API where it enables the author of an API or service to clearly describe the nature of a change in the implementation, so clients can understand any compatibility issues with earlier versions.

The version attribute used for a CICS TS application, a CICS bundle, or an OSGi bundle takes the form *<major>*.*<minor>*.*<micro>*: for example, 2.3.1 The version should be updated manually by whoever is responsible for maintaining the resource concerned, for example, the application developer. The version should be the first change made as part of a new activity.

The version part changes should be used in the following ways:

Micro
: For example, from 1.0.0 to 1.0.1. No external change (for example, a bug fix).

Minor
: for example, from 1.0.1 to 1.100.0. Compatible with an earlier version or an external change (for example, existing clients are unaffected).

  You might choose to increment the minor version by 100 instead of by 1. This technique is valuable when you want to run multiple versions of an application simultaneously. For example, take a scenario where you run version 1.0.0 of an application is in production and you add in version 1.1.0. For a while, both versions run at the same time without problems. Then, a bug is found in version 1.0.0 that requires a small change to the externals of the application. Ideally, you would increase the minor version number, because the externals have changed, but V1.1.0 is already in use. If you use version 1.100.0 when you added the new version of the application, you could fix the bug in V1.0.0 and increased its minor version number to reflect the change to the externals of the application. This technique is used by CICS TS to version the OSGi bundles that are included with the JCICS APIs.

Major
: For example, from 1.100.0 to 2.0.0. Change is incompatible with earlier versions (for example, a different format of file records or an operation is removed.)

When you change the version of an application, according to the principles of semantic versioning, the new version should reflect the largest change in a CICS bundle that is included in the application. For example, you might change one CICS bundle for an application from Version 1.0.1 to Version 1.0.2, which is a micro version change, and change another CICS bundle for the application from Version 1.2.0 to Version 1.3.0, which is a minor version change. The application bundle that includes these two CICS bundles should therefore have a minor version change, so if the application was previously at Version 2.5.1, it should change to Version 2.6.0.

Chapter 4. Designing applications

Use these basic concepts to help you design CICS applications

Changes are suggested that can improve performance and efficiency, but further guidance on programming for efficiency is provided in “Design for performance” on page 96.

The programming models implemented in CICS are inherited from those designed for 3270s, and exhibit many of the characteristics of conversational, terminal-oriented applications. There are basically three styles of programming model:

- Terminal-initiated, that is, the conversational model
- Distributed program link (DPL), or the RPC model
- START, that is, the queuing model.

Once initiated, the applications typically use these and other methods of continuing and distributing themselves, for example, with pseudoconversations, RETURN IMMEDIATE or DTP. The main difference between these models is in the way that they maintain state (for example, security), and hence state becomes an integral part of the application design. This presents the biggest problem when you attempt to convert to another application model.

A pseudoconversational model is mostly associated with terminal-initiated transactions and was developed as an efficient implementation of the conversational model. With increased use of 1-in and 1-out protocols such as HTTP, it is becoming necessary to add the pseudoconversational characteristic to the DPL or RPC model.

Related information

CICS programming roadmap

How tasks are started

Work is started in CICS - that is, tasks are initiated - from unsolicited input, or by automatic task initiation (ATI).

Automatic task initiation occurs when:

- An existing task asks CICS to create another one. The START command, the IMMEDIATE option on a RETURN command (discussed in “RETURN IMMEDIATE” on page 258), and the SEND PAGE command (in “The SEND PAGE command” on page 407) all do this.
- CICS creates a task to process a transient data queue (see “Automatic transaction initiation (ATI)” on page 305).
- CICS creates a task to deliver a message sent by a BMS ROUTE request (see “Message routing” on page 418). The CSPG tasks you see after using the CICS-supplied transaction CMSG are an example of this. CMSG uses a ROUTE command which creates a CSPG transaction for each target terminal in your destination list.

The primary mechanism for initiating tasks, however, is unsolicited input. When a user transmits input from a terminal which is not the ***principal facility*** of an existing task, CICS creates a task to process it. The terminal that sent the input becomes the principal facility of the new task.

Principal facility

CICS allows a task to communicate directly with only one terminal, namely its principal facility. CICS assigns the principal facility when it initiates the task, and the task "owns" the facility for its duration. No other task can use that terminal until the owning task ends. If a task needs to communicate with a terminal other than its principal facility, it must do so indirectly, by creating another task that has the terminal as its principal facility. This requirement arises most commonly in connection with printing, and how you can create such a task is explained in “Using CICS printers” on page 353.

© Copyright IBM Corp. 1974, 2023

Unsolicited inputs from other systems are handled in the same way: CICS creates a task to process the input, and assigns the conversation over which the input arrived as the principal facility. (Thus a conversation with another system may be either a principal or alternate facility. In the case where a task in one CICS region initiates a conversation with another CICS region, the conversation is an alternate facility of the initiating task, but the principal facility of the partner task created by the receiving system. By contrast, a terminal is always the principal facility.)

Alternate facility
Although a task may communicate directly with only one terminal, it can also establish communications with one or more remote systems. It does this by asking CICS to assign a conversation with that system to it as an **alternate facility** . The task "owns" its alternate facilities in the same way that it owns its principal facility. Ownership lasts from the point of assignment until task end or until the task releases the facility.

Not all tasks have a principal facility. Tasks that result from unsolicited input always do, by definition, but a task that comes about from automatic task initiation may or may not need one. When it does, CICS waits to initiate the task until the requested facility is available for assignment to the task.

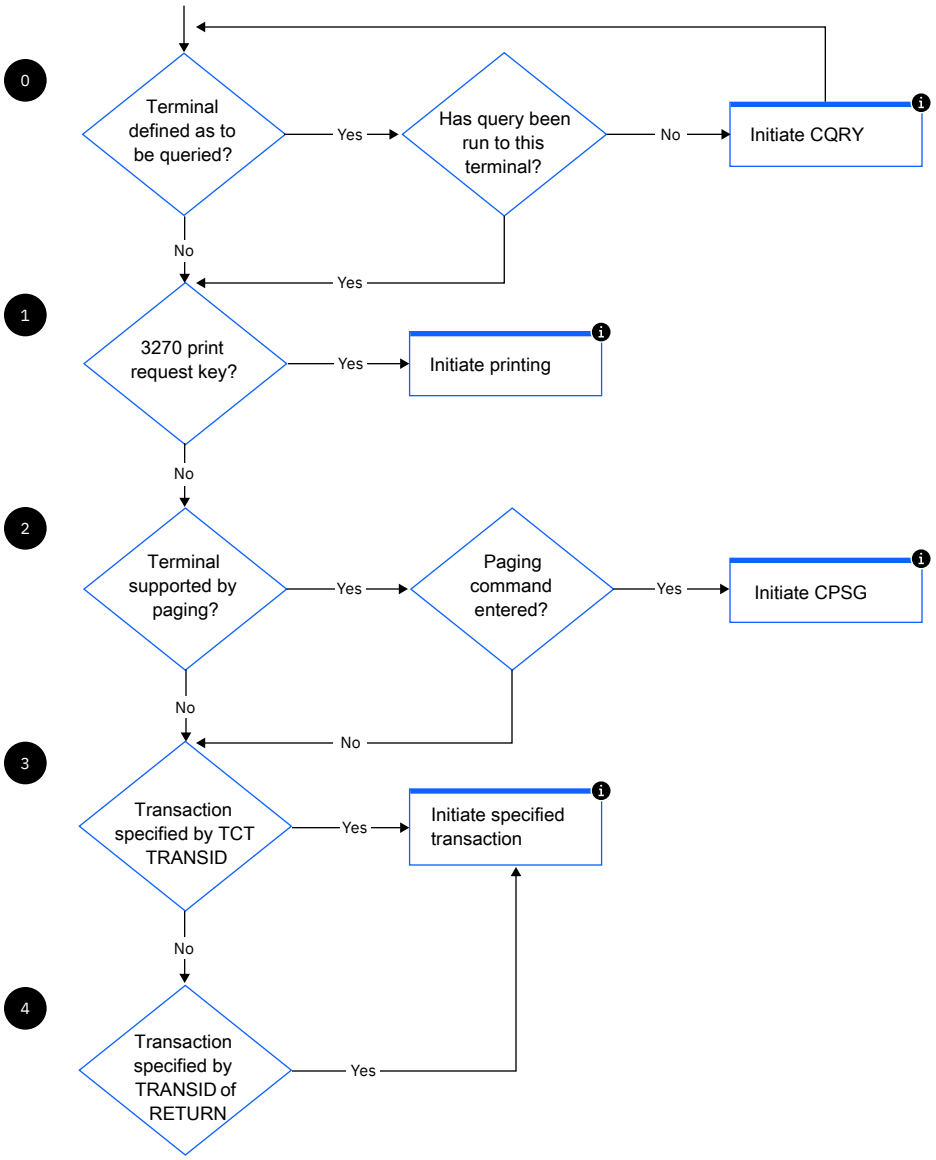
Which transaction?

On receipt of an unsolicited input, how does CICS determine which transaction to run? Typically the previous task with the same principal facility determines what transaction CICS runs next, using the TRANSID option on its final RETURN.

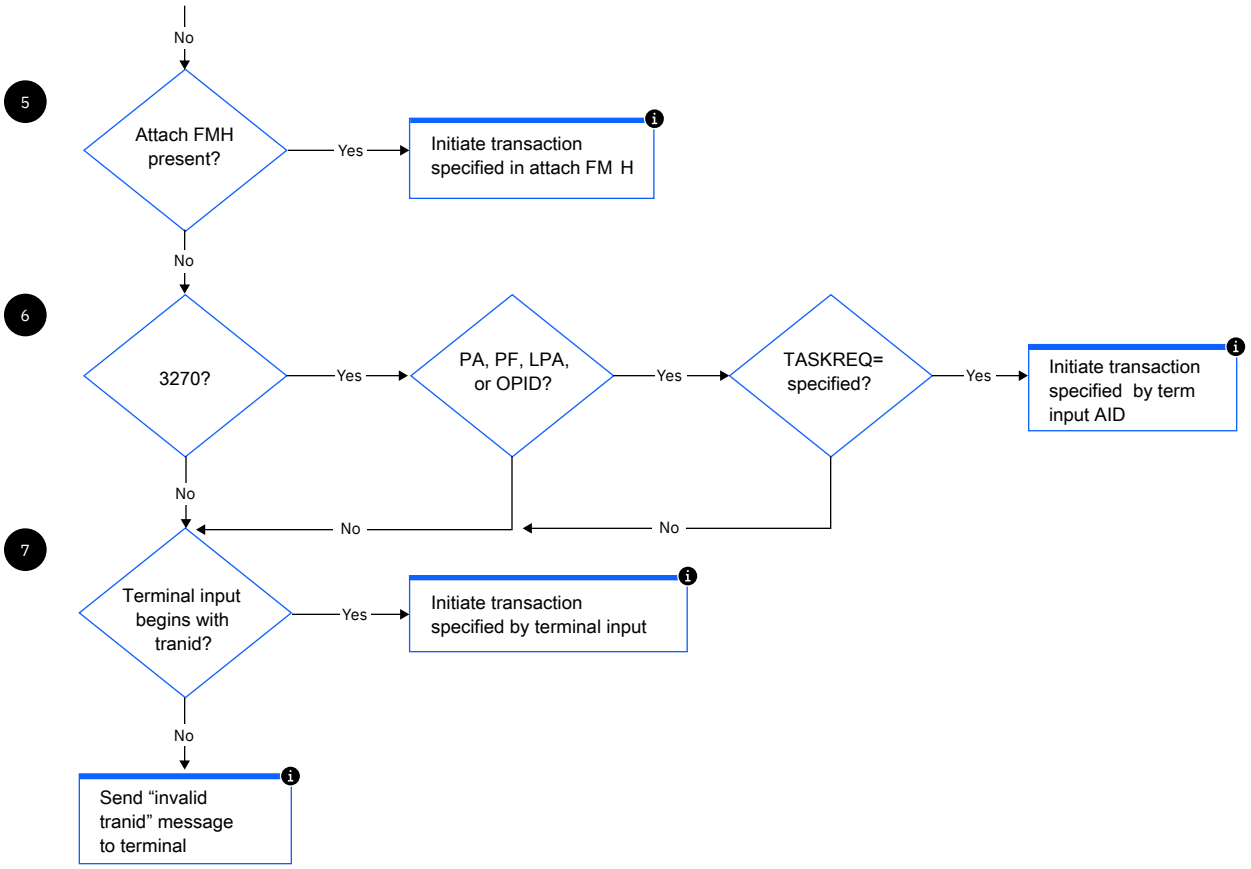
This situation is almost always the case in a pseudoconversational transaction sequence, and typically in menu-driven applications as well. Failing that, and in any case to get a sequence started, CICS interprets the first few characters of the input as a transaction code. However, it is more complicated than that; the exact process goes as follows. The step numbers indicate the order in which the tests are made and refer to Figure 24 on page 74 , a diagram of this logic.

Figure 24. Determining which transaction to execute

Test sequence



Test sequence



Global user exit XZCATT in module DFHZATT is invoked at these points.

0.

On the first input from a terminal, CICS sometimes schedules a preliminary task before creating one to process the input. This task runs the CICS-supplied query transaction, CQRY, which causes the terminal to transmit an encoded description of some of its hardware characteristics—extended attributes, character sets, and so on.

CQRY allows the system programmer to simplify maintenance of the terminal network by omitting these particulars from the terminal definitions. It occurs only if the terminal definition so specifies, and has no effect on the subsequent determination of what transaction to use to process the input, which goes as follows.

1.

If the terminal is a 3270 and the input is the "print request key", the CICS-supplied transaction that prints the contents of the screen, CSPP, is initiated. See CICS print key in "Printing display screens" on page 357 for more information about this feature. For this purpose, a *3270 logical unit* or any other device that accepts the 3270 data stream counts as a 3270.

2.

If full BMS support is present, the terminal is of a type supported by BMS terminal paging, and the input is a paging command, the CICS-supplied transaction CSPG is initiated to process the request. BMS support levels are explained in "BMS support levels" on page 367 , and the same section contains a list of the terminals that BMS supports. The PGRET, SKRxxxx, PGCHAIN, PGCOPY, and PGPURGE options in the system initialization table define the paging commands. As paging requires full BMS, this step is skipped if the CICS system contains less than that level.

3. If the terminal definition indicates that a specific transaction should be used to process all unsolicited inputs from that terminal, the indicated transaction is executed. (If present, this information appears in the TRANSACTION attribute of the TERMINAL definition.)
4. If the previous task at the terminal specified the TRANSID option of the RETURN command that ended it, the transaction named is executed.
5. If an attach function management header is present in the input, the attach names in the header are converted to a four-character CICS transaction identifier, and that transaction is executed.
6. If the terminal is a 3270, and the attention identifier is defined as a transaction, that transaction is executed. Attention keys in "Input from a 3270 terminal" on page 284 explains attention identifiers. You define one as a transaction identifier with the TASKREQ attribute of the corresponding TRANSACTION definition.
7. If all the preceding tests fail, the initial characters of the input are used to identify the transaction to be executed. The characters used are the first ones (up to four) after any control information in the data stream and before the first field separator character or the next 3270 Control Character (X'00' to X'3F') . Field separators are defined in the FLDSEP option of the system initialization table (the default is a blank).

   If there are no such characters in the input, as occurs when you use the CLEAR key, for example, or if there is no transaction definition that matches the input, CICS cannot determine what transaction to execute and sends an invalid transaction identification message to the terminal.

Note: This logic for deciding which transaction to execute applies only to tasks initiated to process unsolicited inputs. For automatic transaction initiation, the transaction is always known. You specify it in the TRANSID option when you create a task with a START or RETURN IMMEDIATE. Similarly, you specify what transaction should be used to process a transient data queue in the queue definition. Tasks created to route messages always execute the CICS-supplied transaction CSPG.

Separating business and presentation logic

In general, it is good practice to split applications into a part containing the business code that is reusable, and a part responsible for presentation to the client. This technique enables you to improve performance by optimizing the parts separately, and allows you to reuse your business logic with different forms of presentation.

When separating the business and presentation logic, you need to consider the following:

- Avoid affinities between the two parts of the application.
- Be aware of the DPL-restricted API; see Exception conditions for LINK command for details.
- Be aware of hidden presentation dependencies, such as EIBTRMID usage.

Figure 25 on page 78 illustrates a simple CICS application that accepts data from a user, updates a record in a file, and sends a response back to the user. The transaction that runs this program is the second in a pseudoconversation. The first transaction has sent a BMS map to the user's terminal, and the second transaction reads the data with the EXEC CICS RECEIVE MAP command, updates the record in the file, and sends the response with the EXEC CICS SEND MAP command.

The EXEC CICS RECEIVE and EXEC CICS SEND MAP commands are part of the transaction's presentation logic, while the EXEC CICS READ UPDATE and EXEC CICS REWRITE commands are part of the business logic.

```
..
EXEC CICS RECEIVE MAP
..
EXEC CICS READ UPDATE
..
EXEC CICS REWRITE
..
EXEC CICS SEND MAP
..
```

Figure 25. CICS functions in a single application program

A sound principle of modular programming in CICS application design is to separate the presentation logic from the business logic, and to use a communication area and the EXEC CICS LINK command to make them into a single transaction. Figure 26 on page 78 and Figure 27 on page 78 illustrate this approach to application design.

```
..
EXEC CICS RECEIVE MAP
..
EXEC CICS LINK..
..
EXEC CICS SEND MAP
..
```

Figure 26. Presentation logic

```
..
EXEC CICS ADDRESS COMMAREA
..
EXEC CICS READ UPDATE
..
EXEC CICS REWRITE
..
EXEC CICS RETURN..
```

Figure 27. Business logic

Once the business logic of a transaction has been isolated from the presentation logic and given a communication area interface, it is available for reuse with different presentation methods. For example, you could use “Distributed program link (DPL)” on page 211 to implement a two-tier model, or CICS Web support with the CICS business logic interface, where the presentation logic is HTTP-based.

Multithreading: Reentrant, quasi-reentrant, and threadsafe programs

Multithreading is a technique where a single copy of an application program can be processed by several transactions concurrently. For example, one transaction can begin to execute an application program. When an **EXEC CICS** command is reached, causing a CICS WAIT and call to the dispatcher, another transaction can then execute the same copy of the application program.

Compare this technique with single-threading, which is the execution of a program to completion: processing of the program by one transaction is completed before another transaction can use it.

Multithreading requires that all CICS application programs are quasi-reentrant; that is, they must be serially reusable between entry and exit points. CICS application programs that use the EXEC CICS interface obey this rule automatically. For COBOL, C, and C++ programs, reentrancy is ensured by obtaining a fresh copy of working storage each time the program is invoked. You should always use the RENT option on the compile or pre-link utility, even for C and C++ programs that do not have writable statics and that are naturally reentrant. Temporary variables and DFHEIPTR fields inserted by the CICS translator are usually defined as writable static variables and require the RENT option. For these programs to stay reentrant, variable data should not appear as static storage in PL/I, or as a DC in the program CSECT in assembler language.

As well as requiring that your application programs are compiled and link-edited as reentrant, CICS also identifies programs as being either quasi-reentrant or threadsafe; these attributes are set on the PROGRAM resource definition. The following table shows you the CONCURRENCY and API settings that are available on the program, and the type of TCB that the application program is run on.

Table 13. Combination of PROGRAM CONCURRENCY and API settings and the type of TCB used.

CONCURRENCY attribute	**API attribute**	**CICS TCB**
CONCURRENCY(QUASIRENT)	API(CICSAPI)	The application program always runs under the CICS quasi-reentrant (QR TCB). If a switch to an open TCB occurs for a resource manager, CICS always returns to QR TCB before returning to the application.
CONCURRENCY(QUASIRENT)	API(OPENAPI)	Invalid combination. OPENAPI programs cannot run on a QR TCB.
CONCURRENCY(THREADSAFE)	API(CICSAPI)	The application program can run on a QR TCB or an open TCB. If a switch to an open TCB occurs for a resource manager, CICS stays on the open TCB when returning to the application. If a switch to the QR TCB occurs, CICS stays on QR TCB when returning to the application.
CONCURRENCY(THREADSAFE)	API(OPENAPI)	Same as CONCURRENCY(REQUIRED) API(OPENAPI).
CONCURRENCY(REQUIRED)	API(CICSAPI)	The application program always runs on an open TCB. CICS services do not require TCB key matching so the application always runs on a L8 open TCB. An OPENAPI resource manager uses an L8 TCB so no TCB switch is required if it is invoked. If a switch to the QR TCB occurs, CICS returns to the open TCB when returning to the application.
CONCURRENCY(REQUIRED)	API(OPENAPI)	The application program always runs on an open TCB. The key of the TCB must match the execution key of the program. CICS uses an L9 TCB if EXECKEY(USER) is set and an L8 TCB if EXECKEY(CICS) is set. If the application is user key and an OPENAPI resource manager is invoked then a switch occurs from the L9 TCB to the L8 TCB. CICS returns to L9 TCB before returning to the application. If a switch to the QR TCB occurs, CICS returns to the open TCB when returning to the application.

Quasi-reentrant application programs

A *quasi-reentrant* program is a program that is in a consistent state when control is passed to it, both on entry, and before and after each **EXEC CICS** command. Such quasi-reentrancy guarantees that each invocation of an application program is unaffected by previous runs, or by concurrent multithreading through the program by multiple CICS tasks.

CICS runs user programs under a CICS-managed task control block (TCB). If your programs are defined as quasi-reentrant (on the CONCURRENCY attribute of the program resource definition), CICS always invokes them under the CICS quasi-reentrant (QR TCB). The requirements for a quasi-reentrant program in a multithreading context are less stringent than if the program were to run concurrently on multiple TCBs.

CICS requires that an application program is reentrant so that it guarantees consistent conditions. In practice, an application program might not be truly reentrant; CICS expects "quasi-reentrancy".

For example, application programs could modify their executable code, or the variables defined within the program storage, but these changes must be undone, or the code and variables reinitialized, before there is any possibility of the task losing control and another task running the same program.

CICS quasi-reentrant user programs (application programs, user-replaceable modules, global user exits, and task-related user exits) are given control by the CICS dispatcher under the QR TCB. When running under this TCB, a program can be sure that no other quasi-reentrant program can run until it relinquishes control during a CICS request, at which point the user task is suspended, leaving the program still "in use" . The same program can then be reinvoked for another task, which means the application program can be in use concurrently by more than one task, although only one task at a time can be running.

To ensure that programs cannot interfere with each other's working storage, CICS obtains a separate copy of working storage every time an application program runs. For example, if a user application program is being used by 11 user tasks, there are 11 copies of working storage in the appropriate dynamic storage area (DSA).

Quasi-reentrancy allows programs to access globally shared resources, for example the CICS common work area (CWA), without the need to protect those resources from concurrent access by other programs. Such resources are effectively locked exclusively to the running program, until it issues its next CICS request. For example, an application can update a field in the CWA without using compare and swap (CS) instructions or locking (enqueuing on) the resource.

Note: The CICS QR TCB provides protection through exclusive control of global resources only if all user tasks that access those resources run under the QR TCB. It does not provide automatic protection from other tasks that run concurrently under another open TCB.

Take care if a program involves lengthy calculations: because an application program retains control from one EXEC CICS command to the next, the processing of other transactions on the QR TCB is excluded. However, you can use the task-control SUSPEND command to allow other transaction processing to proceed; see “Task control” on page 289 for details. Runaway task time interval is controlled by the transaction definition and the system initialization parameter **ICVR**. CICS purges a task that does not return control before expiry of the specified interval.

Threadsafe programs

In the CICS open transaction environment (OTE), when application programs, task-related user exits (TRUEs), global user exit programs, and user-replaceable modules are defined to CICS as threadsafe, they can run concurrently on open TCBs in the open transaction environment (OTE).

Accessing the open transaction environment

Applications that involve a task-related user exit (TRUE) enabled using the OPENAPI option on the **ENABLE PROGRAM** command are automatically involved with the open transaction environment. These applications can gain performance benefits from being threadsafe. The CICS Db2 task-related user exit used by CICS applications that access Db2 resources is an open API TRUE, so CICS Db2 applications can gain performance benefits from being threadsafe. For more details about threadsafe programming for CICS Db2 applications, see Enabling CICS Db2 applications to use the open transaction environment (OTE) through threadsafe programming.

For other user application programs, PLT programs, user replaceable modules, or task-related user exits, you can opt to use the open transaction environment by defining them as OPENAPI programs, in which case they must be threadsafe. For more details about threadsafe programming for OPENAPI programs, see “OPENAPI programs” on page 86.

If you define your program as CONCURRENCY(REQUIRED) it always runs on an open TCB. The type of open TCB used depends on the API setting. For CICSAPI programs, CICS uses an L8 open TCB regardless of the execution key of the program. For OPENAPI programs, CICS uses an L9 TCB if EXECKEY(USER) is set and an L8 TCB if EXECKEY(CICS) is set. REQUIRED is applicable to user application programs, PLT programs, and user replaceable modules. For global user exit programs and task-related user exit programs, if you specify CONCURRENCY(REQUIRED), CICS treats the program as if you had specified CONCURRENCY(THREADSAFE).

Global user exits cannot be defined as OPENAPI programs, but if you use the THREADSAFE option on the **ENABLE PROGRAM** command for a global user exit, it is enabled as a threadsafe program and can run on the same open TCB as an application that uses it. If the **ENABLE PROGRAM** command does not specify the CONCURRENCY or API options then the options on the program definition are used.

Serialization techniques

An application that is running on an open TCB cannot rely on quasi-reentrancy to protect shared resources from concurrent access by another program. Furthermore, quasi-reentrant programs might also be placed at risk if they access shared resources that can also be accessed by a user task running concurrently under an open TCB. The techniques used by user programs to access shared resources must therefore take into account the possibility of simultaneous access by other programs.

To gain the performance benefits of the open transaction environment while maintaining the integrity of shared resources, serialization techniques must be used to prohibit concurrent access to shared resources. Programs that use appropriate serialization techniques when accessing shared resources are described as threadsafe.

For most resources, such as files, transient data queues, temporary storage queues, and Db2 tables, CICS processing automatically ensures access in a threadsafe manner. Some of the CICS commands that operate on these resources are coded to use appropriate serialization techniques that allow them to run on open TCBs; that is, they are threadsafe commands. Where this is not the case, CICS ensures threadsafe processing by forcing a switch to the QR TCB, so that access to the resources is serialized regardless of the behavior of the command.

For any other resources that are accessed directly by user programs, such as shared storage, it is the responsibility of the user program to ensure threadsafe processing. For information about threadsafe programming for user application programs, see “Making applications threadsafe” on page 82.

TCB switching

Threadsafe programs achieve performance benefits by remaining on an open TCB, rather than switching between the open TCB and the QR TCB. When a program is defined as threadsafe and is running on an open TCB, TCB switching from the open TCB to the QR TCB occurs in the following circumstances:

- From the point a task is attached to the point CICS determines that an open TCB is required, for non-Liberty transactions, all processing takes place on the QR TCB; for Liberty transactions, a T8 TCB is used.
- If the program issues any **EXEC CICS** commands that are not threadsafe, CICS switches back from the open TCB to the QR TCB to run the nonthreadsafe code. If the program is defined as OPENAPI or CONCURRENCY(REQUIRED), CICS then switches back to the open TCB to continue running the application logic. If the program is not defined as OPENAPI or CONCURRENCY(REQUIRED), it continues to run on the QR TCB. For a CICS Db2 application, if the program is not defined as OPENAPI or CONCURRENCY(REQUIRED) and does not make any further Db2 requests, the switch back to the QR TCB is a disadvantage because it increases the usage of your QR TCB for the time taken to run any remaining application code. However, if the program makes any further Db2 requests, CICS must switch back again to the open TCB.
- If the program calls a task-related user exit program that is not defined as threadsafe, CICS switches back to the QR TCB and gives control to the task-related user exit program. When the task-related user exit program completes processing, the situation is the same as after a nonthreadsafe EXEC CICS command: an OPENAPI or CONCURRENCY(REQUIRED) program switches back to the open TCB, and a program not defined as OPENAPI or CONCURRENCY(REQUIRED) continues to run on the QR TCB.
- When the program issues a threadsafe CICS command or makes a Db2 request, a global user exit might be invoked as part of the processing for the command or request. If a global user exit program is used that is not defined as threadsafe, CICS switches back to the QR TCB and gives control to the global user exit program. When the user exit program completes processing, CICS switches back to the open TCB to continue processing the threadsafe CICS command or to complete the Db2 request.

- When the program completes, CICS switches back to the QR TCB for task termination. This switch is always necessary.

Threadsafe considerations for statically or dynamically called routines

If you define a program with CONCURRENCY(THREADSAFE) or CONCURRENCY(REQUIRED), all routines that are statically or dynamically called from that program (for example, COBOL routines) must also be coded to threadsafe standards.

When an **EXEC CICS LINK** command is used to link from one program to another, the program link stack level is incremented. However, a routine that is statically called, or dynamically called, does not involve passing through the CICS command level interface, and so does not cause the program link stack level to be incremented. With COBOL routines, for a static call a simple branch and link is involved to an address that is resolved at link-edit time. For a dynamic call, although there is a program definition involved, this is required only to allow Language Environment® to load the program. After that, a simple branch and link is executed. So when a routine is called by either of these methods, CICS does not regard this as a change of program. The program that called the routine is still considered to be executing, and so the program definition for that program is still considered to be the current one.

If the program definition for the calling program states CONCURRENCY(THREADSAFE) or CONCURRENCY(REQUIRED), the called routine must also comply with this specification. Programs with the CONCURRENCY(THREADSAFE) or CONCURRENCY(REQUIRED) attribute remain on an open TCB when they return from a Db2 call, and this is not appropriate for a program that is not threadsafe. For example, consider the situation where the initial program of a transaction, program A, issues a dynamic call to program B, which is a COBOL routine. Because the CICS command level interface was not involved, CICS is unaware of the call to program B, and considers the current program to be program A. Program B issues a Db2 call. On return from the Db2 call, CICS must determine whether the program can remain on the open TCB, or whether the program must switch back to the QR TCB to ensure threadsafe processing. To do this, CICS examines the CONCURRENCY attribute of what it considers to be the current program, which is program A. If program A is defined as CONCURRENCY(THREADSAFE) or CONCURRENCY(REQUIRED), then processing can continue on the open TCB. In this scenario program B is executing, so if processing is to continue safely, program B must be coded to threadsafe standards.

Making applications threadsafe

When you make an application program threadsafe, you can use the open transaction environment, avoid TCB switching, and gain performance benefits.

Before you begin

To use threadsafe application programs, ensure that the system initialization parameter **FORCEQR** is not set to YES. **FORCEQR** forces programs defined as threadsafe to run on the QR TCB, and it might be set to YES as a temporary measure while problems connected with threadsafe-defined programs are investigated and resolved.

Also, select an appropriate setting for the system initialization parameter **FCQRONLY** in your file-owning CICS regions. If **FCQRONLY** is set to YES, CICS forces all file control requests in the CICS region to run on the QR TCB.

- If you use IPIC connections to function ship file control requests to remote CICS TS 4.2 or later regions, to improve performance for those connections set **FCQRONLY** to NO in the file-owning regions.
- If you use only MRO links or ISC over SNA connections, or your file-owning regions are earlier than CICS TS 4.2 , set **FCQRONLY** to YES in the file-owning regions.

If you are using CICS intercommunication to make requests for functions or programs to run in remote CICS systems, choose IP interconnectivity (IPIC) over TCP/IP connections between the CICS systems to provide optimal support for threadsafe applications. With IPIC connections, CICS uses an open TCB to run the mirror program that manages the request on the remote CICS system, providing improved throughput. With other connection types, CICS does not use an open TCB to run the mirror program. The **EXEC CICS**

LINK command, used for distributed program link (DPL), is threadsafe for IPIC connections to remote CICS regions where a long-running mirror is used, but not for other connection types.

About this task

Threadsafe programs explains what it means for a program to be threadsafe, and the circumstances when TCB switching takes place between open TCBs and the QR TCB.

To make an application program threadsafe and enable it to remain on an open TCB, use the following procedure.

Procedure

1. Define the program to CICS as threadsafe, by specifying `CONCURRENCY(THREADSAFE)` in the PROGRAM resource definition.

   For a program that is defined as OPENAPI, CICS requires the `CONCURRENCY(THREADSAFE)` option. Only code that has been defined as threadsafe is permitted to run on open TCBs. By defining a program to CICS as threadsafe, you are specifying only that the application logic is threadsafe, not that all the **EXEC CICS** commands included in the program are threadsafe. CICS can ensure that EXEC CICS commands are processed safely by using TCB switching, but, to permit your program to run on an open TCB, CICS needs you to guarantee that your application logic is threadsafe.

   Alternatively, you can define the program as `CONCURRENCY(REQUIRED)` to enable your program to run from the start on an open TCB. Programs defined as `CONCURRENCY(REQUIRED)` must be coded to threadsafe standards as they must always run on an open TCB. The type of open TCB used depends on the API setting.
2. Ensure that the program logic, that is, the native language code between the **EXEC CICS** commands, is threadsafe.

   If you define a program to CICS as threadsafe but include application logic that is not threadsafe, the results are unpredictable, and CICS cannot protect your program from the possible consequences. To make your program logic threadsafe, you must use appropriate serialization techniques when accessing shared resources, to prohibit concurrent access to those resources. When you use **EXEC CICS** commands to access resources such as files, transient data queues, temporary storage queues, and Db2 tables, CICS ensures threadsafe processing, but for any resources that are accessed directly by user programs, such as shared storage, the user program must ensure threadsafe processing.

   Typical examples of shared storage are the CICS CWA, the global work areas for global user exits, and storage acquired explicitly by the application program with the shared option. Check whether your application programs use these types of shared storage by looking for occurrences of the following **EXEC CICS** commands:

   - ADDRESS CWA
   - EXTRACT EXIT
   - GETMAIN SHARED

   The load module scanner utility includes a sample table, DFHEIDTH, to help you identify CICS commands that give access to shared storage. Although some of these commands are themselves threadsafe, they all give access to global storage areas, so the application logic that follows these commands and uses the global storage areas can potentially not be threadsafe. To ensure that it is threadsafe, an application program must include the necessary synchronization logic to guard against concurrent update.

   Tip: When identifying programs that use shared resources, also include any program that modifies itself. Such a program is effectively sharing storage and you must consider it at risk.

   You can use the following techniques to provide threadsafe processing when accessing a shared resource:

   - Retry access, if the resource has been changed concurrently by another program, using the compare and swap instruction.

- Enqueue on the resource, to obtain exclusive control and ensure that no other program can access the resource, using one of the following techniques:
  - An **EXEC CICS ENQ** command, in an application program.
  - An **XPI ENQUEUE** function call to the CICS enqueue (NQ) domain, in a global user exit program.
  - An MVS service such as ENQ, in an open API task-related user exit only when L8 TCBs are enabled for use. Note that the use of MVS services in an application that can run under the QR TCB might result in performance degradation because of the TCB being placed in a wait.
- Perform accesses to shared resources only in a program that is defined as quasi-reentrant, by linking to the quasi-reentrant program using the **EXEC CICS LINK** command. This technique applies to threadsafe application programs and open API task-related user exits only. A linked-to program defined as quasi-reentrant runs under the QR TCB and can take advantage of the serialization provided by CICS quasi-reentrancy. Note that, even in quasi-reentrant mode, serialization is provided only for as long as the program retains control and does not wait.
- Place all transactions that access the shared resource into a restricted transaction class (TRANCLASS), one that is defined with the number of active tasks specified as MAXACTIVE(1) . This approach effectively provides a very coarse locking mechanism, but might have a severe impact on performance.

Note: Although the term threadsafe is defined in the context of individual programs, a user application as a whole can be considered threadsafe only if all the application programs that access shared resources obey the rules. A program that is written correctly to threadsafe standards cannot safely update shared resources if another program that accesses the same resources does not obey the threadsafe rules.

3. For best performance, ensure that the program uses only threadsafe **EXEC CICS** commands.

   If you include an **EXEC CICS** command that is not threadsafe in a program that is defined as threadsafe and running on an open TCB, CICS switches back from the open TCB to the QR TCB to ensure that the command is processed safely. The results of your application are not affected, but its performance might be affected.

   The commands that are threadsafe are indicated in the command descriptions of the CICS API and SPI command topics with the statement "This command is threadsafe" . They are also listed in Threadsafe commands and Threadsafe SPI commands.

   The load module scanner utility includes a sample table, DFHEIDNT, to help identify any CICS commands in your applications that are not threadsafe.

4. If any user exit programs are in the execution path used by the program, for best performance ensure that they are also coded to threadsafe standards and defined to CICS as threadsafe. These exits might be dynamic plan exits, global user exits, or task-related user exits. Also check that user exit programs supplied by any vendor software are coded to threadsafe standards and defined to CICS as threadsafe.

   A threadsafe user exit program can be used on the same open TCB as a threadsafe application that calls it, and it can use non-CICS APIs without having to create and manage subtask TCBs, and exploit the open transaction environment for itself. If any user exit programs in the execution path used by the program are not threadsafe, CICS switches to the QR TCB to run them, which might be detrimental to the application's performance.

   Be aware of the following important user exits:

   - The global user exits XEIIN and XEIOUT are called before and after **EXEC CICS** commands.
   - The global user exit XPCFTCH is called before a program defined to CICS receives control.
   - For CICS Db2 requests, the CICS Db2 task-related user exit DFHD2EX1 is threadsafe. Other important exits for CICS Db2 requests include the default dynamic plan exit DSNCUEXT, which is not defined as threadsafe, the alternative dynamic plan exit DFHD2PXT, which is defined as threadsafe, and the global user exits XRMIIN and XRMIOUT.
   - If you function ship CICS file control, transient data, or temporary storage requests to a remote CICS region over an IPIC connection, the requests are threadsafe and can run in the remote CICS region on an open TCB. Any global user exit programs that are called in the remote CICS region for file

control, transient data, or temporary storage requests must be enabled as threadsafe programs for the best performance.

a) To define a user exit program to CICS as threadsafe, you can specify appropriate attributes in its PROGRAM resource definition.

   - For a task-related user exit program, specify OPENAPI and THREADSAFE, or just THREADSAFE.
   - For a global user exit program, you cannot use OPENAPI, but you can specify THREADSAFE.

   If you specify CONCURRENCY(REQUIRED) on a global user exit program or task-related user exit program, CICS treats the program as if you had specified **CONCURRENCY(THREADSAFE)**.

b) Alternatively, to define a user exit program to CICS as threadsafe, you can specify appropriate options when you enable the program by using the **EXEC CICS ENABLE PROGRAM** command.

   - For a task-related user exit program, specify OPENAPI or THREADSAFE.
   - For a global user exit program, you cannot use OPENAPI, but you can specify THREADSAFE.

   When you enable an exit program using the OPENAPI or THREADSAFE option, this action indicates to CICS that the program logic is threadsafe, so CICS overrides the CONCURRENCY setting on the program definition for the exit and treats the exit program as threadsafe.

c) To define a first-phase PLT global user exit program as threadsafe, specify THREADSAFE on the **EXEC CICS ENABLE PROGRAM** command.

   To ensure that global user exit programs (such as those that run at the recovery exit points) are available as early as possible during CICS initialization, it is common practice to enable them from first-phase PLT programs. Because first-phase PLT programs run so early in CICS initialization, you cannot use installed PROGRAM resource definitions or the program autoinstall user program to define the exit programs. CICS automatically installs exit programs that are enabled from first-phase PLT programs with **CONCURRENCY(QUASIRENT)** . However, the setting on the **EXEC CICS ENABLE PROGRAM** command overrides the **CONCURRENCY(QUASIRENT)** setting on the system autoinstalled program definition.

CONCURRENCY(REQUIRED) programs

Defining an application program as CONCURRENCY(REQUIRED) means that from the start of the program, it always runs on an open task control block (open TCB) instead of the main CICS quasi-reentrant TCB (QR TCB). If CICS must switch to the QR TCB to process an **EXEC CICS** command, CICS switches back to the open TCB before returning control to the application program.

The type of open TCB used depends on what APIs the program is to use:

- If the program uses only CICS supported APIs (including access to external resource managers such as Db2, IMS , and IBM MQ), it must be defined with program attribute API(CICSAPI). In this case CICS always uses an L8 open TCB, irrespective of the execution key of the program, because CICS commands do not rely on the key of the TCB.
- If the program is to use other non- CICS APIs, it must be defined with program attribute API(OPENAPI). In this case CICS uses an L9 TCB or an L8 TCB depending on the execution key of the program. This is to allow the non-CICS APIs to operate correctly.

Global user exits cannot be defined as CONCURRENCY(REQUIRED), but if you enable them using the THREADSAFE option on the ENABLE PROGRAM command, they can run on an open TCB when necessary.

Task-related user exits (TRUEs) can be defined or enabled as CONCURRENCY(REQUIRED), in which case, they must be written to threadsafe standards. A TRUE that is defined as CONCURRENCY(REQUIRED) API(OPENAPI) always runs on an L8 TCB. A TRUE that is defined as CONCURRENCY(REQUIRED) API(CICSAPI) can run on a T8, L8, or X8 TCB, depending on the need of the application environment. The CICS-Db2 TRUE is enabled as CONCURRENCY(REQUIRED) API(CICSAPI).

Existing threadsafe CICS-Db2 applications that are defined as THREADSAFE CICSAPI (to take advantage of the performance gains of being able to run on the same TCB as the Db2 call), can be further enhanced by defining them as REQUIRED CICSAPI. This definition means that the programs can run on an L8 open

TCB immediately, without waiting for the first Db2 call to move them on to the open TCB. The additional benefit achieved depends on how many, if any, non-threadsafe CICS commands the application executes.

OPENAPI programs

The open transaction environment (OTE) is an environment where CICS application code can use non-CICS services inside the CICS address space, without interference from other transactions. You can use the OTE by defining user application programs, PLT programs, user replaceable modules, or task-related user exits (TRUEs) as OPENAPI programs, by using the OPENAPI attribute in the PROGRAM resource definitions.

Defining a program as an OPENAPI program by using the attribute API(OPENAPI) means that from the start of the program, it always runs on an L8 or L9 mode open task control block (open TCB) instead of the main CICS quasi-reentrant TCB (QR TCB). Global user exits cannot be defined as OPENAPI programs, but if you enable them using the THREADSAFE option on the **ENABLE PROGRAM** command, they can be run on an open TCB when necessary.

Moving application workloads off the QR TCB onto multiple open TCBs gives the possibility of achieving better throughput, particularly with CPU-intensive programs. You can use other non- CICS APIs, but you must note that the use of non-CICS APIs within CICS is entirely at the discretion and risk of the user. No testing of non-CICS APIs within CICS has been undertaken and the use of such APIs is not supported by IBM Service.

OPENAPI programs must be *threadsafe* in order to run on an open TCB. The requirements for a threadsafe program are as follows:

1. The program must be defined to CICS as CONCURRENCY(REQUIRED) meaning that the program is required to run on an open TCB.
2. The logic of the program, that is, the native language code between the EXEC CICS commands, must be threadsafe. If you define a program to CICS to run on an open TCB but include application logic that is not threadsafe, the results are unpredictable, and CICS is not able to protect you from the possible consequences.
3. For best performance, the program must use only threadsafe EXEC CICS commands. If you include a non-threadsafe EXEC CICS command in a program that is running on an open TCB, CICS switches back from the open TCB to the QR TCB to ensure that the command is processed safely. The TCB switching might be detrimental to the performance of the application.
4. For best performance, any user exit programs in the execution path used by the program must also be coded to threadsafe standards and defined to CICS as threadsafe. If any user exit programs in the execution path used by the program are not threadsafe, CICS switches to the QR TCB to run them, which might be detrimental to the performance of the application.

“Threadsafe programs” on page 80 explains the requirements for threadsafe programs in more detail.

OPENAPI programs have some additional obligations and restrictions. For example, they must ensure that all non-CICS resources acquired specifically on behalf of the terminating task are freed, and they must not use certain MVS system services. Threadsafe restrictions for OPENAPI programs explains these requirements.

Candidate programs for defining as REQUIRED OPENAPI (assuming their application logic is threadsafe) are those programs that want to use other non-CICS APIs at their own risk.

TCBs for OPENAPI programs

The following TCBs are used for OPENAPI programs:

- L8 mode TCBs are used for CICS key OPENAPI application programs, including some CICS programs that run on L8 TCBs for processing web services requests, parsing XML, and accessing z/OS UNIX files for CICS web support.
- L9 mode TCBs are used for user key OPENAPI application programs.

L8 mode TCBs are also used when programs need access to a resource manager through a TRUE enabled by using the OPENAPI option on the ENABLE PROGRAM command. An open API TRUE is given control under an L8 mode TCB, and can use non- CICS APIs without creating subtask TCBs.

CICS automatically controls the number of L8 and L9 TCBs in the pool of L8 and L9 mode open TCBs.

The use of OPENAPI programs can cause more TCB switching than ordinary threadsafe programs. If the OPENAPI program uses any EXEC CICS commands or user exit programs that are not threadsafe, causing a switch to the QR TCB, there is an extra switch, because CICS switches back to the open TCB to continue running the application logic. Additional TCB switching might be involved because of the requirement for the key of the TCB to be correct for OPENAPI programs. OPENAPI TRUEs always run in CICS key on an L8 TCB, so, for example, if a user key OPENAPI program runs on an L9 TCB but makes a Db2 call, CICS switches to an L8 TCB to call Db2, then returns to the L9 TCB to continue running the program. Because of this switching, CICS Db2 applications are normally defined as (CICSAPI) threadsafe programs, rather than as OPENAPI programs. CICS key CICS Db2 applications can be defined as OPENAPI programs if required.

Threadsafe restrictions for OPENAPI programs

An OPENAPI program, although freed from the constraints imposed by the QR TCB, still has obligations both to the CICS system as a whole and to future users of the L8 or L9 TCB it is using.

An L8 or L9 TCB is dedicated for use by the CICS task to which it is allocated, but once the CICS task has completed, the TCB is returned to the dispatcher-managed pool of such TCBs, provided it is still in a "clean " state. (An unclean TCB in this context means that the task using the L8 or L9 mode TCB suffered an unhandled abend in an OPENAPI program. It does not mean that the program has broken the threadsafe restrictions, which CICS would not detect.) Note that the TCB is not dedicated for use by a particular OPENAPI program, but is used by all OPENAPI programs and OPENAPI TRUEs invoked by the CICS task to which the L8 mode TCB is allocated. Also, if an application program invoking an OPENAPI program is coded to threadsafe standards, and defined to CICS as threadsafe, it continues to execute on the L8 mode TCB on return from the program.

An OPENAPI program must not treat the executing open TCB environment in such a way that it causes problems for:

- Application program logic that could run on the open TCB
- OPENAPI TRUEs called by the same task
- Future tasks that might use the open TCB
- CICS management code.

At your own risk, if your OPENAPI program decides to use other (non CICS) APIs, you must be aware of the following:

- When invoking CICS services, or when returning to CICS, an OPENAPI program must ensure it restores the MVS programming environment as it was on entry to the program. This includes cross-memory mode, ASC mode, request block (RB) level, linkage stack level, TCB dispatching priority, in addition to cancelling any ESTAEs added.
- At CICS task termination, an OPENAPI program must ensure it leaves the open TCB in a state suitable to be reused by another CICS transaction. In particular, it must ensure that all non-CICS resources acquired specifically on behalf of the terminating task are freed. Such resources might include:
  - Dynamically allocated data sets
  - Open ACBs or DCBs
  - STIMERM requests
  - MVS managed storage
  - ENQ requests
  - Attached subtasks
  - Loaded modules
  - Owned data spaces

  - Added access list entries
  - Name/token pairs
  - Fixed pages
  - Security settings (TCBSENV must be set to zero)
- An OPENAPI program must not use the following MVS system services that will affect overall CICS operation:
  - CHKPT
  - ESPIE
  - QEDIT
  - SPIE
  - STIMER
  - TTIMER
  - XCTL / XCTLX
  - Any TSO/E services.
- An OPENAPI program must not invoke under the L8 or L9 mode TCB a Language Environment program that is using MVS Language Environment services, because L8 and L9 mode TCBs are initialized for Language Environment using CICS services.

Using the FORCEQR system initialization parameter

Running applications with programs defined as threadsafe to use OTE (for example, in CICS Db2 applications) could cause problems if one or more programs is not threadsafe. If this happens, you can force all your applications programs on to the QR TCB using the FORCEQR system initialization parameter.

This could be useful in a production region, where you cannot afford to have applications out of service while you investigate the problem.

The default for this parameter is FORCEQR=NO, which means that CICS honors the CONCURRENCY attribute on your program resource definitions. As a temporary measure, while you investigate and resolve problems connected with threadsafe-defined programs, you can set FORCEQR=YES. Remember to change this back to FORCEQR=NO when you are ready for your programs to resume use of open TCBs under the OTE.

Non-reentrant programs

There is nothing to prevent non-reentrant application programs being executed by CICS. However, such an application program would not provide consistent results in a multi-threading environment.

To use non-reentrant application programs, or tables or control blocks that are modifiable by the execution of associated application programs, specify the RELOAD(YES) option on their resource definition. RELOAD(YES) results in a fresh copy of the program or module being loaded into storage for each request. This option ensures that multithreading tasks that access a non- reentrant program or table each work from their own copy of the program, and are unaffected by changes made to another version of the program by other concurrent tasks running in the CICS region.

For information about RELOAD(YES), see PROGRAM attributes.

CICS loads any program link-edited with the RENT attributes into a CICS read-only dynamic storage area (DSA). CICS uses the RDSA for RMODE(24) programs, and the ERDSA for RMODE(ANY) programs.By default, the storage for these DSAs is allocated from read-only key-0 protected storage, protecting any modules loaded into them from all except programs running in key-zero or supervisor state.(If CICS initializes with the RENTPGM=NOPROTECT system initialization parameter, it does not use read-only key-0 storage, and use CICS-key storage instead.)

If you want to execute a non-reentrant program or module, it must be loaded into a non-read-only DSA. The SDSA and ESDSA are user-key storage areas for non-reentrant user-key programs and modules.

For more information about CICS DSAs, see CICS dynamic storage areas.

Storing data in a transaction

CICS provides a variety of facilities to store data in and between transactions. Each one differs according to how available it leaves data to other programs in a transaction and to other transactions; in the way it is implemented; and in its overhead, recovery, and enqueuing characteristics.

Storage facilities that exist for the lifetime of a transaction include:

- Transaction work area (TWA)
- User storage (by GETMAIN commands issued without the SHARED option)
- COMMAREA
- Program storage

These areas are all main storage facilities whose source is the dynamic storage areas (DSAs), extended dynamic storage areas (EDSAs), or above-the-bar dynamic storage areas (GDSAs). None of them is recoverable, and none can be protected by resource security keys. They differ in accessibility and duration, so each storage facility meets a different set of storage needs.

Transaction work area (TWA)

The transaction work area (TWA) is allocated when a transaction is initiated, and is initialized to binary zeroes. It lasts for the entire duration of the transaction, and is accessible to all local programs in the transaction.

Any remote programs that are linked by a distributed program link command do not have access to the TWA of the client transaction. The size of the TWA is determined by the TWASIZE option on the transaction resource definition. If this size is nonzero, the TWA is always allocated. See TRANSACTION attributes for more information about determining the TWASIZE.

Processor overhead associated with using the TWA is minimal. You do not need a GETMAIN command to access it, and you address it using a single ADDRESS command. The TASKDATAKEY option governs whether the TWA is obtained in CICS-key or user-key storage. (See “Storage control” on page 292 for a full explanation of CICS-key and user-key storage.) The TASKDATALOC option of the transaction definition governs whether the acquired storage can be above the 16MB line or not.

The TWA is suitable for quite small data storage requirements and for larger requirements that are both relatively fixed in size and are used more or less for the duration of the transaction. Because the TWA exists for the entire transaction, a large TWA size has much greater effect for conversational than for nonconversational transactions.

User storage

User storage is available to all the programs in a transaction, but some effort is required to pass it between programs using LINK or XCTL commands. Its size is not fixed, and it can be obtained (using GETMAIN commands) just when the transaction requires it and returned as soon as it is not needed.

User storage is useful for large storage requirements that are variable in size or are shorter-lived than the transaction.

See “Storage control” on page 292 for information about how the USERDATAKEY and CICSDATAKEY options of the GETMAIN commands override the TASKDATAKEY option of the transaction resource definition.

The SHARED option of the GETMAIN commands causes the acquired storage to be retained after the end of the task. The storage can be passed in the communication area from one task to the next at the same terminal. The first task returns the address of the communication area in the COMMAREA option of the RETURN command. The second task accesses the address in the COMMAREA option of the ADDRESS command. You must use the SHARED option of the GETMAIN commands to ensure that your storage is in common storage.

Because of the processor overhead involved in GETMAIN commands, do not use user storage for a small amount of storage. Use the transaction work area (TWA) for the smaller amounts, or group them into a larger request. Although the storage acquired by GETMAIN commands can be held longer when using combined requests, the processor overhead and the reference set size are both reduced.

COMMAREA in LINK and XCTL commands

A communication area (COMMAREA) is a facility used to transfer information between two programs within a transaction, or between two transactions from the same terminal.

For information about using COMMAREA between transactions, see " Using the COMMAREA in RETURN commands" on page 107.

Information in COMMAREA is available only to the two participating programs, unless those programs explicity make the data available to other programs that might be invoked later in the transaction.

- When one program links to another, the COMMAREA can be any data area to which the linking program has access. It is often in the working storage or LINKAGE SECTION of that program. In this area, the linking program can both pass data to the program it is invoking and receive results from that program.
- When a program transfers control (an XCTL command) to another, CICS might copy the specified COMMAREA into a new area of storage, because the invoking program and its control blocks might no longer be available after it transfers control.

In both cases, the address of the area is passed to the program that is receiving control, and the CICS command-level interface sets up addressability. See "Program control" on page 146 for further information.

When XCTL is used, CICS ensures that any COMMAREA is addressable by the program that receives it, by creating the COMMAREA in an area that conforms to the addressing mode of the receiving program. If the receiver is AMODE(24), the COMMAREA is created below the 16 MB line, and if the receiver is AMODE(31), the COMMAREA is created above 16 MB but below 2 GB.

The COMMAREA is copied to USERKEY storage where necessary, depending on the addressing mode and EXECKEY attributes of the receiving program. See "Storage control" on page 292 for more information about EXECKEY.

CICS contains algorithms designed to reduce the number of bytes to be transmitted. The algorithms remove some trailing binary zeros from the COMMAREA before transmission and restore them after transmission. The operation of these algorithms is transparent to the application programs, which always see the full-size COMMAREA.

The overhead for using COMMAREA in an LINK command is minimal; it is slightly more with the XCTL and RETURN commands, when CICS creates the COMMAREA from a larger area of storage used by the program.

Channels in LINK and XCTL commands

Instead of using a communication area (COMMAREA), a more modern method of transferring data between CICS programs is to use a channel.

Channels have several advantages over COMMAREAs - see "Benefits of channels" on page 118 . To pass a channel on a LINK or XCTL command, you use the CHANNEL option in place of the COMMAREA option.

Channels are described in "Transferring data between programs using channels" on page 108.

Program storage

CICS creates a separate copy of the variable area of a CICS program for each transaction using the program. This area is known as *program storage*.

This area is called the WORKING-STORAGE SECTION in COBOL, automatic storage in C, C++, and PL/I, and the DFHEISTG section in assembler language. Like the TWA, this area is of fixed size and is allocated by CICS without you having to issue a GETMAIN command. The EXEC CICS interface sets up

addressability automatically. Unlike the TWA, however, this storage lasts only while the program is being run, not for the duration of the transaction. This makes it useful for data areas that are not required outside the program and that are either small or, if large, are fixed in size and are required for all or most of the execution time of the program.

Temporary storage queues

Temporary storage is the primary CICS facility for storing data that must be available to multiple transactions. Data items in temporary storage are kept in temporary storage queues. The items can be retrieved by the originating task, or by any other task, by using the symbolic name assigned to the temporary storage queue.

A temporary storage queue containing multiple items can be thought of as a small data set. Specific items (logical records) in a queue are referred to by relative position numbers. The items can be addressed either sequentially or directly, by item number. If a queue contains only a single item, it can be thought of as a named scratchpad area.

Temporary storage queues are identified by symbolic names of up to 16 characters. To avoid conflicts caused by duplicate names, establish a naming convention. For example, the operator identifier or terminal identifier could be used as a suffix to each programmer-supplied symbolic name. The fact that temporary storage queues can be named as they are created provides a powerful form of direct access to saved data. You can access scratchpad areas for resources such as terminals and data set records by including the terminal name or record key in the queue name.

Compared with other methods to pass data from task to task, temporary storage queues can require more processor use. You use an EXEC CICS command for every interaction with temporary storage queues, and CICS must find or insert the data by using its internal index. The processor use with main temporary storage is therefore greater than with the CWA or TCTUA. With auxiliary storage, there is typically data set I/O as well. Shared temporary storage pools require temporary storage servers, and applications must access the coupling facility to retrieve the data.

However, temporary storage queues have a number of advantages over other methods to pass data. You do not need to allocate temporary storage until it is required. You keep it only as long as it is required, and the item size is not fixed until you issue the command that creates it. Temporary storage queues are therefore a good choice for relatively high-volume data and data that varies in length or duration. Resource protection is also available for temporary storage queues.

Temporary storage queues remain in storage until they are deleted by the originating task, by any other task, or on an initial or cold start. Your application can use the **DELETEQ TS** command to delete temporary storage queues at the end of their useful life. If your application cannot always delete temporary storage queues, consider setting up automatic deletion. You can make CICS automatically delete non-recoverable temporary storage queues in main storage or auxiliary storage if they have not been accessed recently. The expiry interval in your TSMODEL resource definitions controls automatic deletion.

Selecting a location for temporary storage queues

Temporary storage for a CICS region can be in main storage, auxiliary storage, or shared temporary storage pools in a z/OS coupling facility. For an overview of the different temporary storage locations and the features available for temporary storage queues in each location, see CICS temporary storage: overview in Improving performance.

An application uses the **WRITEQ TS** command to write the first item of data to a temporary storage queue. The command specifies the symbolic name of the temporary storage queue. If this name matches an installed TSMODEL resource definition, CICS creates the temporary storage queue in the temporary storage location specified by the TSMODEL resource definition. If there is no matching TSMODEL resource definition, CICS uses the temporary storage location that the application specifies on the **WRITEQ TS** command. The default location is auxiliary temporary storage.

To choose where your temporary storage queues are located, consider these factors:

Lifetime and frequency of use

- Main temporary storage is more appropriate for temporary storage queues that are needed for short periods of time, or that are accessed frequently.
- Auxiliary temporary storage and shared temporary storage pools are more appropriate for temporary storage queues that have a relatively long lifetime, or are accessed infrequently. You could use a cutoff point of a 1 second lifetime to decide whether queues go in main storage, or auxiliary or coupling facility storage.

Recovery

- Temporary storage queues in main storage cannot be defined as recoverable.
- Temporary storage queues in auxiliary storage can be defined as recoverable.
- CICS recovery is not available for temporary storage queues in shared temporary storage pools. However, the coupling facility is not affected by a CICS restart, so temporary storage queues in shared temporary storage pools can be considered persistent.

Automatic deletion

You can specify that CICS deletes eligible temporary storage queues automatically when they are no longer required, by adding an expiry interval to the corresponding temporary storage models.

- You can set an expiry interval for temporary storage queues in main storage.
- You can set an expiry interval for nonrecoverable queues in auxiliary temporary storage. Recoverable queues cannot be deleted automatically.
- Temporary storage queues in shared temporary storage pools can be deleted automatically.

Storage type

Main temporary storage is in 64-bit storage in the CICS region. If you do not require recoverable temporary storage, you can specify that an application uses main temporary storage. As a result, there is less pressure on space in 31-bit storage, and reduced I/O activity to write data to disk.

Locking and waits for temporary storage queues

The CICS temporary storage domain can process multiple requests concurrently, but it serializes requests made for the same temporary storage queue. The queue is locked for the duration of each request.

Only one transaction at a time can write to or delete a recoverable temporary storage queue. If you choose to make queues recoverable, bear in mind the probability of enqueues.

If a task tries to write to temporary storage and there is no space available, CICS normally suspends it. The task can regain control using either a **HANDLE CONDITION NOSPACE** command, or the RESP or NOHANDLE option on the **WRITEQ TS** command. If suspended, the task is not resumed until some other task frees the necessary space in main storage or the VSAM data set. This situation can produce unexplained response delays, especially if the waiting task owns exclusive-use resources, so all other tasks needing those resources must also wait.

Intrapartition transient data

Intrapartition transient data consists of queues of data, kept together in a single data set, with an index that CICS maintains in main storage.

Intrapartition transient data has some characteristics in common with auxiliary temporary storage. (See “Efficient sequential data set access” on page 102 for information about ***extrapartition*** transient data.)

You can use transient data for many of the purposes for which you would use auxiliary temporary storage, but there are some important differences.

- Transient data does not have the same dynamic characteristics as temporary storage. Unlike temporary storage queues, transient data queues cannot be created at the time data is written by an application program. However, transient data queues can be defined and installed using RDO while CICS is running.

- Transient data queues must be read sequentially. Each item can be read only once. After a transaction reads an item, that item is removed from the queue and is not available to any other transaction. In contrast, items in temporary storage queues may be read either sequentially or directly (by item number). They can be read any number of times and are not removed from the queue until the entire queue is purged.

  These two characteristics make transient data inappropriate for scratch-pad data but suitable for queued data such as audit trails and output to be printed. In fact, for data that is read sequentially once, transient data is preferable to temporary storage.
- Items in a temporary storage queue can be changed; items in transient data queues cannot.
- Transient data queues are always written to a data set. (There is no form of transient data that corresponds to main temporary storage.)
- You can define transient data queues so that writing items to the queue causes a specific transaction to be initiated (for example, to process the queue). Temporary storage has nothing that corresponds to this "trigger" mechanism, although you may be able to use a START command to perform a similar function.
- Transient data has more recovery options than temporary storage. Transient data queues can be physically or logically recoverable.
- Because the commands for intrapartition and extrapartition transient data are identical, you can switch between the two types of data set. To do this, change only the transient data queue definitions and not your application programs themselves. Temporary storage has no corresponding function of this kind.

GETMAIN SHARED command

Storage acquired using the SHARED option of the GETMAIN or GETMAIN64 command is not released when the acquiring task ends. This enables one task to leave data in storage for use by another task.

The storage is not released until a FREEMAIN or FREEMAIN64 command is issued, either by the acquiring task or by another task.

Your own data sets

You can also use your own data sets to save data between transactions. This method probably has the largest overhead in terms of instructions processed, buffers, control blocks, and user programming requirements, but does provide extra functions and flexibility.

Not only can you define data sets as recoverable resources, but you can log changes to them for forward recovery. You can specify the number of strings for the data set, as well as on the temporary storage and transient data sets, to ensure against access contention, and you can use resource security.

Lengths of areas passed to CICS commands

When a CICS command includes a LENGTH option, it typically accepts the length as a signed halfword binary value. The use of a signed halfword binary value places a theoretical upper limit of 32 KB on the length. In practice, the limits are lower and vary for each command.

The limits depend on data set definitions, recoverability requirements, buffer sizes, and local networking characteristics.

LENGTH options

In COBOL, C, C++, PL/I, and assembler language, the translator deals with lengths.

See Application development reference for programming information, including details of when you need to specify the LENGTH option. Where possible, do not let the length specified in CICS command options exceed 24 KB. For more information, see LENGTH options in CICS commands.

Many commands involve the transfer of data between the application program and CICS. In all cases, the length of the data to be transferred must be provided by the application program.

When you use an **EXEC CICS LINK** command to pass data in a COMMAREA, ensure that you specify a LENGTH value that matches the length of the data being passed in the COMMAREA. Do not specify 0 (zero) for LENGTH, because the resulting behavior is unpredictable and the **EXEC CICS LINK** command might fail. When you use a COMMAREA to pass data, the program that is linked to must verify that the EIBCALEN field in the EIB of the task matches what the program expects. Discrepancies might result in storage violations or system failures. For more information, see "Passing data to other programs by using COMMAREA" on page 149.

In most cases, the LENGTH option must be specified if the SET option is used; the syntax of each command and its associated options show whether this rule applies.

There are options on the **WAIT EXTERNAL** command and a number of **QUERY SECURITY** commands that give the resource status or definition. CICS supplies the values associated with these options, hence the name, CICS-value data areas. The options are shown in the syntax of the commands with the term *cvda* in parentheses. For programming information about CVDAs, see CICS-value data areas (CVDAs).

For journal commands, the restrictions apply to the sum of the LENGTH and PFXLENG values. See "Journaling" on page 186.

Journal records

For journal records, the journal buffer size can impose a limit lower than 64 KB. The limit applies to the sum of the LENGTH and PFXLENG values.

Data set definitions

For temporary storage, transient data, and file control, the data set definitions can impose limits lower than 24 KB.

For information about creating data sets, see Defining data sets. For information about resource definition for files, see FILE resources.

Recommendation

For any command in any system, 32,000 bytes is a good working limit for LENGTH specifications. Subject to user-specified record and buffer sizes, this limit is unlikely to cause an error or to place a constraint on applications.

Note: The value in the LENGTH option must never exceed the length of the data area addressed by the command.

Minimizing errors

Use these techniques to help you make your applications error-free. Some of these suggestions apply not only to programming, but also to operations and systems.

What often happens is that, when two application systems that run perfectly by themselves are run together, performance goes down and you begin experiencing "lockouts" or waits. The scope of each system has not been defined enough.

The key points in a well-designed application system are:

- At all levels, each function is defined clearly with inputs and outputs well-stated
- Resources that the system uses are adequately defined
- Interactions with other systems are known

Protecting CICS from application errors

- You can use the storage protection facility to prevent CICS code and control blocks from being overwritten by your application programs. You can choose whether to use this facility by using CICS system initialization parameters. For more information about this facility, see Storage protection.

- Consider using standards that avoid problems that can be caused by techniques such as the use of GETMAIN commands.

General rules for testing applications

- Do not test on a production CICS system. Use a test system, where you can isolate errors without affecting live databases.
- Use someone other than the application developer to do the testing, if possible.
- Document the data you use for testing.
- Test your applications several times. See “Testing applications” on page 718.
- Use the CEDF transaction for initial testing. See “Execution diagnostic facility (EDF)” on page 629.
- Use stress or volume testing to find problems that are not found in a single-user environment. A good tool for this is Teleprocessing Network Simulator (TPNS, licensed program number 5740-XT4).

  TPNS is a telecommunications testing package that enables you to test and evaluate application programs before you install them. You can use TPNS for testing logic, user exit routines, message logging, data encryption, and device-dependencies, if these are used in application programs in your organization. It is useful in investigating system performance and response times, stress testing, and evaluating TP network design.
- Test whether the application can handle correct data and incorrect data.
- Test against complete copies of the related databases.
- Consider using multiregion operation. See Multiregion operation.
- Before you move an application to the production system, run a final set of tests against a copy of the production database to catch any errors.

In particular, look for destroyed storage chains.

Assembler language programs (if not addressing data areas properly) can be harder to identify because they can alter something that affects (and abends) another transaction.

For more information about solving a problem, see Troubleshooting and support.

Non-terminal transaction security

CICS can protect resources used in non-terminal transactions against unauthorized use.

These transactions are of three types:

- Transactions that are started by a START command and that do not specify a terminal ID.
- Transactions that are started, without a terminal, as a result of the trigger level being reached for an intrapartition transient data queue.
- The CICS internal transaction (CPLT), which runs during CICS startup, to execute programs specified in the program list table (PLT). This transaction executes both first and second phases of PLTs.

Also, resource security checking can now be carried out for PLT programs that are run during CICS shutdown. PLT shutdown programs execute as part of the transaction that requests the shutdown, and therefore run under the authorization of the user issuing the shutdown command.

The START command handles security for non-terminal transactions started by the START command.

A surrogate user who is authorized to attach a transaction for another user, or cause it to be attached, or who inherits all the resource access authorizations for that transaction, can act for the user.

CICS can issue up to three surrogate user security checks on a single START command, depending on the circumstances:

1. The userid of the transaction that issues the START command, if USERID is specified
2. The userid of the CEDF transaction, if the transaction that issues the START command is being run in CEDF dual-screen mode

3. The CICS region userid of the remote system, if the START command is function shipped to another CICS system and link security is in effect.

A separate surrogate user security check is done for each of these userids, as required, before the transaction is attached.

For programming information about the USERID option, USERIDERR condition, and INVREQ, and NOTAUTH conditions, see CICS command summary.

Design for performance

You can change the design of your application program to improve performance and efficiency.

- "Program size" on page 96
- "Virtual storage" on page 97
- "Exclusive control of resources" on page 100
- "Operational control" on page 100
- "Operating system waits" on page 101
- "The NOSUSPEND option" on page 101
- "Efficient sequential data set access" on page 102
- "Efficient logging" on page 103

Other aspects of application design are addressed in Chapter 4, "Designing applications," on page 73

If you have a performance problem that applies in a particular situation, try to isolate the changes you make so that their effects apply only in that situation. After fixing the problem and testing the changes, use them in your most commonly-used programs and transactions, where the effects on performance are most noticeable.

Program size

The early emphasis on small programs led CICS programmers to break up programs into units that were as small as possible, and to transfer control using the XCTL command, or link using the LINK command, between them.

In current systems, however, it is not always better to break up programs into such small units, because there is CICS processing overhead for each transfer and, for LINK commands, there is also storage overhead for the register save areas (RSAs).

For modestly-sized blocks of code that are processed sequentially, inline code is most efficient. The exceptions to this rule are blocks of code that are:

- Fairly long and used independently at several different points in the application
- Subject to frequent change (in which case, you balance the overhead of LINK or XCTL commands with ease of maintenance)
- Infrequently used, such as error recovery logic and code to handle uncommon data combinations

If you have a block of code that for one of these reasons, has to be written as a subroutine, the best way of dealing with this from a performance viewpoint is to use a closed subroutine within the invoking program (for example, code that is dealt with by a PERFORM command in COBOL). If it is needed by other programs, it should be a separate program. A separate program can be called, with a CALL statement (macro), or it can be kept separate and processed using an XCTL or a LINK command. Execution overhead is least for a CALL, because no CICS services are invoked; for example, the working storage of the program being called is ***not*** copied. A called program, however, must be linked into the calling one and so cannot be shared by other programs that need it unless you use special COBOL, C, or PL/I facilities. A called subroutine is loaded as part of each program that CALLs it and hence uses more storage. Thus, subsequent transactions using the program may or may not have the changes in the working storage made to the called program. This depends entirely on whether CICS has loaded a new copy of the program into storage.

Overhead (but also flexibility) is highest with the XCTL and LINK commands. Both processor and storage requirements are much greater for a LINK command than for an XCTL command. Therefore, if the invoking program does not need to have control returned to it after the invoked program is processed, it should use an XCTL command.

The load module resulting from any application program can occupy up to two gigabytes of main storage. Clearly, there is an extra cost associated with loading and initializing very large load modules, and CICS dynamic storage limits (EDSA) would need to be set correspondingly high. You should, if possible, avoid the use of large load modules. However large applications written in an object-oriented language, such as C++, can easily exceed 16M in size. Experience with C++ classes bound into a single DLL is that performance of the classes is degraded if the single DLL is reorganized into two or more DLLs. This is due to the processing required to resolve function references between multiple DLLs.

You may get an abend code of APCG if your program occupies all the available storage in the dynamic storage area (DSA).

Virtual storage

By careful design, you can minimize the amount of virtual storage used, and reduce your application's overhead.

A truly conversational CICS task is one that converses with the terminal user for several or many interactions, by issuing a terminal read request after each write (for example, using either a SEND command followed by a RECEIVE command, or a CONVERSE command). This means that the task spends most of its extended life waiting for the next input from the terminal user.

Any CICS task requires some virtual storage throughout its life and, in a conversational task, some of this virtual storage is carried over the periods when the task is waiting for terminal I/O. The storage areas involved include the TCA and associated task control blocks (including EIS or EIB) and the storage required for all programs that are in use when any terminal read request is issued. Also included are the work areas (such as copies of COBOL working storage) associated with this task's use of those programs.

With careful design, you can sometimes arrange for only one very small program to be retained during the period of the conversation. The storage needed could be shared by other users. You must multiply the rest of the virtual storage requirement by the number of concurrent conversational sessions using that code.

By contrast, a pseudoconversational sequence of tasks requires almost all of its virtual storage only for the period spent processing message pairs. Typically, this takes a period of 1-3 seconds in each minute (the rest being time waiting for operator input). The overall requirement for multiple concurrent users is thus perhaps five percent of that needed for conversational tasks. However, you should allow for data areas that are passed from each task to the next. This may be a COMMAREA of a few bytes or a large area of temporary storage. If it is the latter, you are normally recommended to use temporary storage on disk rather than in main storage, but that means adding extra temporary storage I/O overhead in a pseudoconversational setup, which you do not need with conversational processing.

The extra virtual storage you need for conversational applications usually means that you need a correspondingly greater amount of real storage. The paging you need to control storage involves additional overhead and virtual storage. The adverse effects of paging increase as transaction rates go up, and so you should minimize its use as much as possible. See “Reducing paging effects” on page 97 for information about doing so.

Reducing paging effects

Reducing paging effects is a technique used by CICS in a virtual-storage environment. The key objective of programming in this environment is the reduction of page faults. A page fault occurs when a program refers to instructions or data that do not reside in real storage, in which case the page in virtual storage that contains the instructions or data referred to must be paged into real storage. The more paging required, the lower the overall system performance.

About this task

Although an application program may be able to communicate directly with the operating system, the results of such action are unpredictable and can degrade performance.

An understanding of the following terms is necessary for writing application programs to be run in a virtual-storage environment:

Locality of reference
The consistent reference, during the execution of the application program, to instructions and data within a relatively small number of pages (compared to the total number of pages in a program) for relatively long periods.

Working set
The number and combination of pages of a program needed during a given period.

Reference set
Direct reference to the required pages, without intermediate storage references that retrieve useless data.

Locality of reference

Keep the instructions processed and data used in a program within a relatively small number of pages (4096 byte segments). This quality in a program is known as *locality of reference*.

About this task

You can do this by:

- Making the execution of the program as linear as possible.
- Keeping any subroutines that you use in the normal execution sequence as close as possible to the code that invokes them.
- Placing code inline, even if you have to repeat it, if you have a short subroutine that is called from only a few places.
- Separating error handling and other infrequently processed code from the main flow of the program.
- Separating data used by such code from data used in normal execution.
- Defining data items (especially arrays and other large structures) in the order in which they are referred to.
- Defining the elements in a data structure in the approximate order in which they are referred to. For example, in PL/I, all the elements of one row are stored, then the next row, and so on. Define an array so that you can process it by row rather than by column.
- Initializing data as close as possible to where it is first used.
- Avoiding COBOL variable MOVE operations, because these expand into subroutine calls.
- Issuing as few GETMAIN commands as possible. It is better for the program to add up its requirements and do one GETMAIN command than to do several smaller ones, unless the durations of these requirements vary greatly.
- Avoiding use of the INITIMG option on a GETMAIN command, if possible. It causes an immediate page reference to the storage that is obtained, which might otherwise not occur until much later in the program, when there are other references to the same area.

Note: Some of the previous suggestions can conflict with your installation's programming standards if these are aimed at the readability and maintainability of the code, rather than speed of execution in a virtual storage environment. Some structured programming methods, in particular modular programming techniques, make extensive use of the PERFORM command in COBOL (and the equivalent programming techniques in C, PL/I, and assembler language) to make the structure of the program clear. This can also result in more exceptions to sequential processing than are found in a nonstructured program. Nevertheless, the much greater productivity associated with structured code can be worth the possible loss of locality of reference.

Working set

The working set is the number and combination of pages of a program needed during a given period.

About this task

To minimize the size of the working set, the amount of storage that a program refers to in a given period should be as small as possible. You can do this by:

- Writing modular programs and structuring the modules according to frequency and anticipated time of reference. Do not modularize merely for the sake of size; consider duplicate code inline as opposed to subroutines or separate modules.
- Using separate subprograms whenever the flow of the program suggests that execution is not be sequential.
- Not tying up main storage awaiting a reply from a terminal user.
- Using command-level file control locate-mode input/output rather than move-mode.
- In COBOL programs, specifying constants as literals in the PROCEDURE DIVISION, rather than as data variables in the WORKING STORAGE section.
- In C, C++, and PL/I programs, using static storage for constant data.
- Avoiding the use of LINK commands where possible, because they generate requests for main storage.

Reference set

Try to keep the overall number of pages that a program uses during normal operation as small as possible. These pages are termed the reference set, and they give an indication of the real storage requirement of the program.

About this task

The reference set can be reduced by:

- Specifying constants in COBOL programs as literals in the PROCEDURE DIVISION, rather than as data variables in the WORKING STORAGE SECTION. The reason for this is that there is a separate copy of working storage for every task executing the program, whereas literals are considered part of the program itself, of which only one copy is used in CICS.
- Using static storage in C, C++, and PL/I for data that is genuinely constant, for the same reason as in the previous point.
- Reusing data areas in the program as much as possible. You can do this with the REDEFINES clause in COBOL, the union clause in C and C++, based storage in PL/I, and ORG or equivalents in assembler language. In particular, if you have a map set that uses only one map at a time, code the DFHMSD map set definition without specifying either the STORAGE=AUTO or the BASE operand. This allows the maps in the map set to redefine one another.

Refer to data directly by:

- Avoiding long searches for data in tables
- Using data structures that can be addressed directly, such as arrays, rather than structures that must be searched, such as chains
- Avoiding methods that simulate indirect addressing

Do not attempt to use overlays (paging techniques) in an application program. System paging is provided automatically and has superior performance. The design of an application program for a virtual storage environment is like that for a real environment. The system should have all modules resident so that code on pages that are not referred to do not need to be paged in.

If the program is dynamic, the entire program must be loaded across adjacent pages before execution begins. Dynamic programs can be purged from storage if they are not being used and an unsatisfied storage request exists. Allowing sufficient dynamic area to prevent purging is more expensive than

making them resident, because a dynamic program does not share unused space on a page with another program.

Exclusive control of resources

The fundamental and powerful recovery facilities that CICS provides have performance implications. You can adopt various approaches to reduce contention delays for resources.

CICS serializes updates to recoverable resources so that if a transaction fails, its changes to those resources can be backed out independently of those made by any other transaction. Consequently, a transaction updating a recoverable resource gets control of that resource until it terminates or indicates that it wants to commit those changes with a SYNCPOINT command. Other transactions requiring the same resource must wait until the first transaction finishes with it.

The primary resources that produce these locking delays are data sets, DL/I databases, temporary storage, and transient data queues. The unit where protection is based is the individual record (key) for data sets, the program specification block (PSB) for DL/I databases, and the queue name for temporary storage. For transient data, the "read" end of the queue is considered a separate resource from the "write" end (that is, one transaction can read from a queue while another is writing to it).

To reduce transaction delays from contention for resource ownership, the length of time between the claiming of the resource and its release (the end of the UOW) should be minimized. In particular, conversational transactions should not own a critical resource across a terminal read.

Note: Even for unrecoverable data sets, VSAM prevents two transactions from reading the same record for update at the same time. This enqueue ends as soon as the update is complete, however, rather than at the end of the UOW. Even this protection for a BDAM data set, can be relinquished by defining them with "no exclusive control" (SERVREQ=NOEXCTL) in the file control table.

This protection scheme can also produce deadlocks as well as delays, unless specific conventions are observed. If two transactions update more than one recoverable resource, they should always update the resources in the same order. If they each update two data sets, for example, data set "A" should be updated before data set "B" in all transactions. Similarly, if transactions update several records in a single data set, they should always do so in some predictable order (low key to high, or conversely). You might also consider including the TOKEN keyword with each READ UPDATE command. See “The TOKEN option” on page 250 for information about the TOKEN keyword. Transient data, temporary storage, and user journals must be included among such resources. “Locking (enqueuing on) resources in application programs” on page 566 contains further information about the extent of resource protection.

It might be appropriate here to note the difference between CICS data sets on a VSAM control interval, and VSAM internal locks on the data set. Because CICS has no information about VSAM enqueue, a SHARE OPTION 4 control interval that is updated simultaneously from batch and CICS can result in, at best, reduced performance and, at worst, an undetectable deadlock situation between batch and CICS. You should avoid such simultaneous updates between batch and CICS . In any case, if a data set is updated by both batch and CICS , CICS is unable to ensure data integrity.

Operational control

You can use a number of operational techniques to influence the performance and efficiency of the CICS system.

MXT

The CICS system initialization parameter MXT specifies the maximum number of user tasks that can exist in a CICS system at the same time. MXT is invaluable for avoiding short-on-storage (SOS) conditions and for controlling contention for resources in CICS systems. It works by delaying the creation of user tasks to process input messages, if there are already too many activities in the CICS system. In particular, the virtual storage occupied by a message awaiting processing is usually much less than that needed for the task to process it, so you save virtual storage by delaying the processing of the message until you can do so quickly.

Transaction classes are useful in limiting the number of tasks of a particular user-defined type, or class, if these are heavy resource users.

Runaway tasks

CICS only resets a task's runaway time (ICVR) when a task is suspended. An EXEC CICS command cannot be guaranteed to cause a task to suspend during processing because of the unique nature of each CICS implementation. The runaway time may be exceeded causing a task to abend AICA. This abend can be prevented by coding an EXEC CICS SUSPEND command in the application This causes the dispatcher to suspend the task that issued the request and allow any task of higher priority to run. If there is no task ready to run, the program that issued the suspend is resumed. For further information about abend AICA, see Investigating loops that are not detected by CICS .

Auxiliary trace

Use auxiliary trace to review your application programs. For example, it can show up any obviously unnecessary code, such as a data set browse from the beginning of a data set instead of after a SETL, too many or too large GETMAIN commands, failure to release storage when it is no longer needed, unintentional logic loops, and failure to unlock records held for exclusive control that are no longer needed.

Operating system waits

You should avoid using facilities that cause operating system waits. All CICS activity stops when one of these waits occurs, and all transactions suffer response delays.

The chief sources of operating system waits are:

- Extrapartition transient data sets. (See “Efficient sequential data set access” on page 102 .)
- Those COBOL, C, C++, and PL/I language facilities that you should not use in CICS programs and for which CICS generally provides alternative facilities. For guidance information about the language restrictions, see Chapter 8, “Developing COBOL applications,” on page 513 , Chapter 7, “Developing C and C++ applications,” on page 467 , and Chapter 10, “Developing PL/I applications,” on page 547.
- SVCs and assembler language macros that invoke operating system services, such as write-to-operator (WTO).

The NOSUSPEND option

The default action for certain commands is to suspend the application until the required resource becomes available. You can use the NOSUSPEND option to inhibit this wait and return to the next instruction in the application program.

The default action for the ENQBUSY, NOJBUFSP, NOSPACE, NOSTG, QBUSY, SESSBUSY, and SYSBUSY conditions is to suspend the execution of the application until the required resource (for example, storage) becomes available, and then resume processing of the command. The following commands can result in these conditions:

- ALLOCATE
- ENQ
- GETMAIN
- WRITE JOURNALNAME
- WRITE JOURNALNUM
- READQ TD
- WRITEQ

You can use the NOSUSPEND option (also known as the NOQUEUE option for the ALLOCATE command) to inhibit this wait and cause an immediate return to the instruction in the application program that follows the command.

CICS maintains a table of conditions referred to by the HANDLE CONDITION and IGNORE CONDITION commands in a COBOL application program.

Restriction: HANDLE CONDITION and IGNORE CONDITION commands are supported only in COBOL, PL/I, and assembler language applications (but not AMODE(64) assembler language applications). They are not supported in all other supported high level languages.

Execution of these commands either updates the existing entry, or causes a new entry to be made if the condition has not yet been the subject of such a command. Each entry indicates one of the following three states:

- A label is currently specified:

```
HANDLE CONDITION condition(label)
```

- The condition is to be ignored:

```
IGNORE CONDITION
```

- No label is currently specified:

```
HANDLE CONDITION
```

When the condition occurs, the following tests are made:

1. If the command has the NOHANDLE or RESP option, control returns to the next instruction in the application program. Otherwise, the condition table is scanned to see what to do.
2. If an entry for the condition exists, this entry determines the action.
3. If no entry exists and the default action for this condition is to suspend execution:
   - If the command has the NOSUSPEND or NOQUEUE option, control returns to the next instruction.
   - If the command does not have one of these options, the task is suspended.
4. If no entry exists and the default action for this condition is to abend, a second search is made looking for the ERROR condition:
   - If found, this entry determines the action.
   - If ERROR is searched for and not found, the task is abended.

Efficient sequential data set access

CICS provides a number of different sequential processing options. Each has different performance characteristics.

Temporary storage and intrapartition transient data queues (already discussed in “Temporary storage queues” on page 91 and in “Intrapartition transient data” on page 92) are the most efficient to use, but they must be created and processed entirely within CICS.

Extrapartition transient data is the CICS way of handling standard sequential (QSAM/BSAM) data sets. It is the least efficient of the three forms of sequential support listed, because CICS has to issue operating system waits to process the data sets, as it does when handling BDAM. Moreover, extrapartition transient data sets are not recoverable. VSAM ESDSs, on the other hand, are recoverable within limitations, and processing is more efficient. The recovery limitation is that records added to an ESDS during an uncompleted UOW cannot be removed physically during the backout process, because of VSAM restrictions. They can, however, be flagged as deleted by a user exit routine.

CICS journals provide another good alternative to extrapartition transient data, although only for output data sets. Journals are managed by the MVS system logger, but flexible processing options permit very efficient processing. Each journal command specifies operation characteristics, for example, synchronous or asynchronous, whereas extrapartition operations are governed entirely by the parameters in the transient data queue definition.

Transactions should journal asynchronously, if possible, to minimize task waits in connection with journaling. However, if integrity considerations require that the journal records be physically written

before end of task, you must use a synchronous write. If there are several journal writes, the transaction should use asynchronous writes for all but the last logical record, so that the logical records for the task are written with a minimum number of physical I/Os and only one wait.

You can use journals for input (in batch) as well as output (online) while CICS is running. The supplied batch utility DFHJUP can be used for access to journal data, for example, by printing or copying. Note that reading a journal in batch involves the following restrictions:

- Access to MVS system logger log stream data is provided through a subsystem interface, LOGR.
- Reading records from a journal is possible offline by means of a batch job only.

Efficient logging

CICS provides options to log some or all types of activity against a data set.

Logging updates enables you to reconstruct data sets from backup copies, if necessary. You might also want to log reads for security reasons. Again, you have to balance the need for data integrity and security against the performance effects of logging. These are the actual operations needed to do the logging and the possible delays caused because of the exclusive control that logging implies.

Designing for asynchronous requests

When designing an application, you must consider data types, passing data between successive transactions, programs, recovery, performance, security, and integrity. As an application programmer, you'll want to know which aspects that you, as the programmer, must handle, and which are handled by CICS processing.

Passing data between programs

When a parent task runs a child, a channel can be specified by the **EXEC CICS RUN TRANSID** command. All containers on the channel are copied and passed to the child (which prevents accidentally updating channel data when the child is processing). If the channel does not exist, one is created.

To ensure that the parent task can receive responses from the child task, always specify the CHANNEL option on the **EXEC CICS RUN TRANSID** command, even if the parent task has no data to pass to the child task.

When the parent task receives a response from the child task using **EXEC CICS FETCH CHILD CHANNEL** , the parent task will get a new channel named by CICS. The parent uses this new channel to receive data back from the child.

Note: An asynchronously started child task is not associated with a terminal and therefore cannot return control using the **EXEC CICS RETURN TRANSID** command.

Securing the parent and child tasks

The child tasks will run under the same user ID as the parent task.

Connecting the parent and children

The child tasks must run locally. They can call remote services, for example, calling web services or doing a distributed program link to another CICS region, but they must be initially local to their parent task. The child can communicate this to its children and linked-to programs. Only the parent can receive back results or child data.

Designing for recovery

Each parent and child is a separate task. There is no unit-of-work coordination by CICS between the children, nor between the parent and child.

Sharing data across transactions

CICS has several facilities for sharing data across transactions. You can use the Common Work Area (CWA), the TCTTE user area (TCTUA), the COMMAREA, the display screen, or channels and containers.

Data stored in the TCTUA and the CWA is available to any transaction in the system. Subject to resource security and storage protection restrictions, any transaction can write to them and any transaction can read them.

The use of some of these facilities might cause inter-transaction affinities. See “Affinity” on page 156 for more information about transaction affinities.

This section describes:

- “Using the common work area (CWA)” on page 104
- “Using the TCTTE user area (TCTUA)” on page 106
- “ Using the COMMAREA in RETURN commands” on page 107
- “Using a channel on RETURN commands” on page 108
- “Using the display screen to share data” on page 108

Using the common work area (CWA)

The common work area (CWA) is a single control block that is allocated at system startup time and exists for the duration of that CICS session. The size is fixed, as specified in the system initialization parameter, WRKAREA.

The CWA has the following characteristics:

- There is almost no overhead in storing or retrieving data from the CWA. Command-level programs must issue one ADDRESS command to get the address of the area but, after that, they can access it directly.
- Data in the CWA is not recovered if a transaction or the system fails.
- It is not subject to resource security.
- CICS does not regulate use of the CWA. All programs in all applications that use the CWA must follow the same rules for shared use. These are usually set down by the system programmers, in cooperation with application developers, and require all programs to use the same "copy" module to describe the layout of the area.

  You must not exceed the length of the CWA, because this causes a storage violation. Furthermore, you must ensure that the data used in one transaction does not overlay data used in another. One way to protect CWA data is to use the storage protection facility that protects the CWA from being written to by user-key applications. See “Protecting the CWA” on page 104 for more information.
- The CWA is especially suitable for small amounts of data, such as status information, that are read or updated frequently by multiple programs in an application.
- The CWA is not suitable for large-volume or short-lived data because it is always allocated.

Protecting the CWA

The CWAKEY system initialization parameter allows you to specify whether the CWA is to be allocated from CICS-key or user-key storage.

If you want to restrict write access to the CWA, you can specify CWAKEY=CICS. This means that CICS allocates the CWA from CICS-key storage, restricting application programs defined with EXECKEY(USER) to read-only access to the CWA. The only programs allowed to write to a CWA allocated from CICS-key storage are those you define with EXECKEY(CICS).

Programs that run in CICS key can also write to CICS storage, ensure that such programs are thoroughly tested to make sure that they do not overwrite CICS storage.

To give preference to protecting CICS rather than the CWA, specify CWAKEY=USER for the CWA, and EXECKEY(USER) for all programs that write to the CWA. This ensures that if a program exceeds the length of the CWA it does not overwrite CICS storage. For more information about storage protection, see “Storage control” on page 292.

Figure 28 on page 105 illustrates a particular use of the CWA where the CWA itself is protected from user-key application programs by CWAKEY=CICS.

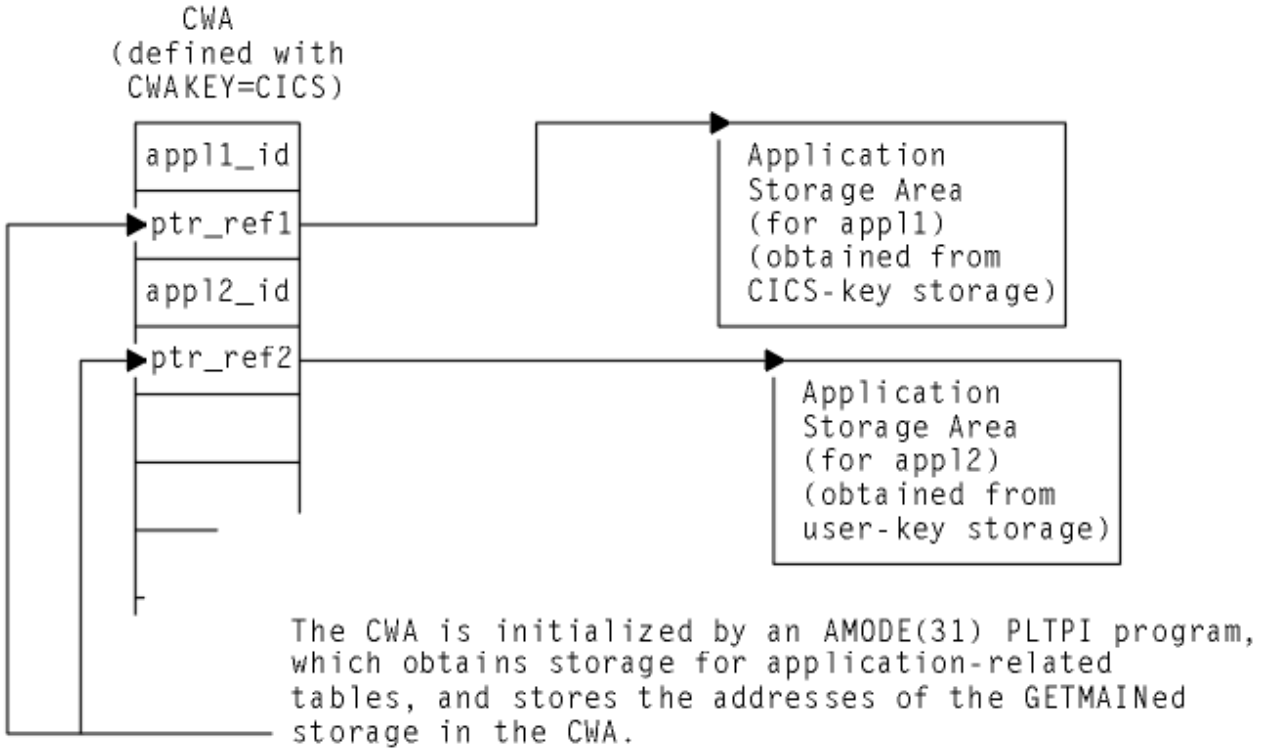


Figure 28. Example of use of CWA in CICS-key storage

In this illustration, the CWA is not used directly to store application data and constants. The CWA contains pairs of application identifiers and associated addresses, with the address fields containing the addresses of data areas that hold the application-related data. For protection, the CWA is defined with CWAKEY=CICS, therefore the program which in this illustration is a program defined in the program list table post initialization (PLTPI) list, and that loads the CWA with addresses and application identifiers must be defined with EXECKEY(CICS). Any application programs requiring access to the CWA should be defined with EXECKEY(USER), ensuring the CWA is protected from overwriting by application programs. In Figure 28 on page 105 , one of the data areas is obtained from CICS-key storage, while the other is obtained from user-key storage.

In the sample code shown in Figure 29 on page 106 , the program list table post-initialization (PLTPI) program is setting up the application data areas, with pointers to the data stored in the CWA.

This example illustrates how to create global data for use by application programs, with addresses of the data stored in the CWA—for example, by a PLTPI program. The first data area is obtained from CICS-key storage, which is the default on a GETMAIN command issued by a PLTPI program, the second from user-key storage by specifying the USERDATAKEY option. The CWA itself is in CICS-key storage, and PLTPROG is defined with EXECKEY(CICS).

```
ID DIVISION.
PROGRAM-ID. PLTPROG.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 APPLID PIC X(8) VALUE SPACES.
77 SYSID PIC X(4) VALUE SPACES.
01 COMM-DATA.
03 AREA-PTR USAGE IS POINTER.
03 AREA-LENGTH PIC S9(8) COMP.
LINKAGE SECTION.
01 COMMON-WORK-AREA.
03 APPL-1-ID PIC X(4).
03 APPL-1-PTR USAGE IS POINTER.
03 APPL-2-ID PIC X(4).
03 APPL-2-PTR USAGE IS POINTER.
PROCEDURE DIVISION.
MAIN-PROCESSING SECTION.
* Obtain APPLID and SYSID values
EXEC CICS ASSIGN APPLID(APPLID)
SYSID(SYSID)
END-EXEC.
* Set up addressability to the CWA
EXEC CICS ADDRESS
CWA(ADDRESS OF COMMON-WORK-AREA)
END-EXEC.
* Get 12KB of CICS-key storage for the first application ('APP1')
MOVE 12288 TO AREA-LENGTH.
EXEC CICS GETMAIN SET(AREA-PTR)
FLENGTH(AREA-LENGTH)
SHARED
END-EXEC.
* Initialize CWA fields and link to load program
* for storage area 1.
MOVE 'APP1' TO APPL-1-ID.
SET APPL-1-PTR TO AREA-PTR.
EXEC CICS LINK PROGRAM('LOADTAB1')
COMMAREA(COMM-DATA)
END-EXEC.
```

```
* Get 2KB of user-key storage for the second application
('APP2')
MOVE 2048 TO AREA-LENGTH.
EXEC CICS GETMAIN SET(AREA-PTR)
FLENGTH(AREA-LENGTH)
SHARED
USERDATAKEY
END-EXEC.
* Initialize CWA fields and link to load program
* for storage area 2.
MOVE 'APP2' TO APPL-2-ID.
SET APPL-2-PTR TO AREA-PTR.
EXEC CICS LINK PROGRAM('LOADTAB2')
COMMAREA(COMM-DATA)
END-EXEC.
EXEC CICS RETURN
END-EXEC.
MAIN-PROCESSING-EXIT.
GOBACK.
```

Figure 29. Sample code for loading the CWA

Using the TCTTE user area (TCTUA)

The TCT user area (TCTUA) is an optional extension to the terminal control table entry (TCTTE). Each entry in the TCT specifies whether this extension is present and, if so, how long it is (by means of the USERAREALEN attribute of the TYPETERM resource definition used for the terminal).

See Autoinstalling model terminal definitions for more information about the TYPETERM resource definition.

The system initialization parameters TCTUALOC and TCTUAKEY specify the location and storage key for all TCTUAs.

- TCTUALOC=BELOW or ANY specifies whether you want 24- or 31-bit addressability to the TCTUAs, and whether TCTCUAs must be stored below the 16MB line or may be either above or below the line.
- TCTUAKEY=USER or CICS specifies whether you want the TCTUAs allocated from user-key or CICS-key storage.

TCTUAs have the following characteristics in common with the CWA:

- Minimal processor overhead (only one ADDRESS command needed)
- No recovery
- No resource security
- No regulation of use by CICS
- Fixed length
- Unsuitability for large-volume or short-lived data

Unlike the CWA, however, the TCTUA for a particular terminal is usually shared only among transactions using that terminal. It is therefore useful for storing small amounts of data of fairly standard length between a series of transactions in a pseudoconversational sequence. Another difference is that it is not necessarily permanently allocated, because the TCTUA only exists while the TCTTE is set up. For non-autoinstall terminals the TCTUA is allocated from system startup; for autoinstall terminals the TCTUA is allocated when the TCTTE is generated.

Using the TCTUA in this way does not require special discipline among using transactions, because data is always read by the transaction following the one that wrote it. However, if you use TCTUAs to store longer-term data (for example, terminal or operator information needed by an entire application), they require the same care as the CWA to ensure that data used in one transaction does not overlay data used in another. You should not exceed the length of the allocated TCTUA, because this produces a storage violation.

Using the COMMAREA in RETURN commands

The COMMAREA option of the RETURN command is designed specifically for passing data between successive transactions in a pseudoconversational sequence. It is implemented as a special form of user storage, although the EXEC interface, rather than the application program, issues the GETMAIN and FREEMAIN requests.

The COMMAREA is allocated from the CICS shared subpool in main storage, and is addressed by the TCTTE, between tasks of a pseudoconversational application. The COMMAREA is freed unless it is passed to the next task.

The first program in the next task has automatic addressability to the passed COMMAREA, as if the program had been invoked by either a LINK command or an XCTL command (see “COMMAREA in LINK and XCTL commands” on page 90). You can also use the COMMAREA option of the ADDRESS command to obtain the address of the COMMAREA.

For a COMMAREA passed between successive transactions in a pseudoconversational sequence in a distributed environment, z/OS Communications Server for SNA imposes a limit of 32KB on the size of the total data length. This limit applies to the entire transmitted package, which includes control data added by Communications Server. The amount of control data increases if the transmission uses intermediate links.

To summarize:

- Processor overhead is low (equivalent to using COMMAREA with an XCTL command and approximately equal to using main temporary storage).
- It is not recoverable.
- There is no resource security.
- It is not suitable for large amounts of data (because main storage is used, and it is held until the terminal user responds).

- As with using COMMAREA to transfer data between programs, it is available only to the first program in a transaction, unless that program explicitly passes the data or its address to succeeding programs.

Using a channel on RETURN commands

Instead of using a communication area (COMMAREA), a more modern method of passing data to the next program in a pseudoconversation is to use a channel.

Channels have several advantages over COMMAREAs - see “Benefits of channels” on page 118 . To pass a channel on a RETURN command, you use the CHANNEL option in place of the COMMAREA option.

Channels are described in “Transferring data between programs using channels” on page 108.

Using the display screen to share data

Data can be stored between pseudoconversational transactions from a 3270 display terminal on the display screen itself.

For example, errors made by users in data entry are highlighted (with highlights or messages) by the transaction processing the data. The next transaction identifier is then set to point to itself (so that it processes the corrected entry), and returns to CICS.

The transaction has two ways of using the ***valid*** data. It can save it (for example, in COMMAREA), and pass it on for the next time it is run. In this case, the transaction must merge the changed data on the screen with the data from previous entries. Alternatively, it can save the data on the screen by not turning off the modified data tags of the keyed fields.

Saving the data on the screen is easy to code, but it is not secure. You are ***not*** recommended to save screens that contain large amounts of data as errors can occur because of the additional network traffic needed to resend the unchanged data. (This restriction does not apply to locally attached terminals.)

Secondly, if the user presses the CLEAR key, the screen data is lost, and the transaction must be able to recover from this. This can be avoided by defining the CLEAR key to mean CANCEL or QUIT, if appropriate for the application concerned.

Data other than keyed data can also be stored on the screen. This data can be protected from changes (except those caused by CLEAR) and can be nondisplay, if necessary.

Transferring data between programs using channels

Channels are a method of transferring data between programs, in amounts that far exceed the 32 KB limit that applies to COMMAREAs.

This section contains:

- “Channels: quick start” on page 109
- “Using channels: some typical scenarios” on page 112
- “Creating a channel” on page 119
- “The current channel” on page 120
- “The scope of a channel” on page 124
- “Discovering which containers were passed to a program” on page 129
- “Discovering which containers were returned from a link” on page 129
- “ CICS read-only containers ” on page 133
- “Designing a channel: Best practices” on page 131
- “Constructing and using a channel: an example” on page 132
- “Channels and BTS activities” on page 134
- “Using channels with Java classes” on page 134
- “Dynamic routing with channels” on page 136

- “Data conversion” on page 137
- “Benefits of channels” on page 118
- “Migrating from COMMAREAs to channels” on page 143
- “Deleting channels and containers and freeing their storage” on page 129

Channels: quick start

A brief introduction to channels and containers.

Channels and containers

Containers are named blocks of data, designed for passing information between programs. Programs can pass any number of containers between each other. Containers are grouped in sets that are called *channels* . A channel is analogous to a parameter list.

To create named containers and assign them to a channel, a program uses **EXEC CICS PUT CONTAINER** (*container-name*) **CHANNEL** (*channel-name*) commands. It can then pass the channel and its containers to a second program by using the CHANNEL(*channel-name*) option of the **EXEC CICS LINK** , **XCTL** , **START** , **RUN TRANSID** or **RETURN** commands.

The second program can read containers that are passed to it using the **EXEC CICS GET CONTAINER** (*container-name*) command. This command reads the named container that belongs to the channel that the program was invoked with.

If the second program is called by an **EXEC CICS LINK** command, it can also return containers to the calling program. It can do this by creating new containers, or by reusing existing containers.

Channels and containers are visible only to the program that creates them and the programs they are passed to, or (for **EXEC CICS FETCH CHANNEL** commands) the program which fetches the channel. When these programs end, CICS automatically destroys the containers and their storage. Programs can also issue commands to delete channels and containers before that time.

AMODE (64) programs can use channels and containers to transfer data in 64-bit storage in the same way, by using **EXEC CICS PUT64 CONTAINER** and **EXEC CICS GET64 CONTAINER** commands. These commands are for use only in non- Language Environment (LE) AMODE(64) assembly language application programs, and CICS business transaction services (BTS) containers are not supported.

Channel containers are not recoverable. Pseudoconversational transactions that are started with RETURN TRANSID CHANNEL( ) cannot be restarted. If you must use recoverable containers, use CICS business transaction services (BTS) containers.

Programs running in z/OS but outside of CICS can pass data to a CICS program by using channels and containers on the external CICS interface (EXCI), as an alternative to using a communications area. The following commands are supported: **EXEC CICS DELETE CHANNEL** , **EXEC CICS DELETE CONTAINER** , **EXEC CICS GET CONTAINER** , **EXEC CICS MOVE CONTAINER** , and **EXEC CICS PUT CONTAINER** . EXCI does not support **EXEC CICS PUT64 CONTAINER** and **EXEC CICS GET64 CONTAINER** commands.

Channels and containers used by an EXCI job will be freed when the job ends. However, good programming practice would be for the job to issue an **EXEC CICS DELETE CHANNEL** command to delete a channel and its set of containers when the channel is no longer required. Otherwise, a subsequent use of EXCI within the same job will have access to the channel and its containers.

If you are using **STARTBROWSE** or **GETNEXT** commands, be aware that the order in which containers are returned is undefined and might change. Applications should not rely on the order of returned containers. If you have existing applications that are written as such, see Upgrading applications for advice.

The transaction channel DFHTRANSACTION

Channels normally go out of scope when the link level changes. They might therefore not be available to all the programs in a transaction. If you create a channel with the name DFHTRANSACTION, it does not go out of scope when the link level changes. It is therefore available to all programs in a transaction,

including any exit points that are API enabled. However, the transaction channel cannot be passed to a CICS region at a version earlier than CICS TS 5.2.

Transaction channels can only be used on MRO and IPIC link types.

Transaction channels are not supported for ISC over SNA link types.

DFHTRANSACTION can be used in all API commands that accept a channel name.

You cannot delete a transaction channel, so when you want to clean up use of data, delete containers from the channel rather than deleting the channel.

Transaction channels are shipped on DPL requests. They are not shipped on other function shipped requests so are not available to exits on the target region.

When you have transaction programs that make distributed program link (DPL) calls to remote programs, if you want to use the DFHTRANSACTION channel, the front-end program in the DPL stack should be the program that creates the channel, even if it creates an empty channel.

An EXCI job using channels and containers can use the transaction channel DFHTRANSACTION. The transaction channel with its container data is shipped to CICS in addition to the channel and its set of containers that are named on the **EXEC CICS LINK** command or call level **DPL_REQUEST** . The transaction channel is not shipped with COMMAREAS on an **EXCI EXEC CICS LINK** command or call level **DPL_REQUEST**.

If you want to use a transaction channel in the EXCI client, the channel must be created in the EXCI client, not on the CICS server task. The transaction channel has a lifetime of the EXCI job. It will operate on the CICS server in the same way as if the transaction channel was shipped by a DPL request from another CICS system.

If an EXCI job passes a transaction channel to a CICS region at a version earlier than CICS TS 5.2, the CICS program will abend with abend code AXGA.

Basic examples

A simple example of a program that creates a channel and passes it to second program.

Figure 30 on page 111 shows a COBOL program, `CLIENT1` , that:

1. Uses PUT CONTAINER(*container-name*) CHANNEL(*channel-name*) commands to create a channel called `inqcustrec` and add two containers, `custno` and `branchno` , to it; these contain a customer number and a branch number, respectively.
2. Uses a LINK PROGRAM(*program-name*) CHANNEL(*channel-name*) command to link to program SERVER1 , passing the `inqcustrec` channel.
3. Uses a GET CONTAINER(*container-name*) CHANNEL(*channel-name*) command to retrieve the customer record returned by SERVER1. The customer record is in the `custrec` container of the `inqcustrec` channel.

Note that the same COBOL copybook, `INQINTC` , is used by both the client and server programs. Line 3 and lines 5 through 7 of the copybook represent the `INQUIRY-CHANNEL` and its containers. These lines are not strictly necessary to the working of the programs, because channels and containers are created by being named on, for example, **PUT CONTAINER** commands; they do not have to be defined. However, the inclusion of these lines in the copybook used by both programs makes for easier maintenance; they record the names of the containers used.

Recommendation

For ease of maintenance of a client/server application that uses a channel, create a copybook that records the names of the containers used and defines the data fields that map to the containers. Include the copybook in both the client and the server program.

Note: This example shows two COBOL programs. The same techniques can be used in any of the other languages supported by CICS. However, *for COBOL programs only* , if the server program uses the SET option (instead of INTO) on the **EXEC CICS GET CONTAINER** command, the structure of the storage

pointed to by SET must be defined in the LINKAGE section of the program. This means that you will require two copybooks rather than one. The first, in the WORKING-STORAGE section of the program, names the channel and containers used. The second, in the LINKAGE section, defines the storage structure.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CLIENT1.

WORKING-STORAGE SECTION.

COPY INQINTC
* copybook INQINTC
* Channel name
* 01 INQUIRY-CHANNEL PIC X(16) VALUE 'inqcustrec'.
* Container names
* 01 CUSTOMER-NO PIC X(16) VALUE 'custno'.
* 01 BRANCH-NO PIC X(16) VALUE 'branchno'.
* 01 CUSTOMER-RECORD PIC X(16) VALUE 'custrec'.
* Define the data fields used by the program
* 01 CUSTNO PIC X(8).
* 01 BRANCHNO PIC X(5).
* 01 CREC.
* 02 CUSTNAME PIC X(80).
* 02 CUSTADDR1 PIC X(80).
* 02 CUSTADDR2 PIC X(80).
* 02 CUSTADDR3 PIC X(80).

PROCEDURE DIVISION.
MAIN-PROCESSING SECTION.

*
* INITIALISE CUSTOMER RECORD
*
... CREATE CUSTNO and BRANCHNO
*
* GET CUSTOMER RECORD
*
EXEC CICS PUT CONTAINER(CUSTOMER-NO) CHANNEL(INQUIRY-CHANNEL) FROM(CUSTNO) FLENGTH(LENGTH OF
CUSTNO) END-EXEC
EXEC CICS PUT CONTAINER(BRANCH-NO) CHANNEL(INQUIRY-CHANNEL) FROM(BRANCHNO) FLENGTH(LENGTH OF
BRANCHNO) END-EXEC

EXEC CICS LINK PROGRAM('SERVER1') CHANNEL(INQUIRY-CHANNEL) END-EXEC

EXEC CICS GET CONTAINER(CUSTOMER-RECORD) CHANNEL(INQUIRY-CHANNEL) INTO(CREC) END-EXEC

*
* PROCESS CUSTOMER RECORD
*
... FURTHER PROCESSING USING CUSTNAME and CUSTADDR1 etc...

EXEC CICS RETURN END-EXEC

EXIT.
```

Figure 30. A simple example of a program that creates a channel and passes it to a second program

Figure 31 on page 112 shows the SERVER1 program linked to by CLIENT1. SERVER1 retrieves the data from the custno and branchno containers it has been passed, and uses it to locate the full customer record in its database. It then creates a new container, custrec , on the same channel, and returns the customer record in it.

Note that the programmer hasn't specified the CHANNEL keyword on the GET and PUT commands in SERVER1 : if the channel isn't specified explicitly, the current channel is used—that is, the channel that the program was invoked with.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SERVER1.

WORKING-STORAGE SECTION.

COPY INQINTC
* copybook INQINTC
* Channel name
* 01 INQUIRY-CHANNEL PIC X(16) VALUE 'inqcustrec'.
* Container names
* 01 CUSTOMER-NO PIC X(16) VALUE 'custno'.
* 01 BRANCH-NO PIC X(16) VALUE 'branchno'.
* 01 CUSTOMER-RECORD PIC X(16) VALUE 'custrec'.
* Define the data fields used by the program
* 01 CUSTNO PIC X(8).
* 01 BRANCHNO PIC X(5).
* 01 CREC.
* 02 CUSTNAME PIC X(80).
* 02 CUSTADDR1 PIC X(80).
* 02 CUSTADDR2 PIC X(80).
* 02 CUSTADDR3 PIC X(80).

PROCEDURE DIVISION.
MAIN-PROCESSING SECTION.

EXEC CICS GET CONTAINER(CUSTOMER-NO) INTO(CUSTNO) END-EXEC
EXEC CICS GET CONTAINER(BRANCH-NO) INTO(BRANCHNO) END-EXEC

... USE CUSTNO AND BRANCHNO TO FIND CREC IN A DATABASE

EXEC CICS PUT CONTAINER(CUSTOMER-RECORD) FROM(CREC) FLENGTH(LENGTH OF CREC) END-EXEC

EXEC CICS RETURN END-EXEC

EXIT.
```

Figure 31. A simple example of a linked to program that retrieves data from the channel it has been passed

Using channels: some typical scenarios

Channels and containers provide a powerful way to pass data between programs. These scenarios show some examples of how channels can be used.

One channel, one program

This example shows a stand-alone program with a single channel.

Figure 32 on page 113 shows the simplest scenario—a "stand-alone" program with a single channel with which it can be invoked.

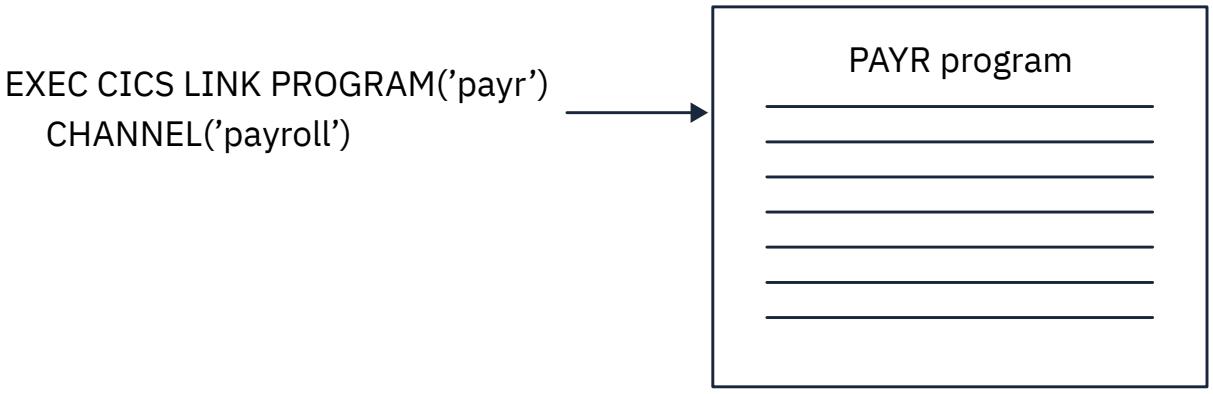


Figure 32. A stand-alone program with a single channel

One channel, several programs (a component)

This example shows a set of related programs (a component) invoked through a single channel.

In Figure 33 on page 115 , there is a single channel to the top-level program in a set of inter-related programs. The set of programs within the shaded area can be regarded as a "component". The client program "sees" only the external channel and has no knowledge of the processing that takes place nor of the existence of the back-end programs.

Inside the component, the programs can pass the channel between themselves. Alternatively, a component program could, for example, pass a subset of the original channel, by creating a new channel and adding one or more containers from the original channel.

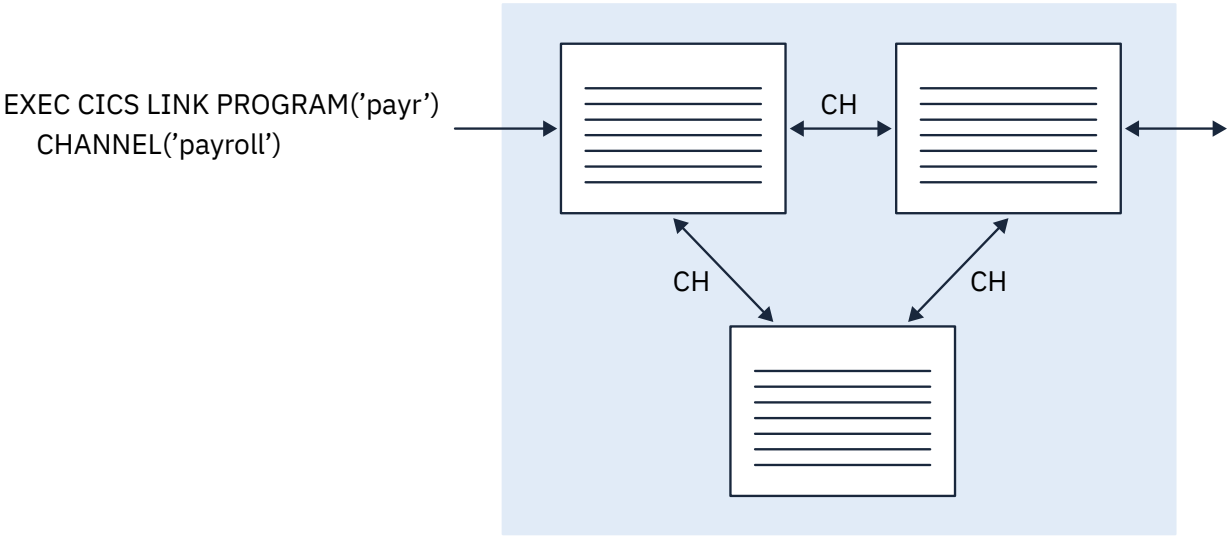


Figure 33. A "component" —a set of related programs invoked through a single external channel

Several channels, one component

This example shows a set of related programs (a component) which can be invoked through two alternative channels.

As in the previous example, we have a set of inter-related programs that can be regarded as a component. However, this time there are two, alternative, external channels with which the component can be invoked. The top-level program in the component issues an EXEC CICS ASSIGN CHANNEL command to determine which channel it has been invoked with, and tailors its processing accordingly.

The "loose coupling" between the client program and the component permits easy evolution. That is, the client and the component can be upgraded at different times. For example, first the component could be upgraded to handle a third channel, consisting of a different set of containers from the first, or second channels. Next, the client program could be upgraded (or a new client written) to pass the third channel.

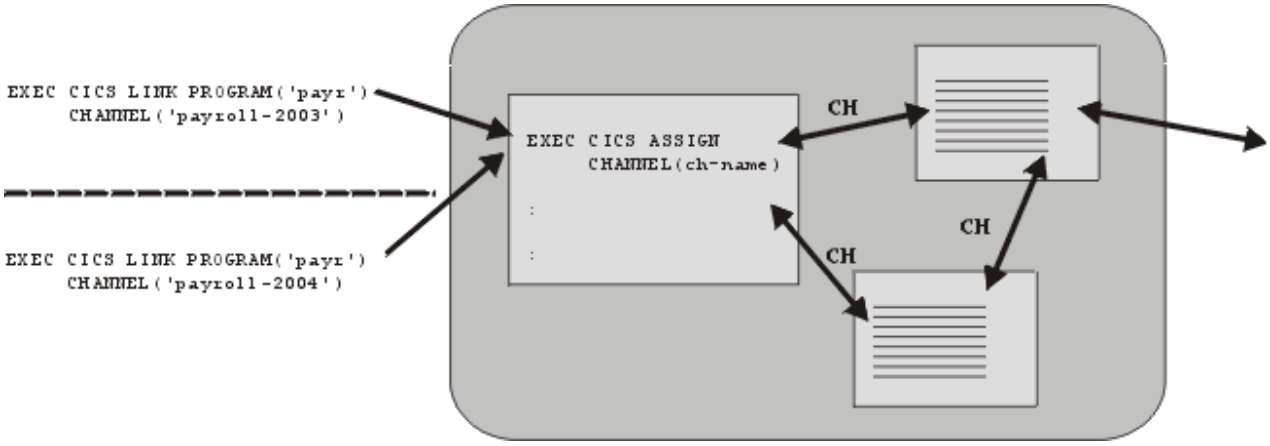


Figure 34. Multiple external channels to the same component

Multiple interactive components

This example shows how multiple components can interact through their channels.

Figure 35 on page 117 shows a "Human resources" component and a "Payroll" component, each with a channel with which it can be invoked. The Payroll component is invoked from both a stand-alone program and the Human resources component.

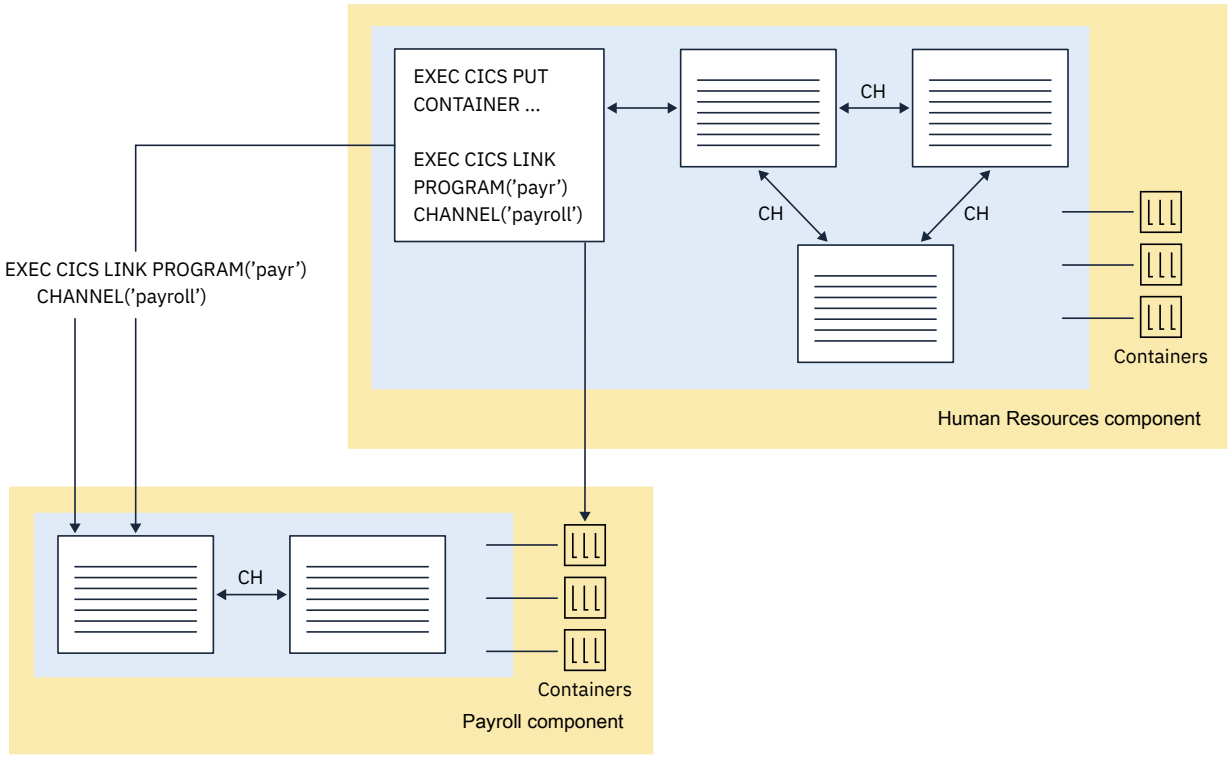


Figure 35. Multiple components which interact through their channels

Benefits of channels

Using the channel and container model in CICS programs to exchange data has several advantages over using communication areas (COMMAREAs).

- There is no limit to the number of containers that can be added to a channel, and the size of individual containers is limited only by the amount of storage that you have available.

  Take care not to create so many large containers that you limit the amount of storage available to other applications.
- A channel can comprise multiple containers, passing data in a structured way.
- Channels do not require the programs that use them to know the exact size of the data returned.
- A channel is a standard mechanism for exchanging data between CICS programs. A channel can be passed on **LINK** , **START** , **XCTL** , **RUN TRANSID** and **RETURN** commands. Distributed program link (DPL) is supported, and the transactions started by **START CHANNEL** and **RETURN TRANSID** commands might be remote.
- The transaction channel, DFHTRANSACTION, remains in scope even when the link level changes, so it can be used to exchange data between any programs in a transaction.
- Channels can be used by CICS application programs written in any of the language supported by CICS. For example, a Java client program on one CICS region can use a channel to exchange data with a COBOL server program on a back-end AOR.
- A server program can be written to handle multiple channels. It can, for example:
  - Discover, dynamically, the channel that it was invoked with.
  - Count and browse the containers in the channel.
  - Vary its processing according to the channel it has been passed.
- You can build "components" from sets of related programs invoked through one or more channels.
- The loose coupling between clients and components permits easy evolution. Clients and components can be upgraded at different times. For example, first a component could be upgraded to handle a new channel, then the client program upgraded (or a new client written) to use the new channel.
- The programmer is relieved of storage management concerns. CICS automatically destroys containers (and their storage) when they go out of scope. Programs may also issue commands to delete individual containers, or a complete channel and all the containers on it, before they go out of scope.
- The data conversion model used by channel applications is simple and is controlled by the application developer, using simple API commands.
- Programmers with experience of CICS business transaction services (BTS) can find it easy to use containers in non-BTS applications.
- Programs that use containers can be called from both channel and BTS applications.
- Non-BTS applications that use containers can be migrated into full BTS applications. (They form a migration route to BTS.)

Channels might not be the best solution in all cases. When designing an application, there are one or two implications of using channels that you must consider:

- When a channel is to be passed to a remote program or transaction, passing a large amount of data might affect performance, particularly if the local and remote regions are connected by an ISC, rather than MRO, connection.
- A channel might use more storage than a COMMAREA designed to pass the same data because container data can be held in more than one place. COMMAREAs are accessed by pointers, whereas the data in containers is copied between programs.

Creating a channel

You can create a channel by naming it on one of a number of API commands. If the channel does not exist in the current program scope, CICS creates it.

About this task

You create a channel by naming it on one of the following commands:

LINK PROGRAM CHANNEL
MOVE CONTAINER CHANNEL TOCHANNEL
PUT CONTAINER CHANNEL
PUT64 CONTAINER
RETURN TRANSID CHANNEL
START TRANSID CHANNEL
XCTL PROGRAM CHANNEL
WEB RECEIVE TOCHANNEL
WEB CONVERSE TOCHANNEL
RUN TRANSID CHANNEL

The most straightforward way to create a channel and populate it with containers of data is to issue a succession of **EXEC CICS PUT CONTAINER(** *container-name* **) CHANNEL(** *channel-name* **) FROM(** *data_area* **)** commands. The first PUT command creates the channel (if it does not exist), and adds a container to it; the subsequent commands add further containers to the channel. If the containers exist, their contents are overwritten by the new data.

AMODE (64) programs can create and use channels in the same way, by using **EXEC CICS PUT64 CONTAINER** commands. This command is for use only in non-Language Environment (LE) AMODE(64) assembler language application programs.

Note: New channels remain in scope until the link level changes , except for the transaction channel, which has the name DFHTRANSACTION. The transaction channel does not go out of scope when the link level changes: it is always accessible in the transaction . For more information about channel scope, see “The scope of a channel” on page 124.

Do not give a channel a name that starts with the characters *DFH* , apart from those that CICS defines as part of its API. These channels must be used only as described by that API.

An alternative way to add containers to a channel is to move them from another channel. To do this, use the following command:

```
EXEC CICS MOVE CONTAINER(
container-name
) AS(
container-new-name
)
CHANNEL(
channel-name1
) TOCHANNEL(
channel-name2
)
```

Note:

1. If the CHANNEL or TOCHANNEL option is not specified, the current channel is implied.
2. The source channel must be in program scope.
3. If the target channel does not exist in the current program scope, it is created.
4. If the source container does not exist, an error occurs.
5. If the target container does not exist, it is created; if the target container exists, its contents are overwritten.

6. When a channel is created, it exists until it goes out of scope. Any storage that is associated with it, either containers or set storage, also exists until the channel goes out of scope.

You can use MOVE CONTAINER, instead of GET CONTAINER and PUT CONTAINER, as a more efficient way of transferring data between channels.

If the channel named on the following commands does not exist in the current program scope, an *empty* channel is created:

- EXEC CICS LINK PROGRAM CHANNEL(*channel-name*)
- EXEC CICS RETURN TRANSID CHANNEL(*channel-name*)
- EXEC CICS START TRANSID CHANNEL(*channel-name*)
- EXEC CICS XCTL PROGRAM CHANNEL(*channel-name*)
- EXEC CICS RUN TRANSID CHANNEL(*channel-name*)

The current channel

A program's *current channel* is the channel - if there is one - with which it was invoked. These examples show how the current channel and its containers are passed between programs.

The program can create other channels. However, the current channel, for a particular invocation of a particular program, does not change. It is analogous to a parameter list.

Current channel example, with LINK commands

This example shows how a program passes the current channel and its containers to another program using the EXEC CICS LINK command.

The following figure illustrates the origin of a program's current channel. It shows five interactive programs. Program A is a top-level program started by, for example, a terminal user. It is not started by a program and does not have a current channel.

B, C, D, and E are second-level, third-level, fourth-level, and fifth-level programs.

Program B's current channel is X , passed by the CHANNEL option on the EXEC CICS LINK command issued by program A. Program B modifies channel X by adding one container and deleting another.

Program C's current channel is also X , passed by the CHANNEL option on the EXEC CICS LINK command issued by program B.

Program D has no current channel, because C does not pass it one.

Program E's current channel is Y , passed by the CHANNEL option on the EXEC CICS LINK command issued by D.

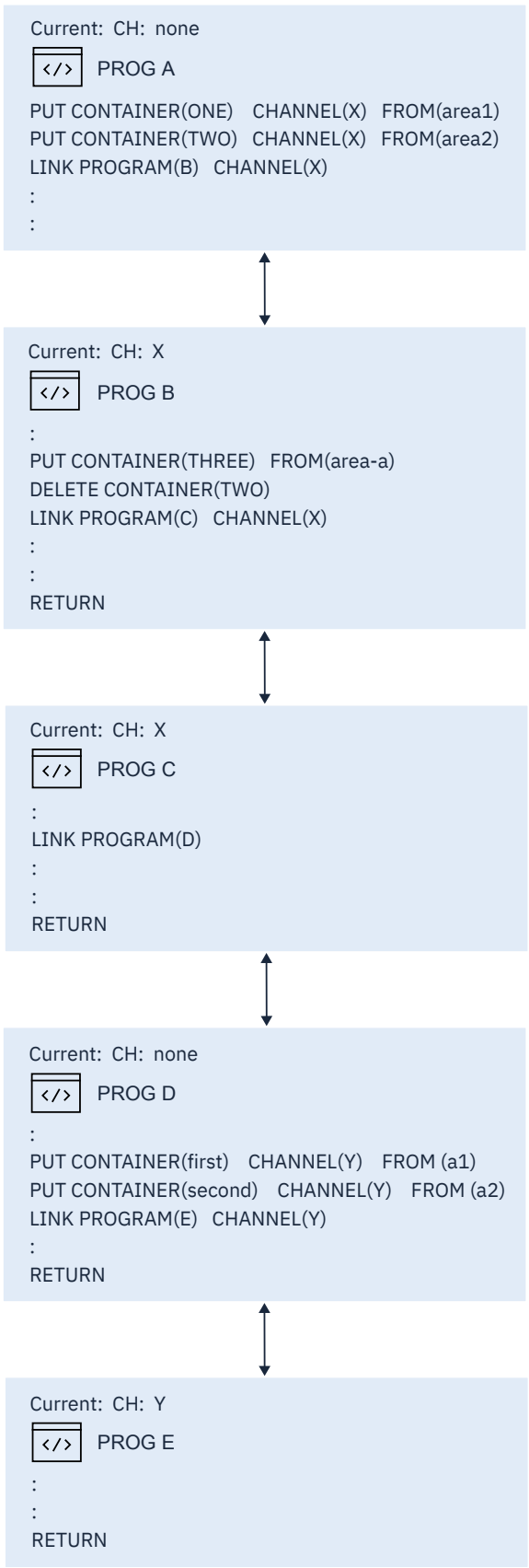


Figure 36. Current channel: example with LINK commands

The following table lists the name of the current channel (if any) of each of the five programs shown in the previous figure.

Table 14. The current channels of interactive programs—example with LINK commands

Prog.	Current CH	Issues commands	Comments
A	None	`. EXEC CICS PUT CONTAINER(ONE) CHANNEL(X) FROM(area1) EXEC CICS PUT CONTAINER(TWO) CHANNEL(X) FROM(area2) EXEC CICS LINK PROGRAM(B) CHANNEL(X) .`	Program A creates channel X and passes it to program B. Note that, by the time control is returned to program A by program B, the X channel has been modified—it does not contain the same set of containers as when it was created by program A. (Container TWO has been deleted and container THREE added by program B.)
B	X	`. EXEC CICS PUT CONTAINER(THREE) FROM(area-a) EXEC CICS DELETE CONTAINER(TWO) EXEC CICS LINK PROGRAM(C) CHANNEL(X).. EXEC CICS RETURN`	Program B modifies channel X (its current channel) by adding and deleting containers, and passes the modified channel to program C. Program B does not need to specify the CHANNEL option on the PUT CONTAINER and DELETE CONTAINER commands; its current channel is implied.
C	X	`. EXEC CICS LINK PROGRAM(D).. EXEC CICS RETURN`	Program C links to program D, but does not pass it a channel.
D	None	`. EXEC CICS PUT CONTAINER(first) CHANNEL(Y) FROM(a1) EXEC CICS PUT CONTAINER(second) CHANNEL(Y) FROM(a2) EXEC CICS LINK PROGRAM(E) CHANNEL(Y).. EXEC CICS RETURN`	Program D creates a new channel, Y, which it passes to program E.
E	Y	`. RETURN .`	Program E performs some processing on the data it's been passed and returns.

Current channel example, with XCTL commands

This example shows how a program passes the current channel and its containers to another program using the EXEC CICS XCTL command.

Figure 37 on page 123 shows four interactive programs. A1 is a top-level program started by, for example, a terminal user. It is not started by a program and does not have a current channel. B1, B2, and B3 are all second-level programs.

B1's current channel is X , passed by the CHANNEL option on the EXEC CICS LINK command issued by A1.

B2 has no current channel, because B1 does not pass it one.

B3's current channel is Y , passed by the CHANNEL option on the EXEC CICS XCTL command issued by B2.

When B3 returns, channel Y and its containers are deleted by CICS.

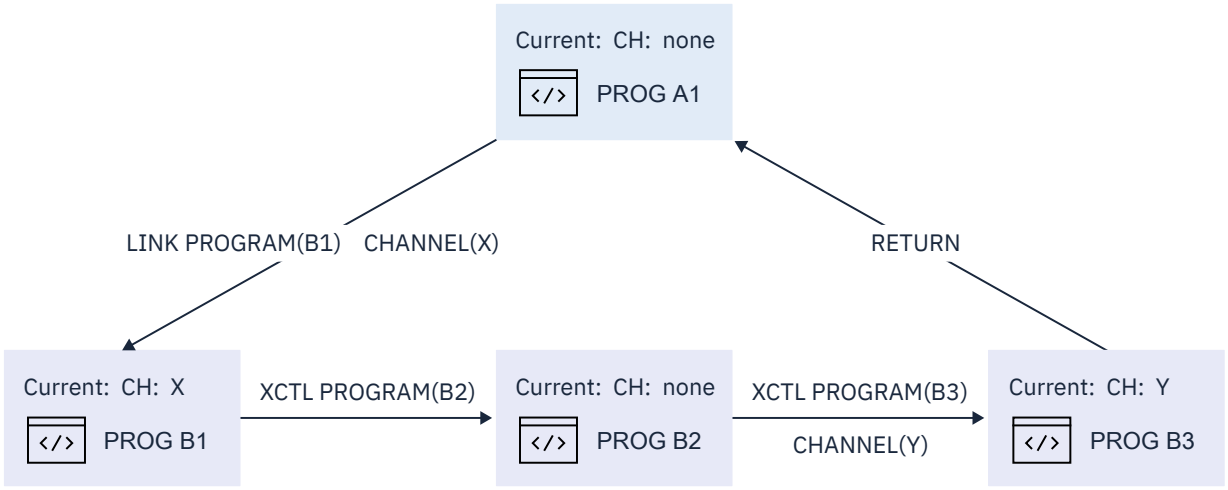


Figure 37. Current channels—example, with XCTL commands

The following table lists the name of the current channel (if any) of each of the four programs shown in Figure 37 on page 123.

Table 15. The current channels of interactive programs—example

Program	Current channel	Issues command
A1	None	`. EXEC CICS LINK PROGRAM(B1) CHANNEL(X) .`
B1	X	`. EXEC CICS XCTL PROGRAM(B2) .`
B2	None	`. EXEC CICS XCTL PROGRAM(B3) CHANNEL(Y) .`
B3	Y	`. EXEC CICS RETURN .`

Current channel: START, RUN TRANSID and RETURN commands

As well as the **LINK** and **XCTL** commands, you can pass channels on the **START** , **RUN TRANSID** and **RETURN** commands.

EXEC CICS START TRANSID(*tranid*) CHANNEL(*channel-name*)
The program that implements the started transaction (or the first program, if there are more than one) is passed the channel, which becomes its current channel.

Note: All EXEC CICS START requests that specify a channel cannot be deferred by specifying INTERVAL, AT, FOR or UNTIL as this is not supported. This is monitored by the CICS translator but not by JCICS. Using these commands to defer an EXEC CICS START request can result in a translator failure.

EXEC CICS RETURN TRANSID(*tranid*) CHANNEL(*channel-name*)
The CHANNEL option is valid only:

- On pseudoconversational RETURNs—that is, on RETURN commands that specify, on the TRANSID option, the next transaction to be run at the user terminal.
- If issued by a program at the highest logical level—that is, a program that returns control to CICS.

The program that implements the next transaction is passed the channel, which becomes its current channel.

The scope of a channel

The scope of a channel is the code from which it can be accessed. These examples show the scope of each channel in the diagram.

The scope is important because it defines the lifetime of the channel and container storage. For more information, see “Deleting channels and containers and freeing their storage” on page 129. For more information about LINK commands, see “Current channel example, with LINK commands” on page 120.

Scope example, with LINK commands

The scope of the X channel is programs A, B, and C. The scope of the Y channel is programs D and E.

Neither of these channels is the transaction channel **DFHTRANSACTION**. The scope of **DFHTRANSACTION** is the whole transaction.

By the time program B returns control to program A, the X channel has been modified - it does not contain the same set of containers as when it was created by program A.

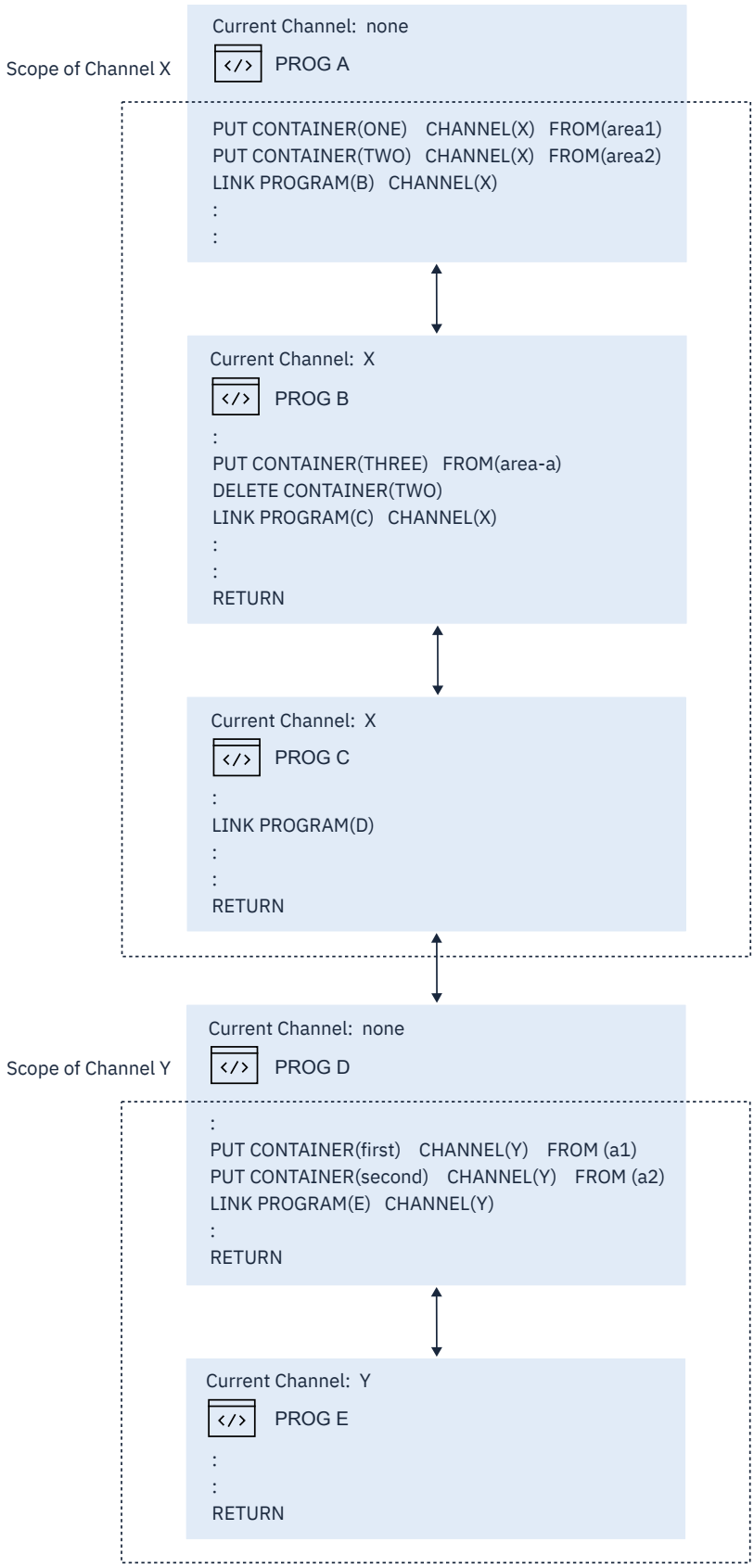


Figure 38. The scope of a channel - example showing LINK commands

The following table lists the name and scope of the current channel (if any) of each of the five programs discussed previously.

Table 16. The scope of a channel - example with LINK commands

Program	Current channel	Scope of channel
A	None	Not applicable
B	X	A, B, C
C	X	A, B, C
D	None	Not applicable
E	Y	D, E

The transaction channel DFHTRANSACTION

Channels go out of scope when the link level changes unless passed on the LINK command, which results in the channel not being available to all the programs in a transaction. To prevent the channel going out of scope, you can create a transaction channel by issuing a PUT CONTAINER command specifying a channel with the reserved name DFHTRANSACTION.

A transaction channel can be used on all commands that support the CHANNEL keyword, except the DELETE CHANNEL command, because CICS controls when a transaction channel is deleted. You can control when data is destroyed by deleting containers from the transaction channel rather than deleting the channel itself.

There can be only one transaction channel for each CICS transaction. That transaction channel is available to all programs running as part of the transaction in the local region, including any exit points that are API enabled.

A transaction channel is passed to a remote region along with the commarea or channel specified on the LINK command, as long as it meets the following specifications:

- The link between the two systems is MRO or IPIC. Systems using ISC over SNA are not supported.
- The target region is CICS TS 5.2 or higher, so supports transaction channels.
- The request is a DPL request, namely a function shipped LINK PROGRAM request. Transaction channels are not passed to a remote region with other function shipped requests, for example a function shipped File Control request.

If your transaction requires a transaction channel in a multi region environment, the transaction channel must be created in the first region that needs to use it. For example, if region A DPLs to region B, the transaction channel must be created in region A.

Scope example, with LINK and XCTL commands

This example adds to the diagram from "Current channel example, with XCTL commands" to show the scope of each channel.

For more information about XCTL commands, see “Current channel example, with XCTL commands” on page 122.

Figure 39 on page 128 shows the same four interactive programs previously described, plus a third-level program, C1, that is invoked by an **EXEC CICS LINK** command from program B1.

The scope of the X channel is restricted to A1 and B1.

The scope of the Y channel is B2 and B3.

The scope of the Z channel is B1 and C1.

None of these channels is the transaction channel DFHTRANSACTION, the scope of which would be the whole transaction.

Note that, by the time control is returned to program A1 by program B3, it is possible that the X channel can have been modified by program B1, it might not contain the same set of containers as when it was created by A1.

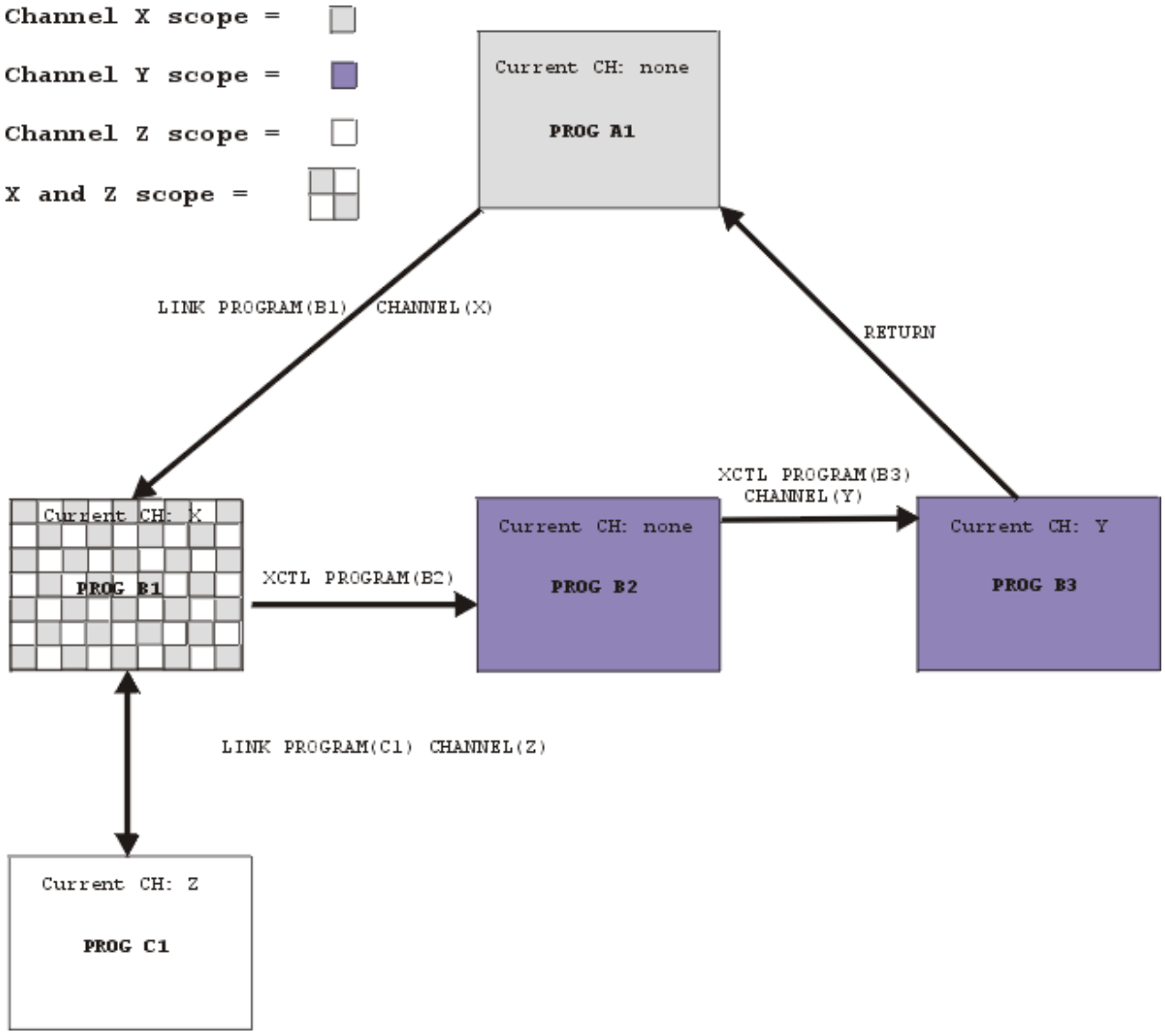


Figure 39. The scope of a channel—example showing LINK and XCTL commands

The following table lists the name and scope of the current channel (if any) of each of the five programs shown in Figure 39 on page 128.

Table 17. The scope of a channel—example with LINK and XCTL commands

Program	Current channel	Scope of channel
A1	None	Not applicable
B1	X	A1 and B1
B2	None	Not applicable
B3	Y	B2 and B3
C1	Z	B1 and C1

Discovering which containers were passed to a program

When a program is invoked, it can determine the names of the channel, and any containers that were passed to it.

Typically, programs that exchange a channel are written to handle that channel. That is, both client and server programs know the name of the channel, and the names and number of the containers in the channel. However, if, for example, a server program or component is written to handle more than one channel, on invocation it must discover which of the possible channels were passed to it.

A program can discover its current channel—that is, the channel with which it was invoked—by issuing an EXEC CICS ASSIGN CHANNEL command. (If there is no current channel, the command returns blanks.)

The program can count the number of containers that are in its current channel by issuing an EXEC CICS QUERY CHANNEL command. The command can be used with any channel, so the program must discover the name of the current channel first and specify it on the command. The program can also get the names of the containers in its current channel by browsing. Typically, these operations are not necessary. A program written to handle several channels is often coded to be aware of the names and number of the containers in each possible channel.

To get the names of the containers in the current channel, use the browse commands:

- EXEC CICS STARTBROWSE CONTAINER BROWSETOKEN(*data-area*) .
- EXEC CICS GETNEXT CONTAINER(*data-area*) BROWSETOKEN(*token*).
- EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(*token*).

Having retrieved the name of its current channel and, if necessary, the number or names of the containers in the channel, a server program can adjust its processing to suit the kind of data that was passed to it.

Discovering which containers were returned from a link

Following a LINK command, a program can discover the names of the containers returned by the program that was linked to.

A program creates a channel, which it passes to a second program by means of an EXEC CICS LINK PROGRAM(*program-name*) CHANNEL(*channel-name*) command. The second program performs some processing on the data that was passed to it, and returns the results in the same channel (its current channel). On return, the first program knows the name of the channel which has been returned, but not necessarily the number and names of the containers in the channel. (Does the returned channel contain the same containers as the passed channel, or has the second program deleted some or created others?)

The first program can count the number of containers that are in the channel by issuing an EXEC CICS QUERY CHANNEL command, specifying the name of the current channel. The first program can also discover the container-names by browsing. To do this, it uses the commands:

- EXEC CICS STARTBROWSE CONTAINER BROWSETOKEN(*data-area*) CHANNEL(*channel-name*).
- EXEC CICS GETNEXT CONTAINER(*data-area*) BROWSETOKEN(*token*).
- EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(*token*).

Deleting channels and containers and freeing their storage

Containers can be large objects, so it is important to manage the storage of these objects correctly to avoid causing your CICS regions to become short of storage. CICS automatically deletes containers and channels when they can no longer be used by the applications that had access to them. Applications can also issue commands to delete individual containers or complete channels.

When an application creates a channel by naming it for the first time on a command, and adds containers to it, CICS uses 64-bit storage (above 2 GB) for the data that is held in the containers. The use of 64-bit storage for channels and containers influences the value that you choose for the z/OS **MEMLIMIT** parameter that applies to the CICS region. You must also allow for other CICS facilities that use 64-bit storage. For more information, see Estimating, checking, and setting MEMLIMIT in Improving performance.

When an application program issues a **GET CONTAINER** command with the **SET** option, CICS creates a copy of the container in task storage, either in 24-bit storage (below 16 MB) or 31-bit storage (above 16 MB but below 2 GB), according to the DATALOCATION setting in the resource definition for the program. The application program can work with the contents of the container, and then use the **PUT CONTAINER** command to copy it back to 64-bit storage (above 2 GB). Non-Language Environment (LE) AMODE(64) assembly language application programs can work directly with channels and containers in 64-bit storage, by using **PUT64 CONTAINER** and **GET64 CONTAINER** commands.

Take care not to create so many large containers that you limit the amount of storage available to other applications. When you need to pass a large amount of data, split it between multiple containers, rather than put it all into one container. When you create a container, CICS checks if the action would cause the total 64-bit storage allocated for the transaction to exceed 5% of the MEMLIMIT value. In that situation, CICS issues warning message DFHPG0400 and does not create the container.

You can use the container storage policy task rule to define a threshold for the amount of container storage allocated to a user task and take an automatic action if the threshold is exceeded. This policy task rule alerts you to transactions that are using a lot of containers and channels or are using large containers, and helps you prevent such situations that cause message DFHPG0400 to be issued.

Deleting containers and freeing their storage

CICS automatically frees the 64-bit storage (above 2 GB) for all the containers that are on a channel when the channel goes out of scope in all programs in the link stack. The scope of a channel is the code from which it can be accessed, including the program that created the channel, and the programs to which the channel was passed. The exception is the transaction channel DFHTRANSACTION, which is available to all the programs in a transaction, and does not go out of scope when the link level changes. For more information about channel scope, see "The scope of a channel" on page 124.

If you want a container to be deleted before the channel goes out of scope, you can issue the **DELETE CONTAINER** command in an application program to delete a named container explicitly. When the command is issued, any data that is contained in the channel is discarded, and the 64-bit storage that was used for the container is freed. The application program that owns a channel can also delete all the containers that are in a channel by issuing the **DELETE CHANNEL** command.

When a program issues the **GET CONTAINER** command and CICS creates a copy of the container for the program to use, CICS maintains the copy of the container in 24-bit storage or 31-bit storage until any of the following occurs:

- A subsequent **GET CONTAINER** or **GET64 CONTAINER** command with the SET option, for the same container in the same channel, is issued by any program that can access this storage.
- The container is deleted by a **DELETE CONTAINER** command.
- The container is moved by a **MOVE CONTAINER** command.
- The channel goes out of program scope.
- The channel, and the containers that are in it, are deleted by a **DELETE CHANNEL** command.

When any of these events takes place, the copy of the container in 24-bit storage or 31-bit storage is deleted.

Deleting channels and freeing their storage

CICS automatically frees the 64-bit storage (above 2 GB) for a channel and for all the containers that are on it when the channel goes out of scope. Any copies of containers in the channel that still exist in 24-bit storage or 31-bit storage are also deleted at that time.

If you want to delete a whole channel before it goes out of scope, you can issue the **DELETE CHANNEL** command in an application program. The command deletes any containers that remain on the channel in a single operation. All the 64-bit storage used for the channel and its remaining containers is freed, and any copies of containers in the channel that still exist in 24-bit storage or 31-bit storage are also deleted.

The application program that issues the **DELETE CHANNEL** command must be the program that owns the channel. The program that owns the channel is the program that created the channel by naming it on one of the following commands:

- **LINK PROGRAM CHANNEL**
- **MOVE CONTAINER CHANNEL TOCHANNEL**
- **PUT CONTAINER CHANNEL**
- **PUT64 CONTAINER**
- **RETURN TRANSID CHANNEL**
- **START TRANSID CHANNEL**
- **XCTL PROGRAM CHANNEL**
- **WEB RECEIVE TOCHANNEL**
- **WEB CONVERSE TOCHANNEL**

An application program cannot delete the following channels:

- The current channel for the application program, that is, the channel with which the program was invoked.
- Any channel that the application program did not create.
- Any channel that is read-only.
- The transaction channel DFHTRANSACTION.

Designing a channel: Best practices

Containers are used to pass information between programs. These containers are grouped together in sets called channels. It is advisable to design your channels to follow a set of best practices.

About this task

At the end of a DPL call, input containers that the server program has not changed are not returned to the client. Input containers whose contents have been changed by the server program, and containers that the server program has created, are returned. Therefore, for optimal DPL performance, use the following best practices:

- Use separate containers for input and output data.
- Ensure that the server program, not the client, creates the output containers.
- Use separate containers for read-only and read-write data.
- If a structure is optional, make it a separate container.
- Use dedicated containers for error information.

If you are using **STARTBROWSE** or **GETNEXT** commands, be aware that the order in which containers are returned is undefined and might change. Applications should not rely on the order of returned containers. If you have existing applications that are written as such, see Upgrading applications for advice.

The following general tips on designing a channel include, and expand on, the recommendations to achieve optimal DPL performance:

- Use separate containers for input and output data. This provides the following benefits:
  - Better encapsulation of the data, making your programs easier to maintain.
  - Greater efficiency when a channel is passed on a DPL call, because smaller containers flow in each direction.
- Ensure that the server program, not the client, creates the output containers. If the client creates them, empty containers are sent to the server region.
- Use separate containers for read-only and read-write data. This provides the following benefits:

  - A simplification of your copybook structure, making your programs easier to understand.
  - Avoidance of the problems with REORDER overlays.
  - Greater transmission efficiency between CICS regions, because read-only containers sent to a server region will not be returned.
- Use separate containers for each structure. This provides the following benefits:
  - Better encapsulation of the data, making your programs easier to understand and maintain.
  - Greater ease in changing one of the structures, because you do not need to recompile the entire component.
  - The ability to pass a subset of the channel to subcomponents, by using the **MOVE CONTAINER** command to move containers between channels.
- If a structure is optional, make it a separate container. This leads to greater efficiency, because the structure is passed only if the container is present.
- Use dedicated containers for error information. This provides the following benefits:
  - Easier identification of error information.
  - Greater efficiency, for the following reasons:
    - The structure containing the error information is passed back only if an error occurs.
    - It is more efficient to check for the presence of an error container by issuing a **GET CONTAINER** (*known-error-container-name*) or **GET64 CONTAINER** (*known-error-container-name*) command (and possibly receiving a NOTFOUND condition) than it is to initiate a browse of the containers in the channel.
- When you need to pass data of different types, for example, character data in `code page1` and character data in `code page2` , use separate containers for each type, rather than one container with a complicated structure. This improves your ability to move between different code pages.
- When you need to pass a large amount of data, split it between multiple containers, rather than put it all into one container.

  When a channel is passed to a remote program or transaction, passing a large amount of data might affect performance. This is particularly true if the local and remote regions are connected by an ISC, rather than MRO, connection.

  Attention: Take care not to create so many large containers that you limit the amount of storage available to other applications.
- Channels and containers use storage below 2 GB (below the bar) and 64-bit storage (above-the-bar). For information on how storage is used and freed, see “Deleting channels and containers and freeing their storage” on page 129 . The use of 64-bit storage for channels and containers influences the value that you choose for the z/OS **MEMLIMIT** parameter that applies to the CICS region. You must also allow for other CICS facilities that use 64-bit storage. For more information, see Estimating, checking, and setting MEMLIMIT in Improving performance.

Constructing and using a channel: an example

In this example, a client program constructs a channel, passes it to a server program, and retrieves the server's output. The server program retrieves data from the channel's containers, and returns output to the client.

Figure 40 on page 133 shows a CICS client program that:

1. Uses EXEC CICS PUT CONTAINER commands to construct (and put data in) a set of containers. The containers are all part of the same named channel— " `payroll-2004` ".
2. Issues an EXEC CICS LINK command to invoke the PAYR server program, passing it the `payroll-2004` channel.
3. Issues an EXEC CICS GET CONTAINER command to retrieve the server's program output, which it knows will be in the `status` container of the `payroll-2004` channel.

```
* create the employee container on the payroll-2004
channel
EXEC CICS PUT CONTAINER('employee') CHANNEL('payroll-2004') FROM('John
Doe')

* create the wage container on the payroll-2004 channel
EXEC CICS PUT CONTAINER('wage') CHANNEL('payroll-2004') FROM('100')

* invoke the payroll service, passing the payroll-2004 channel
EXEC CICS LINK PROGRAM('PAYR') CHANNEL('payroll-2004')

* examine the status returned on the payroll-2004 channel
EXEC CICS GET CONTAINER('status') CHANNEL('payroll-2004') INTO(stat)
```

Figure 40. How a client program can construct a channel, pass it to a server program, and retrieve the server's output

Figure 41 on page 133 shows part of the PAYR server program invoked by the client. The server program:

1. Queries the channel with which it's been invoked.
2. Issues EXEC CICS GET CONTAINER commands to retrieve the input from the `employee` and `wage` containers of the `payroll-2004` channel.
3. Processes the input data.
4. Issues an EXEC CICS PUT CONTAINER command to return its output in the `status` container of the `payroll-2004` channel.

```
"PAYR", CICS COBOL server program

* discover which channel I've been invoked with
EXEC CICS ASSIGN CHANNEL(ch_name)
:
WHEN ch_name 'payroll-2004'
* my current channel is "payroll-2004"
* get the employee passed into this program
EXEC CICS GET CONTAINER('employee') INTO(emp)
* get the wage for this employee
EXEC CICS GET CONTAINER('wage') INTO(wge)
:
* process the input data
:
:
* return the status to the caller by creating the status container
* on the payroll channel and putting a value in it
EXEC CICS PUT CONTAINER('status') FROM('OK')
:
:
WHEN ch_name 'payroll-2005'
* my current channel is "payroll-2005"
:
:
:
```

Figure 41. How a server program can query the channel it's been passed, retrieve data from the channel's containers, and return output to the caller

CICS read-only containers

CICS can create channels and containers for its own use, and pass them to user programs. In some cases CICS marks these containers as read-only, so that the user program cannot modify data that CICS needs on return from the user program.

User programs cannot create read-only containers.

You cannot overwrite, move, or delete a read-only container. If you specify a read-only container on a **PUT CONTAINER** , **PUT64 CONTAINER** , **MOVE CONTAINER** , or **DELETE CONTAINER** command, an INVREQ

condition occurs. Programs cannot delete read-only channels using the **EXEC CICS DELETE CHANNEL** command.

Using channels with Java classes

CICS provides JCICS and JCICSX API classes that CICS Java programs can use to pass and receive channels

For information about using channels with JCICS, see Channel and container examples.

For examples on how to use channels with JCICSX, see JCICSX examples. Full samples can be found at JCICSX samples in GitHub.

Channels and BTS activities

The PUT, GET, MOVE, and DELETE CONTAINER commands used to build and interact with a channel are similar to those used in CICS business transaction services (BTS) applications.

Thus, programmers with experience of BTS will find it easy to use containers in non-BTS applications. Furthermore, server programs that use containers can be called from both channel and BTS applications. An example of this is shown in Figure 42 on page 135.

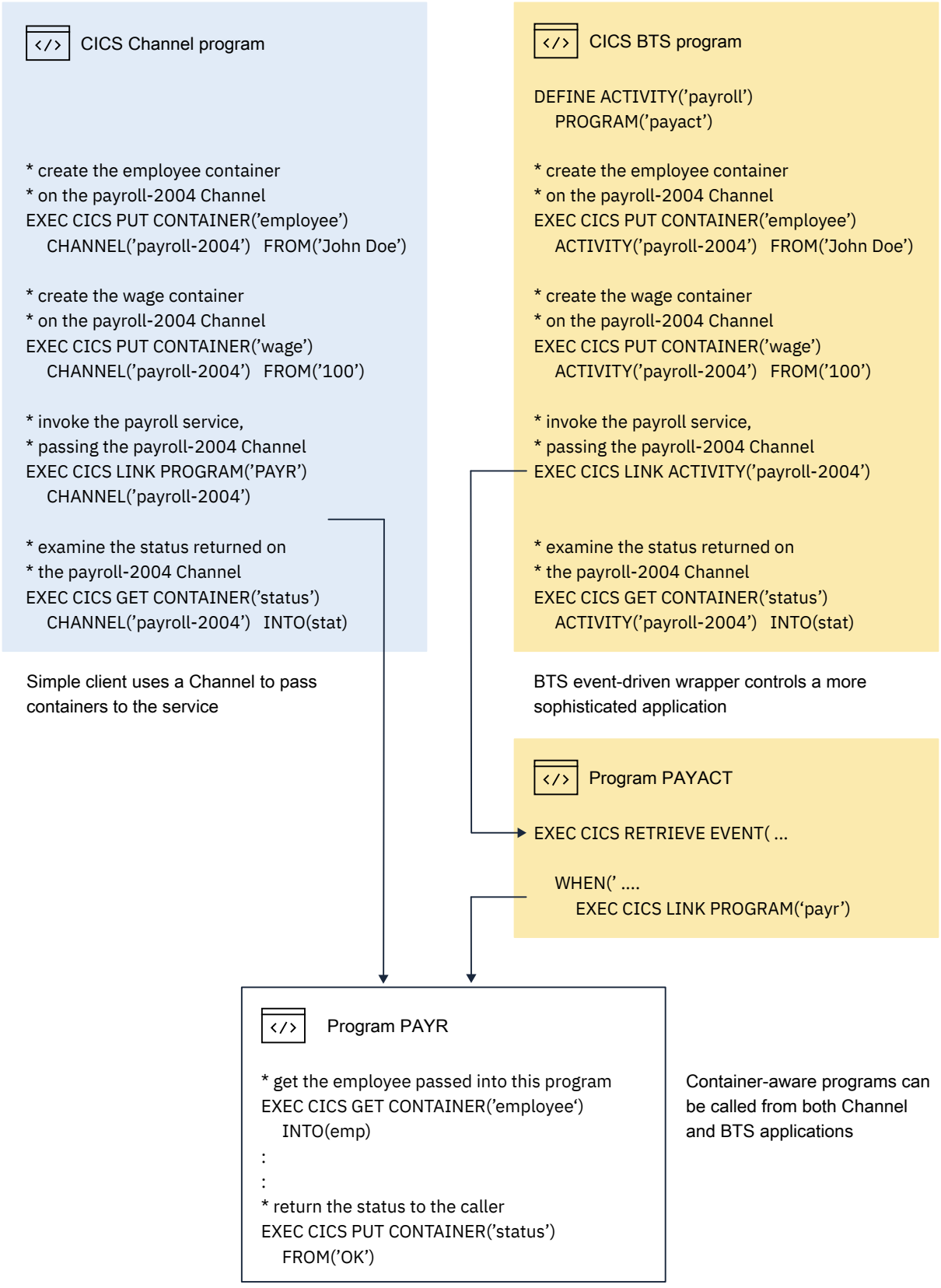


Figure 42. Channels and BTS activities

Context

A program that issues container commands can be used, without change, as part of a channel application or as part of a BTS activity.

For a program to be used in both a channel and a BTS context, the container commands that it issues must not specify any options that identify them as either channel or BTS commands. The options to be avoided on each of the container commands are:

DELETE CONTAINER

ACQACTIVITY (BTS-specific)
ACQPROCESS (BTS-specific)
ACTIVITY (BTS-specific)
CHANNEL (channel-specific)
PROCESS (BTS-specific)

GET CONTAINER

ACQACTIVITY (BTS-specific)
ACQPROCESS (BTS-specific)
ACTIVITY (BTS-specific)
CHANNEL (channel-specific)
INTOCCSID (channel-specific)
PROCESS (BTS-specific)

MOVE CONTAINER

FROMACTIVITY (BTS-specific)
CHANNEL (channel-specific)
FROMPROCESS (BTS-specific)
TOACTIVITY (BTS-specific)
TOCHANNEL (channel-specific)
TOPROCESS (BTS-specific)

PUT CONTAINER

ACQACTIVITY (BTS-specific)
ACQPROCESS (BTS-specific)
ACTIVITY (BTS-specific)
CHANNEL (channel-specific)
DATATYPE (channel-specific)
FROMCCSID (channel-specific)
PROCESS (BTS-specific)

When a container command is executed, CICS analyzes the context (channel, BTS, or neither) in which it occurs, in order to determine how to process the command. To determine the context, CICS uses the following sequence of tests:

1. Channel: does the program have a current channel?
2. BTS: is the program part of a BTS activity?
3. None: the program has no current channel and is not part of a BTS activity. It therefore has no context in which to execute container commands. The command is rejected with an INVREQ condition and a RESP2 value of 4.

Dynamic routing with channels

EXEC CICS LINK and EXEC CICS START commands, which pass channels, can be dynamically routed.

The following types of channel-related request can therefore be dynamically routed:

- Program-link (DPL) requests
- Transactions started by terminal-related START requests
- Non-terminal-related START requests

The routing program is passed, in the DYRCHANL field of its communication area, the name of the channel, if any, associated with the program-link or START command. The DYRCHANL field applies only to the three types of request as previously listed. For other types of request, or if there is no channel associated with the request, it contains blanks.

Note: The routing program's communication area is mapped by the DFHDYPDS DSECT.

Note that the routing program is given the *name* of the channel, not its address, and so is unable to use the DYRCHANL field to inspect or change the contents of the containers.

An application that uses a channel can create, within the channel, a special container named DFHROUTE. If the application issues a LINK or terminal-related START request (but not a non-terminal-related START request) that is to be dynamically routed, the dynamic routing program is given, in the DYRACMAA field of DFHDYPDS, the address of the DFHROUTE container, and can inspect and change its contents.

If you are migrating a program to pass a channel rather than a COMMAREA, you could use its existing COMMAREA structure to map DFHROUTE.

Data conversion

Application programs that use channels and containers frequently need to convert data from one code page to another.

Why is data conversion needed?

Here are some cases in which data conversion is necessary:

- When character data is passed between platforms that use different encoding standards; for example, EBCDIC and ASCII.
- When you want to change the encoding of some character data from one Coded Character Set Identifier (CCSID) to another. For an explanation of CCSIDs, and a list of the CCSIDs supported by CICS, see CICS-supported conversions.

Preparing for code page conversion with channels

CICS supports character conversions with the z/OS conversion services.

Before you begin

If you are preparing for code page conversions that involve Unicode data, see Unicode data conversion by z/OS

About this task

The conversion of data to or from either UTF-8 or UTF-16 and EBCDIC and ASCII code pages, depends on the selection of suitable conversion images. Conversion between the UTF-8 and UTF-16 forms of Unicode is also supported.

See the relevant appendix in z/OS Unicode Services User's Guide and Reference for the conversions that are supported though these services. These conversions include the ability to convert between a broad range of character encodings, including EBCDIC, ASCII, and Unicode.

Note:

1. The conversion between 037 and 500, as used, for example, with the IBM MQ transport, is an EBCDIC to EBCDIC conversion resulting from small differences in the character encodings used by CICS and IBM MQ.

2. Not all points in each code page have direct counterparts in other code pages, for example the EBCDIC character NL. Java and z/OS conversion services might differ in the conversions that they perform. Technotes and other Internet discussions might offer guidance on particular points. Programming communities themselves are divided on the question of what is the more appropriate conversion in particular circumstances.

CICS now supports any of these character conversions by using the z/OS conversion services. Earlier releases of CICS used a set of tables to carry out conversions, and conversions that use those tables continue to be supported. If CICS needs to carry out a conversion between a pair of CCSIDs that are not supported by these tables, the z/OS conversion services are used.

Ensuring that required conversion images are available
Those CCSIDs used as part of CICS applications must be made known to the system programmers responsible for maintaining the z/OS Conversion Image, so that specific conversions are available to the CICS regions where these applications execute.

Handling CCSID 1200
CICS supports conversions involving UTF-16 data using any of the following CCSIDs: 1200, 1201, and 1202. The z/OS conversion services permit CCSID 1200, in its big-endian form, to be used, but does not contain support for the little-endian form or for CCSIDs 1201 or 1202. CICS transforms any source data that is identified in any of these unsupported forms to the big-endian form of 1200 before passing the data to z/OS for conversion. If the target data is one of the unsupported forms then CICS receives the data as the big-endian form of 1200 and transforms it to the required CCSID. If the target CCSID is 1200 then CICS assumes the encoding to be in big-endian form. If the conversion is between any of these CCSIDs then CICS will carry out the transformation without calling the z/OS conversion services.

When setting up the z/OS conversion image for conversions involving any of these forms of UTF-16 then CCSID 1200 must be specified. CCSIDs 1201 and 1202 will not be recognized by z/OS when attempting to create a conversion image.

CICS respects the byte order marker for inbound conversions, but is not able to retain that information when handling a related outbound conversion. All outbound data for CCSID 1200 is UTF16- BE . Application programmers need to know about this and perform their own BE to LE conversions if they so require.

Java programs
Code page conversion facilities exist within Java , so it is not necessary to duplicate these in CICS . The conversion facilities described here do not extend to Java programs. For an example, see Putting data into a container.

Data conversion with channels

Applications that use channels to exchange data use a simple data conversion model. Frequently, no conversion is required and, when it is, a single programming instruction can be used to tell CICS to handle it automatically.

Note the following:

- Usually, when a (non-Java) CICS TS program calls another (non-Java) CICS TS program, no data conversion is required, because both programs use EBCDIC encoding. For example, if a CICS TS C-language program calls a CICS TS COBOL program, passing it some containers holding character data, the only reason for using data conversion would be the unusual one of wanting to change the CCSID of the data.
- The data conversion model used by channel applications is simple. The data in channel containers is converted under the control of the application programmer, by using API commands.
- The application programmer is responsible only for the conversion of user data; that is, the data in containers created by the application programs. System data is converted automatically by CICS , where necessary.
- The application programmer is concerned only with the conversion of character data. The conversion of binary data (between big-endian and little-endian) is not supported.

- Your applications can use the container API as a simple means of converting character data from one code page to another.

For data conversion purposes, CICS recognizes two types of data:

CHAR
Character data; that is, a text string. The data in the container is converted (if necessary) to the code page of the application that retrieves it. If the application that retrieves the data is a client on an ASCII-based system, this will be an ASCII code page. If it is a CICS Transaction Server for z/OS application, it will be an EBCDIC code page.

All the data in a container is converted as if it was a single character string. For single-byte character set (SBCS) code pages, a structure consisting of several character fields is equivalent to a single-byte character string. However, for double-byte character set (DBCS) code pages, this is not the case. If you use DBCS code pages, to ensure that data conversion works correctly you must put each character string into a separate container.

BIT
All non-character data; that is, everything that is not designated as being of type CHAR. The data in the container cannot be converted. This is the default value.

There are two ways to specify the code page for data conversion of the data in a container:

- As a Coded Character Set Identifier, or CCSID. A CCSID is a decimal number that identifies a specific code page. For example, the CCSID for the ASCII character set ISO 8859-1 is 819.
- As an IANA-registered charset name for the code page. This is an alphanumeric name that can be specified in charset= values in HTTP headers. For example, the IANA charset names supported by CICS for ISO 8859-1 are `iso-8859-1` and `iso_8859-1`.

If the application programmer does not specify a code page for data conversion, CICS uses the default code page for the whole of the local CICS region, which is specified on the **LOCALCCSID** system initialization parameter.

You can use the following API commands for data conversion:

- ```
  EXEC CICS PUT CONTAINER [CHANNEL] [DATATYPE] [FROMCCSID
  | FROMCODEPAGE]
  ```
- ```
  EXEC CICS GET CONTAINER [CHANNEL] [INTOCCSID |
  INTOCODEPAGE]
  ```

For non-Language Environment (LE) AMODE(64) assembler language application programs, you can also use the following API commands for data conversion for data in 64-bit storage:

- ```
  EXEC CICS PUT64 CONTAINER [CHANNEL] [DATATYPE]
  [FROMCCSID | FROMCODEPAGE]
  ```
- ```
  EXEC CICS GET64 CONTAINER [CHANNEL] [INTOCCSID |
  INTOCODEPAGE]
  ```

How to cause CICS to convert character data automatically

You can use the DATATYPE(DFHVALUE(CHAR)) option of the PUT CONTAINER or PUT64 CONTAINER command to specify that a container holds character data eligible for conversion. If the client and server platforms are different, the GET CONTAINER or GET64 CONTAINER command converts the data automatically.

About this task

The following procedure demonstrates how to convert character data automatically by using PUT CONTAINER and GET CONTAINER commands.

For AMODE(64) programs, you can convert character data that is in 64-bit storage in the same way by using PUT64 CONTAINER and GET64 CONTAINER commands. These commands are for use only in non-

Language Environment (LE) AMODE(64) assembler language application programs, and CICS business transaction services (BTS) containers are not supported.

Procedure

1. In the **client program** , use the DATATYPE(DFHVALUE(CHAR)) option of the PUT CONTAINER command to specify that a container holds character data and that the data is eligible for conversion. For example:

```
EXEC CICS PUT CONTAINER(
cont_name
) CHANNEL('payroll')
FROM(
data1
) DATATYPE(DFHVALUE(CHAR))
```

   There is no need to specify the FROMCCSID or FROMCODEPAGE option unless the data is not in the default CCSID of the client platform. (For CICS TS regions, the default CCSID is specified on the LOCALCCSID system initialization parameter.) The default CCSID is implied.

2. In the **server program** , issue a GET CONTAINER command to retrieve the data from the program's current channel:

```
EXEC CICS GET CONTAINER(
cont_name
) INTO(
data_area1
)
```

   The data is returned in the default CCSID of the server platform. There is no need to specify the INTOCCSID or INTOCODEPAGE option unless you want the data to be converted to a CCSID other than the default. If the client and server platforms are different, data conversion takes place automatically.

3. In the **server program** , issue a PUT CONTAINER command to return a value to the client:

```
EXEC CICS PUT CONTAINER(
status
) FROM(
data_area2
)
DATATYPE(DFHVALUE(CHAR))
```

   The DATATYPE(DFHVALUE(CHAR)) option specifies that the container holds character data and that the data is eligible for conversion. There is no need to specify the FROMCCSID or FROMCODEPAGE option unless the data is not in the default CCSID of the server platform.

4. In the **client program** , issue a GET CONTAINER command to retrieve the status returned by the server program:

```
EXEC CICS GET CONTAINER(
status
) CHANNEL('payroll')
INTO(
status_area
)
```

   The status is returned in the default CCSID of the client platform. There is no need to specify the INTOCCSID or INTOCODEPAGE option unless you want the data to be converted to a CCSID other than the default. If the client and server platforms are different, data conversion takes place automatically.

Using containers for code page conversion

Your applications can use the container API to convert character data from one code page to another.

About this task

The following example converts data from code page1 to code page2 :

```
EXEC CICS PUT CONTAINER(
temp
) DATATYPE(DFHVALUE(CHAR))
FROMCCSID(
code page1
) FROM(
input-data
)
EXEC CICS GET CONTAINER(
temp
) INTOCCSID(
code page2
)
SET(
data-ptr
) FLENGTH(
data-len
)
```

The following example converts data from the region's default EBCDIC code page to a specified UTF8 code page:

```
EXEC CICS PUT CONTAINER(
temp
) DATATYPE(DFHVALUE(CHAR))
FROM(
ebcdic-data
)
EXEC CICS GET CONTAINER(
temp
) INTOCCSID(utf8_ccsid)
SET(
utf8-data-ptr
) FLENGTH(
utf8-data-len
)
```

When using the container API in this way, bear the following in mind:

- On GET CONTAINER commands, use the SET option, rather than INTO, unless the converted length is known. (You can retrieve the length of the converted data by issuing a GET CONTAINER(*cont_name*) NODATA FLENGTH(*len*) command.)
- If you prefer to specify a supported IANA charset name for the code pages, rather than the decimal CCSIDs, or if you want to specify a CCSID alphanumerically, use the FROMCODEPAGE and INTOCODEPAGE options instead of the FROMCCSID and INTOCCSID options.
- To avoid a storage overhead, after conversion copy the converted data and delete the container.
- To avoid shipping the channel, use a temporary channel. For more information about temporary storage and use of channels, see “Avoiding affinities when using temporary storage” on page 170.
- All-to-all conversion is not possible. That is, a code page conversion error occurs if a specified code page and the channel's code page are an unsupported combination.

A SOAP example

You can use a CICS TS SOAP application to retrieve a UTF-8 or UTF-16 message from a socket or IBM MQ message queue, put a message into a container in UTF-8 format, put EBCDIC data structures into other containers on the same channel, or make a distributed program link (DPL) call to a handler program on a back end AOR, passing the channel.

The back end handler program, also running on CICS TS, can use EXEC CICS GET CONTAINER commands to retrieve the EBCDIC data structures or the messages. It can get the messages in UTF-8 or UTF-16, or in its own or the region's EBCDIC code page. Similarly, it can use EXEC CICS PUT CONTAINER commands to place data into the containers, in UTF-8, UTF-16, or EBCDIC.

To retrieve one of the messages in the region's EBCDIC code page, the handler can issue the command:

```
EXEC CICS GET CONTAINER(
input_msg
) INTO(
msg
)
```

Because the INTOCCSID and INTOCODEPAGE options are not specified, the message data is automatically converted to the region's EBCDIC code page. (This assumes that the PUT CONTAINER command used to store the message data in the channel specified a DATATYPE of CHAR; if it specified a DATATYPE of BIT, the default, no conversion is possible.)

To return some output in the region's EBCDIC code page, the handler can issue the command:

```
EXEC CICS PUT CONTAINER(
output
) FROM(
output_msg
)
```

Because CHAR is not specified, no data conversion is permitted. Because the FROMCCSID and FROMCODEPAGE options are not specified, the message data is taken to be in the region's EBCDIC code page.

To retrieve one of the messages in UTF-8, the INTOCCSID or INTOCODEPAGE option must be specified to identify the code page and prevent automatic conversion of the data to the region's EBCDIC code page. The handler can issue the command:

```
EXEC CICS GET CONTAINER(
input_msg
) INTO(
msg
) INTOCCSID(utf8)
```

In this case, `utf8` is a variable which is defined as a fullword, and is initialized to 1208, which is the Coded Character Set Identifier, or CCSID, for UTF-8. If you prefer to use an IANA charset name for the code page, you can use the INTOCODEPAGE option instead of the INTOCCSID option:

```
EXEC CICS GET CONTAINER(
input_msg
) INTO(
msg
) INTOCODEPAGE(utf8)
```

In this case, `utf8` is a variable which is defined as a character string of length 56, and is initialized to 'utf-8'.

To return some output in UTF-8, the server program can issue the command:

```
EXEC CICS PUT CONTAINER(
output
) FROM(
output_msg
) FROMCCSID(utf8)
```

or alternatively:

```
EXEC CICS PUT CONTAINER(
```

```
output
) FROM(
output_msg
) FROMCODEPAGE(utf8)
```

where the variable `utf8` is defined and initialized in the same ways as for INTOCCSID and INTOCODEPAGE. The FROMCCSID or FROMCODEPAGE option specifies that the message data is currently in UTF-8 format. Because FROMCCSID or FROMCODEPAGE is specified, a DATATYPE of CHAR is implied, so data conversion is permitted.

Migrating from COMMAREAs to channels

CICS application programs that use traditional communications areas (COMMAREAs) to exchange data can continue to work as before. If you want to migrate to channels, here are examples of how to migrate several types of existing application to use channels and containers rather than COMMAREAs.

It is possible to replace a COMMAREA by a channel with a single container. While this may seem the simplest way to move from COMMAREAs to channels and containers, it is not good practice to do this. Because you're taking the time to change your application programs to exploit this new function, you should implement the "best practices" for channels and containers; see “Designing a channel: Best practices” on page 131. Channels have several advantages over COMMAREAs (see “Benefits of channels” on page 118) and it pays to design your channels to make the most of these improvements.

Also, be aware that a channel may use more storage than a COMMAREA designed to pass the same data. (See “Benefits of channels” on page 118 .)

User-written dynamic or distributed routing programs require work whether or not you plan to implement channels and containers in your own applications. If you employ a user-written dynamic or distributed routing program for workload management, rather than CICSPlex SM, you must modify your program to handle the new values that it may be passed in the DYRLEVEL, DYRTYPE, and DYRVER fields of the DFHDYPDS communications area. See Parameters passed to the dynamic routing program.

Migrating LINK commands that pass COMMAREAs

To migrate two programs which use a COMMAREA on a LINK command to exchange a structure, change the instructions shown in this table.

About this task

In these instructions, `structure` is the name of your defined data structure. The EXEC CICS GET CONTAINER and PUT CONTAINER commands use a 16–character field to identify the container. A helpful convention is to give the container the same name as the data structure that you are using, shown here as `structure-name`.

Table 18. Migrating LINK commands that pass COMMAREAs

Program	Before	After
PROG1	`EXEC CICS LINK PROGRAM(PROG2) COMMAREA(structure)`	`EXEC CICS PUT CONTAINER(structure-name) CHANNEL(channel-name) FROM(structure) EXEC CICS LINK PROGRAM(PROG2) CHANNEL(channel-name) ⋮` `EXEC CICS GET CONTAINER(structure-name) CHANNEL(channel-name) INTO(structure)`

Table 18. Migrating LINK commands that pass COMMAREAs (continued)

Program	Before	After

PROG2	`EXEC CICS ADDRESS COMMAREA(structure-ptr)`
	`...`
	`RETURN`
		`EXEC CICS GET CONTAINER(structure-name)`
		`INTO(structure)`
		`...`
		`EXEC CICS PUT CONTAINER(structure-name)`
		`FROM(structure)`
		`RETURN`

Note: In the COMMAREA example, PROG2, having put data in the COMMAREA, has onl y to issue a RETURN command to return the data to PROG1. In the channel example, to return data *PROG2 must issue a PUT CONTAINER command before the RETURN*.

Migrating XCTL commands that pass COMMAREAs

To migrate two programs which use a COMMAREA on an XCTL command to pass a structure, change the instructions shown in this table.

About this task

In these instructions, `structure` is the name of your defined data structure. The EXEC CICS GET CONTAINER and PUT CONTAINER commands use a 16–character field to identify the container. A helpful convention is to give the container the same name as the data structure that you are using, shown here as `structure-name`.

Table 19. Migrating XCTL commands that pass COMMAREAs

Program	Before	After

PROG1	`EXEC CICS XCTL PROGRAM(PROG2)`
	`COMMAREA(structure)`
		`EXEC CICS PUT CONTAINER(structure-name)`
		`CHANNEL(channel-name) FROM(structure)`
		`EXEC CICS XCTL PROGRAM(PROG2)`
		`CHANNEL(channel-name)`
		⋮

PROG2	`EXEC CICS ADDRESS COMMAREA(structure-ptr)`
	`...`
		`EXEC CICS GET CONTAINER(structure-name)`
		`INTO(structure)`
		`...`

Migrating pseudoconversational COMMAREAs on RETURN commands

To migrate two programs which use COMMAREAs to exchange a structure as part of a pseudoconversation, change the instructions shown in this table.

About this task

In these instructions, `structure` is the name of your defined data structure. The EXEC CICS GET CONTAINER and PUT CONTAINER commands use a 16–character field to identify the container. A helpful convention is to give the container the same name as the data structure that you are using, shown here as `structure-name`.

Table 20. Migrating pseudoconversational COMMAREAs on RETURN commands

Program	Before	After
PROG1	`EXEC CICS RETURN TRANSID(PROG2) COMMAREA(structure)`	`EXEC CICS PUT CONTAINER(structure-name) CHANNEL(channel-name) FROM(structure) EXEC CICS RETURN TRANSID(TRAN2) CHANNEL(channel-name)`
PROG2	`EXEC CICS ADDRESS COMMAREA(structure-ptr)`	`EXEC CICS GET CONTAINER(structure-name) INTO(structure)`

Migrating START data

To migrate two programs which use START data to exchange a structure, change the instructions shown in this table.

About this task

In these instructions, `structure` is the name of your defined data structure. The EXEC CICS GET CONTAINER and PUT CONTAINER commands use a 16–character field to identify the container. A helpful convention is to give the container the same name as the data structure that you are using, shown here as `structure-name`.

Table 21. Migrating START data

Program	Before	After
PROG1	`EXEC CICS START TRANSID(TRAN2) FROM(structure)`	`EXEC CICS PUT CONTAINER(structure-name) CHANNEL(channel-name) FROM(structure) EXEC CICS START TRANSID(TRAN2) CHANNEL(channel-name)`
PROG2	`EXEC CICS RETRIEVE INTO(structure)`	`EXEC CICS GET CONTAINER(structure-name) INTO(structure)`

Note that the new version of PROG2 is the same as that in the pseudoconversational example.

Migrating programs that use temporary storage to pass data

In older CICS releases where channels were not available, some applications used temporary storage queues (TS queues) to pass more than 32K of data from one program to another. Typically, this involved multiple writes to and reads from a TS queue.

About this task

If you migrate one of these applications to use channels, be aware of the following points:

- If the TS queue used by your existing application is in main storage, the storage requirements of the new, migrated, application are likely to be similar to those of the existing application.
- If the TS queue used by your existing application is in auxiliary storage, the storage requirements of the migrated application are likely to be greater than those of the existing application. This is because container data is held in storage rather than being written to disk.

Migrating dynamically-routed applications

EXEC CICS LINK and **EXEC CICS START** commands, which can pass either COMMAREAs or channels, can be dynamically routed. You can migrate these commands to use a channel in place of a COMMAREA.

When a LINK or START command passes a COMMAREA rather than a channel, the routing program can, depending on the type of request, inspect or change the COMMAREA's contents. For LINK requests and transactions started by terminal-related START requests (which are handled by the *dynamic* routing program) but not for non-terminal-related START requests (which are handled by the *distributed* routing program) the routing program is given, in the DYRACMAA field of its communication area, the *address* of the application's COMMAREA, and can inspect and change its contents.

Note: The routing program's communication area is mapped by the DFHDYPDS DSECT.

If you migrate a dynamically-routed **EXEC CICS LINK** or **START** command to use a channel rather than a COMMAREA, the routing program is passed, in the DYRCHANL field of DFHDYPDS, the name of the channel. Note that the routing program is given the *name* of the channel, not its address, and so is unable to use the DYRCHANL field to inspect or change the contents of the channel's containers.

To give the routing program the same kind of functionality with channels, an application that uses a channel can create, within the channel, a special container named DFHROUTE. If the application issues a LINK or terminal-related START request (but not a non-terminal-related START request) that is to be dynamically routed, the dynamic routing program is given, in the DYRACMAA field of DFHDYPDS, the address of the DFHROUTE container, and can inspect and change its contents.

If you are migrating a program to pass a channel rather than a COMMAREA, you could use its existing COMMAREA structure to map DFHROUTE.

For introductory information about dynamic and distributed routing, see Introduction to CICS dynamic routing . For information about writing a dynamic or distributed routing program, see Writing a dynamic routing program .

Program control

The CICS program control facility governs the flow of control between application programs in a CICS system.

Java and C++

The application programming interface described here is the CICS API, which is not used in Java programs. For information about Java programs using the JCICS classes to access program control services, see Java development using JCICS and the JCICS Javadoc documentation. For information about C++ programs using the CICS C++ classes, see Using the CICS foundation classes.

Program control using the CICS API

The name of the application referred to in a program control command must have been defined as a program to CICS. You can use program control commands as follows:

- You can link one of your application programs to another, anticipating subsequent return to the requesting program (LINK command). The COMMAREA, CHANNEL, and INPUTMSG options of this command allow data to be passed to the requested application program.
- You can link one of your application programs to another program in a separate CICS region, anticipating subsequent return to the requesting program (LINK command). The COMMAREA or CHANNEL option of this command allows data to be passed to the requested application program. This is referred to as distributed program link (DPL). (You cannot use the INPUTMSG and INPUTMSGLEN options of a LINK command when using DPL. See “CICS intercommunication” on page 209 for more information about DPL.
- You can transfer control from one of your application programs to another, with no return to the requesting program (XCTL command). The COMMAREA, CHANNEL, and INPUTMSG options of this command allow data to be passed to the requested application program. (You cannot use

the INPUTMSG and INPUTMSGLEN options of an XCTL command when using DPL. See "CICS intercommunication" on page 209 for more information about DPL.

- You can return control from one of your application programs to another, or to CICS (RETURN command). The COMMAREA, CHANNEL, and INPUTMSG options of this command allow data to be passed to a newly initiated transaction. You cannot use the INPUTMSG and INPUTMSGLEN options of a RETURN command when using DPL. See "CICS intercommunication" on page 209 for more information about DPL.
- You can load a designated application program, table, or map into main storage (LOAD command).

  If you use the HOLD option with the LOAD and RELEASE command to load a program, table, or map that is not read-only, you could create inter-transaction affinities that could adversely affect your ability to perform dynamic transaction routing.

  To help you identify potential problems with programs that issue these commands, you can use the CICS Interdependency Analyzer. See CICS Interdependency Analyzer for z/OS Overview for more information about this utility and see "Affinity" on page 156 for more information about transaction affinity.
- You can delete a previously loaded application program, table, or map from main storage (RELEASE command).

You can use the RESP option to deal with abnormal terminations.

Program linking

A **LINK** command is used to pass control from an application program at one logical level to an application program at the next lower logical level.

Application program logical levels

Application programs running under CICS are executed at various *logical levels*. The first program to receive control within a task is at the highest logical level.

When an application program is linked to another, expecting an eventual return of control, the linked-to program is considered to reside at the next lower logical level. When control is transferred from one application program to another, without expecting return of control, the two programs are considered to reside at the same logical level.

Link to another program expecting return

If the program receiving control is not already in main storage, it is loaded. When a RETURN command is processed in the linked program, control is returned to the program initiating the link at the next sequential process instruction.

The linked program operates independently of the program that issues the LINK command regarding handling exception conditions, attention identifiers, and abends. For example, the effects of HANDLE commands in the linking program are not inherited by the linked-to program, but the original HANDLE commands are restored on return to the linking program. You can use the HANDLE ABEND command to deal with abnormal terminations in other link levels. Figure 43 on page 148 shows the concept of logical levels.

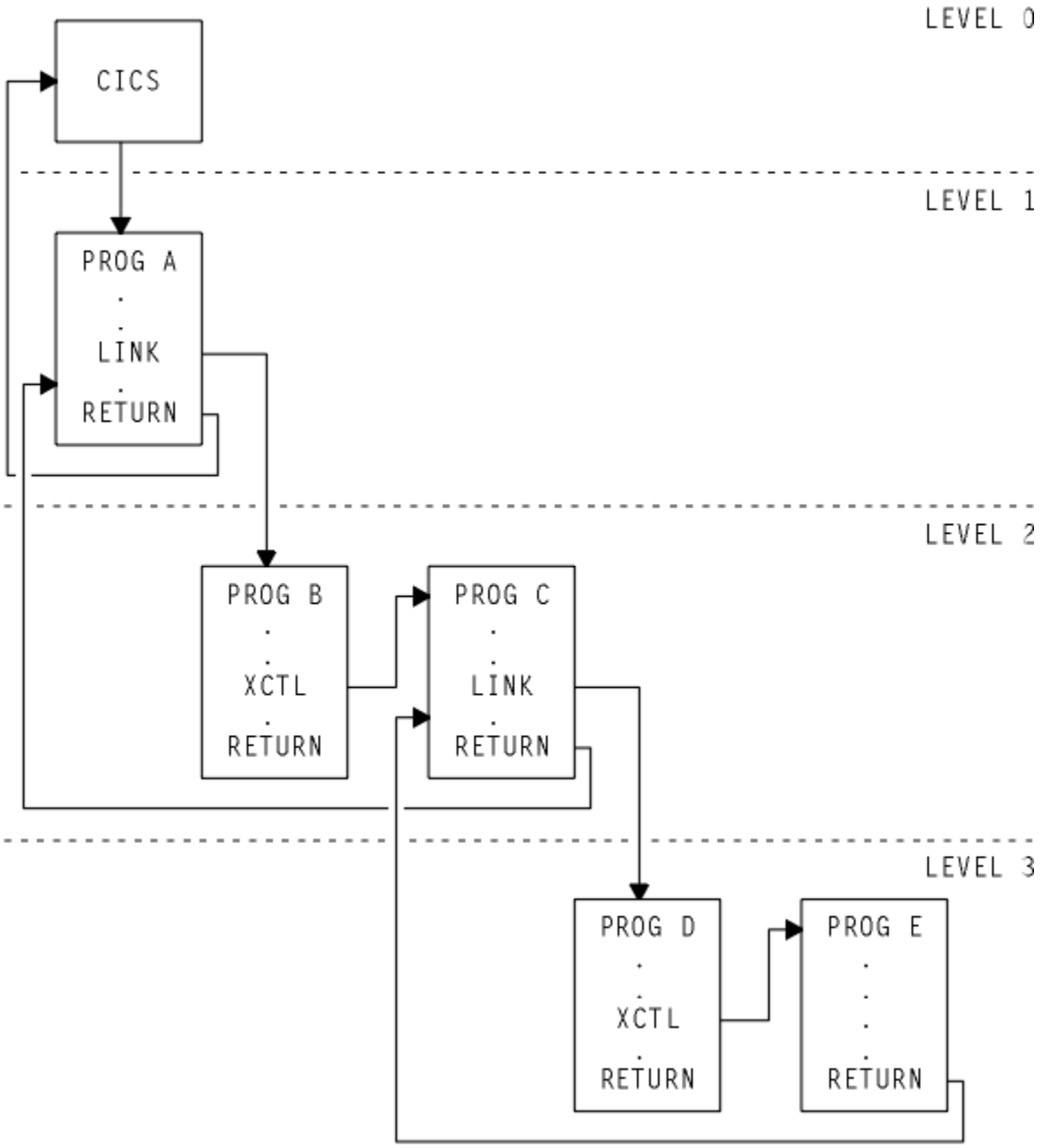


Figure 43. Application program logical levels

Passing data to other programs

You can pass data to another program by using the **EXEC CICS** program control commands LINK, XCTL, and RETURN, and using the COMMAREA, CHANNEL, or INPUTMSG options of those commands. COMMAREA and CHANNEL are mutually exclusive.

Instead of using a COMMAREA, using a channel is a more modern method of transferring data between CICS programs. Channels have several advantages over COMMAREAs. For details, see "Transferring data between programs using channels" on page 108.

Passing data to other programs by using COMMAREA

The COMMAREA specifies the name of a data area (known as a *communication area*) in which data is passed to a program or transaction. It is an option of the LINK, XCTL, and RETURN commands.

The COMMAREA option of LINK and XCTL commands specifies the name of a data area in which data is passed to the program being invoked.

The COMMAREA option of a RETURN command specifies the name of a communication area in which data is passed to the transaction identified in the TRANSID option. The TRANSID option specifies a transaction that is initiated when the next input is received from the terminal associated with the task.

The invoked program receives the data as a parameter. The program must contain a definition of a data area to allow access to the passed data.

In a receiving COBOL program, you must give the data area the name DFHCOMMAREA. In this COBOL program, if a program passes a COMMAREA as part of a LINK, XCTL, or RETURN command, either the working-storage or the LINKAGE SECTION can contain the data area. A program receiving a COMMAREA should specify the data in the LINKAGE SECTION. This applies when the program is either of the following:

- The receiving program during a LINK or XCTL command where a COMMAREA is passed
- The initial program, where a RETURN command of a previously called task specified a COMMAREA and TRANSID

In a C or C++ program that is receiving a COMMAREA, the COMMAREA must be defined as a pointer to a structure. The program then must issue the ADDRESS COMMAREA command to gain addressability to the passed data.

In a PL/I program, the data area can have any name, but it must be declared as a based variable, based on the parameter passed to the program. The pointer to this based variable should be declared explicitly as a pointer rather than contextually by its appearance in the declaration for the area. This prevents the generation of a PL/I error message. No ALLOCATE statement can be processed within the receiving program for any variable based on this pointer. This pointer must not be updated by the application program.

In an assembler language program, the data area should be a DSECT mapping. The register used to address this data area must be loaded from DFHEICAP (communication area pointer), mapped by the DFHEISTG DSECT. A COMMAREA cannot be in 64-bit storage.

The receiving data area can the same length as, or shorter than, the original communication area, but must not be longer. If access is required only to the first part of the data, the receiving data area can be shorter. If the receiving data area is longer than the original communication area, your transaction might attempt to read data outside the area that has been passed. It might also overwrite data outside the area, which might cause CICS to abend.

To avoid this happening, your program should check whether the length of any communication area that has been passed to it is as expected, by accessing the EIBCALEN field in the EIB of the task. If no communication area has been passed, the value of EIBCALEN is zero. Otherwise, EIBCALEN always contains the value specified in the LENGTH option of a LINK, XCTL, or RETURN command, regardless of the size of the data area in the invoked program. Ensure that the value in EIBCALEN matches the value in the DSECT for your program, and ensure that your transaction accesses data in that area.

You can also add an identifier to COMMAREA as an additional check on the data that is being passed. This identifier is sent with the sending transaction and is checked for by the receiving transaction.

When a communication area is passed by using a LINK command, the invoked program is passed a pointer to the communication area itself. Any changes made to the contents of the data area in the invoked program are available to the invoking program, when control returns to it. To access any such changes, the program names the data area specified in the original COMMAREA option.

When a communication area is passed by using an XCTL command, a copy of that area is made, unless the area to be passed has the same address and length as the area that was passed to the program issuing the command. For example, if program A issues a LINK command to program B, which in turn issues an XCTL command to program C, and if B passes to C the same communication area that A passed to B, program C will be passed addressability to the communication area that belongs to A (not a copy of it), and any changes made by C will be available to A when control returns to it.

When a lower-level program that has been accessed by a LINK command issues a RETURN command, control passes back one logical level higher than the program returning control. If the task is associated with a terminal, the TRANSID option can be used at the lower level to specify the transaction identifier for the next transaction to be associated with that terminal. The transaction identifier comes into play only after the highest logical level has relinquished control to CICS using a RETURN command and input is received from the terminal. Any input entered from the terminal, apart from an attention key, is interpreted wholly as data. You can use the TRANSID option without COMMAREA when returning from any link level, but it can be overridden on a later RETURN command. If a RETURN command fails at the top level because of an invalid COMMAREA, the TRANSID becomes null. Also, you can specify COMMAREA or IMMEDIATE only at the highest level, otherwise you get an INVREQ with RESP2=2.

In addition, the COMMAREA option can be used to pass data to the new task that is to be started.

The invoked program can determine which type of command invoked it by accessing field EIBFN in the EIB. This field must be tested before any CICS commands are issued. If the program was invoked by a LINK or XCTL command, the appropriate code is found in the EIBFN field. If it was invoked by a RETURN command, no CICS commands have been issued in the task, and the field contains zeros.

Passing data to other programs by using INPUTMSG

The INPUTMSG option of LINK, XCTL, and RETURN commands is another way of specifying the name of a data area to be passed to the program being invoked.

In this case, the invoked program gets the data by processing a RECEIVE command. This option enables you to invoke ("front end") application programs that were written to be invoked directly from a terminal, and which contain RECEIVE commands, to obtain initial terminal input.

If program that has been accessed with a LINKcommand issues a RECEIVE command to obtain initial input from a terminal, but the initial RECEIVE request has already been issued by a higher-level program, there is no data for the program to receive. In this case, the application waits on input from the terminal. You can ensure that the original terminal input continues to be available to a linked program by invoking it with the INPUTMSG option.

When an application program invokes another program, specifying INPUTMSG on LINK(or XCTL or RETURN) command, the data specified on the INPUTMSG continues to be available even if the linked program itself does not issue an RECEIVE command, but instead invokes yet another application program. See Figure 44 on page 151 for an illustration of INPUTMSG.

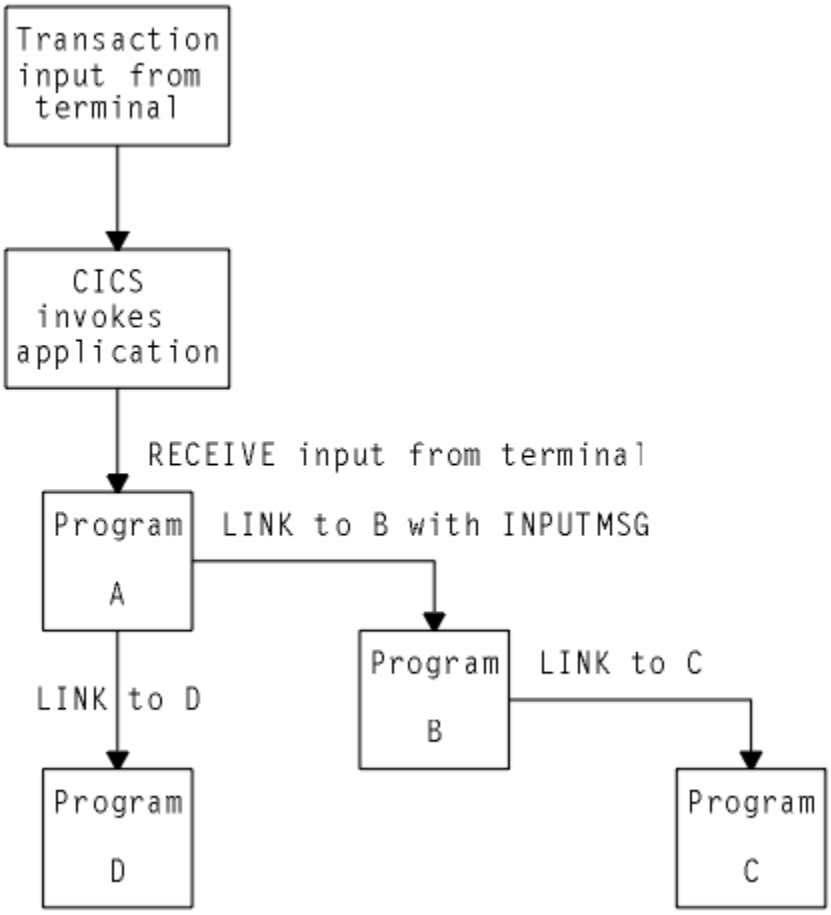


Figure 44. Use of INPUTMSG in a linked chain

Note:

1. In this example, the "real" first RECEIVE command is issued by program A. By linking to program B with the INPUTMSG option, it ensures that the next program to issue a RECEIVE request can also receive the terminal input. This can be either program B or program C.
2. If program A wants to pass on the unmodified terminal input that it received, it can name the same data area for the INPUTMSG option that it used for the RECEIVE command. For example:

```
EXEC CICS RECEIVE
INTO(TERMINAL-INPUT)

EXEC CICS LINK
PROGRAM(PROGRAMB)
INPUTMSG(TERMINAL-INPUT)
```

3. As soon as one program in a LINKchain issues a RECEIVE command, the INPUTMSG data ceases to be available to any subsequent RECEIVE command. In other words, in the example shown, if B issues a RECEIVE request before linking to C, the INPUTMSG data area is not available for C.
4. This method of communicating data from one program to another can be used for any data—it does not have to originate from a user terminal. In our example, program A could move any data into the named data area, and invoke program B with INPUTMSG referencing the data.
5. The "terminal-data" passed on INPUTMSG also ceases to be available when control is eventually returned to the program that issued the link with INPUTMSG. In our example, if C returns to B, and B returns to A, and neither B nor C issues a RECEIVE command, the data is assumed by A to have been received. If A then invokes another program (for example, D), the original INPUTMSG data is no longer available to D, unless the INPUTMSG option is specified.
6. The INPUTMSG data ceases to be available when a SEND or CONVERSE command is issued.

Using the INPUTMSG option on the RETURN command

You can specify INPUTMSG to pass data to the next transaction specified on a RETURN command with the TRANSID option. To do this, RETURN must be issued at the highest logical level to return control to CICS , and the command must also specify the IMMEDIATE option. If you specify INPUTMSG with TRANSID, and do not also specify IMMEDIATE, the next real input from the terminal overrides the INPUTMSG data, which is therefore lost.

If you specify INPUTMSG with TRANSID some time after a SEND command, the SEND message is immediately flushed out to the terminal.

The other use for INPUTMSG, on a RETURN command without the TRANSID option, is intended for use with a dynamic transaction routing program. See Writing a dynamic routing program for programming information about the user-replaceable dynamic transaction routing program.

Using mixed addressing modes

CICS supports the use of LINK, XCTL, and RETURN commands between programs with different addressing modes and between programs with the same addressing mode.

The following restrictions apply to programs passing data using a communication area named by the COMMAREA option:

- Addresses passed within a communication area to an AMODE(31) program must be 31 bits long. Do not use 3-byte addresses with flag data packed into the top byte, unless the called program is specifically designed to ignore the top byte.
- Addresses passed within a communication area to an AMODE(64) program can be 64 bits or 31 bits long. You must change any 31-bit addresses to 64-bit addresses before use.
- Addresses passed as data to an AMODE(24) program must be below the 16 MB line if they are to be accessed correctly by the called program.

These restrictions apply to the address of the communication area itself, and also to addresses within it. However, a communication area above 16 MB but below 2 GB (above the line) can be passed to an AMODE(24) subprogram. CICS copies the communication area into an area below 16 MB for processing. It copies it back again when control returns to the linking program.

CICS does not validate any data addresses passed within a communication area between programs with different addressing modes.

A COMMAREA cannot be in 64-bit storage.

Using LINK to pass data

These examples demonstrate how a LINK command causes data to be passed to the program being linked to. An XCTL command is coded in a similar way.

About this task

Figure 45 on page 153 to Figure 48 on page 154 show how a LINK command is used in COBOL, C, C++, PL/I, and assembler language.

These examples show data being passed in a COMMAREA. For an example of a LINK command that uses a channel to pass data, see “Transferring data between programs using channels” on page 108.

Ensure that the LENGTH value you specify in the LINK command matches the length of the data being passed in the COMMAREA. Do not specify 0 (zero) for LENGTH, because the resulting behavior is unpredictable and the EXEC CICS LINK command might fail.

When you use a COMMAREA to pass data, the program that is linked to must verify that the EIBCALEN field in the EIB of the task matches what the program expects. Discrepancies might result in storage violations or system failures. For more information, see “Passing data to other programs by using COMMAREA” on page 149.

```
 Invoking program
IDENTIFICATION DIVISION.
PROGRAM ID. 'PROG1'..
WORKING-STORAGE SECTION.
01 COM-REGION.
02 FIELD PICTURE X(3)..
PROCEDURE DIVISION.
MOVE 'ABC' TO FIELD.
EXEC CICS LINK PROGRAM('PROG2')
COMMAREA(COM-REGION)
LENGTH(3) END-EXEC..
Invoked program
IDENTIFICATION DIVISION.
PROGRAM-ID. 'PROG2'..
LINKAGE SECTION.
01 DFHCOMMAREA.
02 FIELD PICTURE X(3)..
PROCEDURE DIVISION.
IF EIBCALEN GREATER ZERO
THEN
IF FIELD EQUALS 'ABC' …
```

Figure 45. LINK command: COBOL example

In the following example, the COMMAREA contains a character string. For an example of a COMMAREA that contains a structure, see “Using RETURN to pass data” on page 154.

```
 Invoking program
main()
{
unsigned char field[3];
memcpy(field, "ABC", 3);
EXEC CICS LINK PROGRAM("PROG2")
COMMAREA(field)
LENGTH(sizeof(field));
}
Invoked program
main()
{
unsigned char *commarea;
EXEC CICS ADDRESS COMMAREA(commarea) EIB(dfheiptr);
if (dfheiptr->eibcalen > 0)
{
if (memcmp(commarea, "ABC", 3) == 0)
{.
```

Figure 46. LINK command: C example

```
 Invoking program
PROG1: PROC OPTIONS(MAIN);
DCL 1 COM_REGION AUTOMATIC,
2 FIELD CHAR(3),.
FIELD='ABC';
EXEC CICS LINK PROGRAM('PROG2')
COMMAREA(COM_REGION) LENGTH(3);
END;

Invoked program
PROG2:
PROC(COMM_REG_PTR) OPTIONS(MAIN);
DCL COMM_REG_PTR PTR;
DCL 1 COM_REGION BASED(COMM_REG_PTR),
2 FIELD CHAR(3),.
IF EIBCALEN>0 THEN DO;
IF FIELD='ABC' THEN ....
END;
END;
```

Figure 47. LINK command: PL/I example

```
 Invoking program
DFHEISTG DSECT
COMREG DS 0CL20
FIELD DS CL3.
PROG1 CSECT.
MVC FIELD,=C'XYZ'
EXEC CICS LINK
PROGRAM('PROG2')
COMMAREA(COMREG) LENGTH(3).
END
Invoked program
COMREG DSECT
FIELD DS CL3.
PROG2 CSECT.
L COMPTR,DFHEICAP
USING COMREG,COMPTR
CLC FIELD,=C'ABC'

END
```

Figure 48. LINK command: assembler language example

Using RETURN to pass data

These examples demonstrate how a RETURN command is used to pass data to a new transaction.

About this task

Figures Figure 49 on page 154 to Figure 52 on page 156 show how a RETURN command is used in COBOL, C, C++, PL/I, and assembler language.

These examples show data being returned in a COMMAREA. For an example of a RETURN command that uses a channel to return data, see “Transferring data between programs using channels” on page 108.

```
 Invoking program
IDENTIFICATION DIVISION.
PROGRAM-ID. 'PROG1'..
WORKING-STORAGE SECTION.
01 TERMINAL-STORAGE.
02 FIELD PICTURE X(3).
02 DATAFLD PICTURE X(17)..
PROCEDURE DIVISION.
MOVE 'XYZ' TO FIELD.
EXEC CICS RETURN TRANSID('TRN2')
COMMAREA(TERMINAL-STORAGE)
LENGTH(20) END-EXEC..
Invoked program
IDENTIFICATION DIVISION.
PROGRAM-ID. 'PROG2'.
LINKAGE SECTION.
01 DFHCOMMAREA.
02 FIELD PICTURE X(3).
02 DATAFLD PICTURE X(17)..
PROCEDURE DIVISION.
IF EIBCALEN GREATER ZERO
THEN
IF FIELD EQUALS 'XYZ'
MOVE 'ABC' TO FIELD.
EXEC CICS RETURN END-EXEC.
```

Figure 49. RETURN command: COBOL example

```
 Invoking program
struct ter_struct
{
unsigned char field[3];
unsigned char datafld[17];
};
main()
{
struct ter_struct ter_stor;
memcpy(ter_stor.field,"XYZ",3);
EXEC CICS RETURN TRANSID("TRN2")
COMMAREA(&ter_stor)
LENGTH(sizeof(ter_stor));
}
Invoked program
struct term_struct
{
unsigned char field[3];
unsigned char datafld[17];
};
main()
{
struct term_struct *commarea;
EXEC CICS ADDRESS COMMAREA(commarea) EIB(dfheiptr);
if (dfheiptr->eibcalen > 0)
{
if (memcmp(commarea->field, "XYZ", 3) == 0)
memcpy(commarea->field, "ABC", 3);
}
EXEC CICS RETURN;
}
```

Figure 50. RETURN command: C example

```
 Invoking program
PROG1: PROC OPTIONS(MAIN);
DCL 1 TERM_STORAGE,
2 FIELD CHAR(3),.
FIELD='XYZ';
EXEC CICS RETURN TRANSID('TRN2')
COMMAREA(TERM_STORAGE);
END;
Invoked program
PROG2:
PROC(TERM_STG_PTR) OPTIONS(MAIN);
DCL TERM_STG_PTR PTR;
DCL 1 TERM_STORAGE
BASED(TERM_STG_PTR),
2 FIELD CHAR(3),.
IF EIBCALEN>0 THEN DO;
IF FIELD='XYZ' THEN FIELD='ABC';
END;
EXEC CICS RETURN;
END;
```

Figure 51. RETURN command: PL/I example

```
 Invoking program

DFHEISTG DSECT
TERMSTG DS 0CL20
FIELD DS CL3
DATAFLD DS CL17
:
PROG1 CSECT
:
MVC FIELD,=C'ABC'
EXEC CICS RETURN
TRANSID('TRN2')
COMMAREA(TERMSTG)
:
END

Invoked program

TERMSTG DSECT
FIELD DS CL3
DATAFLD DS CL17
:
PROG2 CSECT
:
CLC EIBCALEN,=H'0'
BNH LABEL2
L COMPTR,DFHEICAP
USING TERMSTG,COMPTR
CLC FIELD,=C'XYZ'
BNE LABEL1
MVC FIELD,=C'ABC'
LABEL1 DS 0H
:
LABEL2 DS 0H
:
END
```

Figure 52. RETURN command: assembler language example

Affinity

CICS transactions and programs use many different techniques to pass data from one to another. Some of these techniques require that the transactions or programs exchanging data must execute in the same CICS region. This imposes restrictions on the regions to which transactions and distributed program link (DPL) requests can be dynamically routed and is said to create an affinity between them.

Java

This guidance on affinity between transactions describes applications written using the EXEC CICS API. However, many of the comments are equally valid for Java applications executing in a CICSplex. For guidance on developing Java applications, see Java development using JCICS.

Transactions, program-link requests, EXEC CICS START requests, and CICS business transaction services (BTS) activities can all be dynamically routed.

You can use a *dynamic* routing program to dynamically route:

- Transactions started from terminals
- Transactions started by eligible terminal-related EXEC CICS START commands
- Eligible CICS-to-CICS DPL requests
- Eligible program-link requests received from outside CICS.

You can use a *distributed* routing program to dynamically route:

- Eligible BTS processes and activities.
- Eligible non-terminal-related EXEC CICS START requests.

For detailed introductory information about dynamic and distributed routing, see Introduction to CICS dynamic routing .

Important:

The following sections talk exclusively about affinities between transactions.

- Affinities can also exist between programs. (Although, strictly speaking, we could say that it is the transactions associated with the programs that have the affinity.) This may impose restrictions on the regions to which program-link requests can be routed.
- The sections on safe, unsafe, and suspect programming techniques apply to the routing of program-link and START requests, as well as to the routing of transactions.

Types of affinity

Two types of affinity affect dynamic routing; inter-transaction affinity and transaction-system affinity. Inter-transaction affinity occurs among a set of transactions that share a common resource or coordinate their processing. Transaction-system affinity is an affinity between a transaction and a particular CICS region, where the transaction interrogates or changes the properties of that CICS region.

Inter-transaction affinity

Transaction affinity among two or more CICS transactions is caused by the transactions using techniques to pass information between one another, or to synchronize activity between one another, in a way that requires the transactions to execute in the same CICS region.

This type of affinity can occur in the following circumstances:

- One transaction terminates, leaving 'state data' in a place that a second transaction can only access by running in the same CICS region as the first transaction.
- One transaction creates data that a second transaction accesses while the first transaction is still running. For this to work safely, the first transaction typically waits on some event, which the second transaction posts when it has read the data created by the first transaction. This technique requires that both transactions are routed to the same CICS region.
- Two transactions synchronize, using an event control block (ECB) mechanism. Because CICS has no function shipping support for this technique, this type of affinity means that the two transactions must be routed to the same CICS region.

  Note: The same is true if two transactions synchronize, using an enqueue (ENQ) mechanism, **unless** you have used appropriate ENQMODEL resource definitions to give sysplex-wide scope to the ENQs. See ENQMODEL resource definitions .

Transaction-system affinity

Transactions with affinity to a particular system, rather than another transaction, are not eligible for dynamic routing. In general, they are transactions that use INQUIRE and SET commands, or have some dependency on global user exit programs, which also have an affinity with a particular CICS region.

Using INQUIRE and SET commands and global user exits

There is no remote (that is, function shipping) support for INQUIRE and SET commands, nor is there a SYSID option on them, hence transactions using these commands must be routed to the CICS regions that own the resources to which they refer. In general, such transactions cannot be dynamically routed to any target region, and therefore transactions that use INQUIRE and SET should be statically routed.

Global user exits running in different CICS regions cannot exchange data. It is unlikely that user transactions pass data or parameters with user exits, but if such transactions do exist, they must run in the same target region as the global user exits.

Programming techniques and affinity

In terms of inter-transaction affinity, the programming techniques used by your application programs in a dynamic or distributed routing environment, can be classed as safe, unsafe, or suspect.

- Safe techniques do not cause inter-transaction affinities.
- Unsafe techniques cause inter-transaction affinities inherently.
- Suspect techniques might, or might not create affinities depending on exactly how they are implemented.

Safe techniques

Safe programming techniques avoid the need for affinities. The programming techniques in the safe category are:

- The use of the communication area (COMMAREA), supported by the CICS API on a number of CICS commands. However, it is the COMMAREA option on the CICS RETURN command only that is of interest in a dynamic or distributed routing environment regarding transaction affinity, because it is the COMMAREA on a RETURN command that is passed to the next transaction in a pseudoconversational transaction.
- The use of a TCT user area (TCTUA) that is optionally available for each terminal defined to CICS.
- Synchronization or serialization of tasks using CICS commands:
  - ENQ / DEQ, provided that you have used appropriate ENQMODEL resource definitions to give sysplex-wide scope to the ENQs. See "Using ENQ and DEQ commands with ENQMODEL resource definitions" on page 162 and ENQMODEL resources for a description of ENQMODELs.
- The use of containers to pass data between CICS Business Transaction Services (BTS) activities. Container data is saved in an RLS-enabled BTS VSAM file.

For more information about the COMMAREA and the TCTUA, see "Programming techniques that avoid affinity" on page 159.

Unsafe techniques

Unsafe programming techniques are techniques that can require the creation of affinities. The programming techniques in the unsafe category are:

- The use of long-life shared storage:
  - The common work area (CWA)
  - GETMAIN SHARED storage
  - Storage obtained by a LOAD PROGRAM HOLD
- The use of task-lifetime local storage shared by synchronized tasks

  It is possible for one task to pass the address of some task-lifetime storage to another task.

  It may be safe for the receiving task to use the passed address, provided the owning task does not terminate. It is possible, but ill-advised, to use a CICS task-synchronization technique to allow the receiving task to prevent the sending task from terminating (or freeing the storage in some other way) before the receiver has finished with the address. However, such designs are not robust because there is a danger of the sending task being purged by some outside agency.

  See "Sharing task-lifetime storage" on page 166 for more details.
- Synchronization or serialization of tasks using CICS commands:
  - WAIT EVENT / WAIT EXTERNAL / WAITCICS
  - ENQ / DEQ, **unless** you have used appropriate ENQMODEL resource definitions to give sysplex-wide scope to the ENQs. See "Using ENQ and DEQ commands with ENQMODEL resource definitions" on page 162 and ENQMODEL resources for a description of ENQMODELs.

For more information about unsafe programming techniques, see “Programming techniques that create affinities” on page 163.

Suspect techniques

Some programming techniques might, or might not, create affinity depending on exactly how they are implemented. A good example is the use of temporary storage. Application programs using techniques in this category must be checked to determine whether they work without restrictions in a dynamic or distributed routing environment. The programming techniques in the suspect category are:

- The use of temporary storage queues with restrictive naming conventions
- Transient data queues and the use of trigger levels
- Synchronization or serialization of tasks using CICS commands:
  - RETRIEVE WAIT / START
  - START / CANCEL REQID
  - DELAY / CANCEL REQID
  - POST / CANCEL REQID
- INQUIRE and SET commands and global user exits

For more information about suspect programming techniques, see “Programming techniques that might create affinities” on page 169.

Recommendations

The best way to deal with inter-transaction affinity is to avoid creating inter-transaction affinity in the first place. Where it is not possible to avoid affinities, you should:

- Make the inter-transaction affinity easily recognizable, by using appropriate naming conventions, and
- Keep the lifetime of the affinities as short as possible.

Even if you could avoid inter-transaction affinities by changing your application programs, this is not necessary provided you include logic in your dynamic and distributed routing programs to cope with the affinities. Finally, you can statically route the affected transactions.

Programming techniques that avoid affinity

Some techniques for passing data between transactions are generally safe in that they do not create inter-transaction affinity. These involve the use of a communication area (COMMAREA), a terminal control table user area (TCTUA), or BTS containers.

To avoid creating affinities, do not use COMMAREAs, TCTUAs, and BTS containers to contain addresses. Generally the storage referenced by such addresses would have to be long-life storage, the use of which is an unsafe programming technique in a dynamic transaction routing environment.

The COMMAREA

The use of the COMMAREA option on the RETURN command is the principal example of a safe programming technique that you can use to pass data between successive transactions in a CICS pseudoconversational transaction.

CICS treats the COMMAREA as a special form of user storage, even though it is CICS that issues the GETMAIN and FREEMAIN requests for the storage, and not the application program.

CICS ensures that the contents of the COMMAREA specified on a RETURN command are always made available to the first program in the next transaction. This is true even when the sending and receiving transactions execute in different target regions. In a pseudoconversation, regardless of the target region to which a dynamic routing program chooses to route the next transaction, CICS ensures the COMMAREA

specified on the previous RETURN command is made available in the target region. This is illustrated in Figure 53 on page 160.

Some general characteristics of a COMMAREA are:

- Processor overhead is low.
- It is not recoverable.
- The length of a COMMAREA on a RETURN command can vary from transaction to transaction, up to a theoretical upper limit of 32 763 bytes. (However to be safe, you should not exceed 24 KB (1 KB = 1024 bytes), as recommended in the Application Programming Reference manual, because of a number of factors that can reduce the limit from the theoretical maximum.)
- CICS holds a COMMAREA in CICS main storage until the terminal user responds with the next transaction. This can be an important consideration if you are using large COMMAREAs, because the number of COMMAREAs held by CICS relates to terminal usage, and not to the maximum number of tasks in a region at any one time.
- A COMMAREA is available only to the first program in the next transaction, unless that program explicitly passes the data to another program or a succeeding transaction.

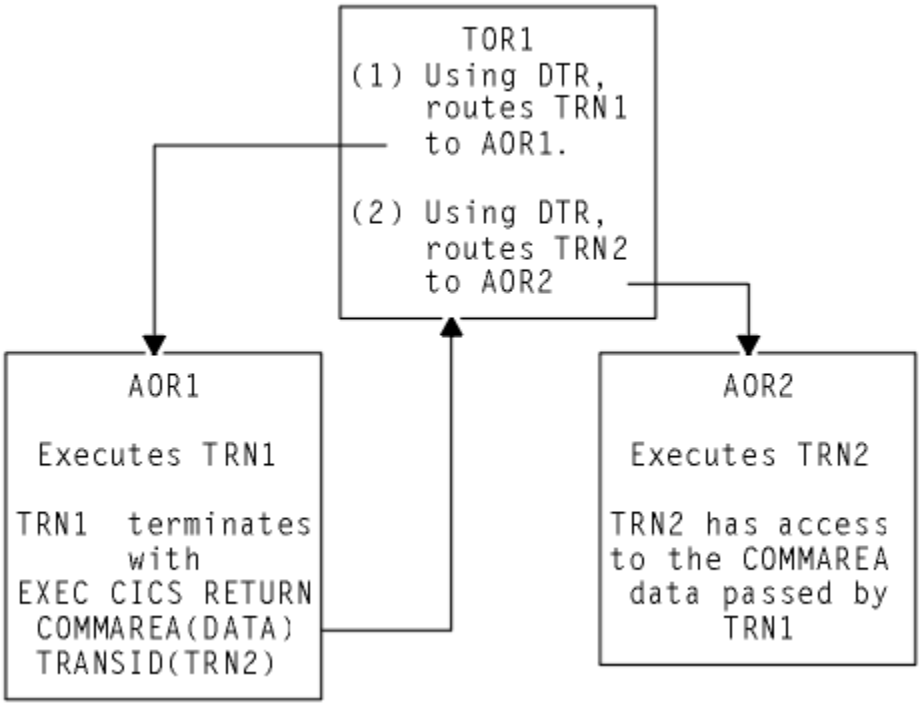


Figure 53. The use of a COMMAREA by a pseudoconversation in a dynamic transaction routing environment

The COMMAREA used in a pseudoconversational transaction, as shown in Figure 53 on page 160 , can be passed from transaction to transaction across a CICSplex, and, provided the COMMAREA contains only data and not addresses of storage areas, no inter-transaction affinity is created.

The TCTUA

The TCTUA is an optional extension to the terminal control table entry (TCTTE), each entry specifying whether the extension is present, and its length.

You specify that you want a TCTUA associated with a terminal by defining its length on the USERAREALEN attribute of a TYPETERM resource definition. This means that the TCTUAs are of fixed length for all the terminals created using the same TYPETERM definition.

A terminal control table user area (TCTUA) is safe to use in a dynamic transaction routing environment as a means of passing data between successive transactions in a pseudoconversational transaction. Like the COMMAREA, the TCTUA is always accessible to transactions initiated at a user terminal, even when the

transactions in a pseudoconversation are routed to different target regions. This is illustrated in Figure 54 on page 161 . Some other general characteristics of TCTUAs are:

- Minimal processor overhead (only one CICS command is needed to obtain the address).
- It is not recoverable.
- The length is fixed for the group of terminals associated with a given TYPETERM definition. It is suitable only for small amounts of data, the maximum size allowed being 255 bytes.
- If the terminal is autoinstalled, the TCTUA lasts as long as the TCTTE, the retention of which is determined by the AILDELAY system initialization parameter. The TCTTE, and therefore any associated TCTUA, is deleted when the AILDELAY interval expires after a session between CICS and a terminal is ended.

  If the terminal is defined to CICS by an explicit terminal definition, the TCTTE and its associated TCTUA are created when the terminal is installed and remain until the next initial or cold start of CICS.

The TCTUA is available to a dynamic routing environment in the routing region as well as application programs in the target region. It can be used store information relating to the dynamic routing of a transaction. For example, you can use the TCTUA to store the name of the selected target region to which a transaction is routed.

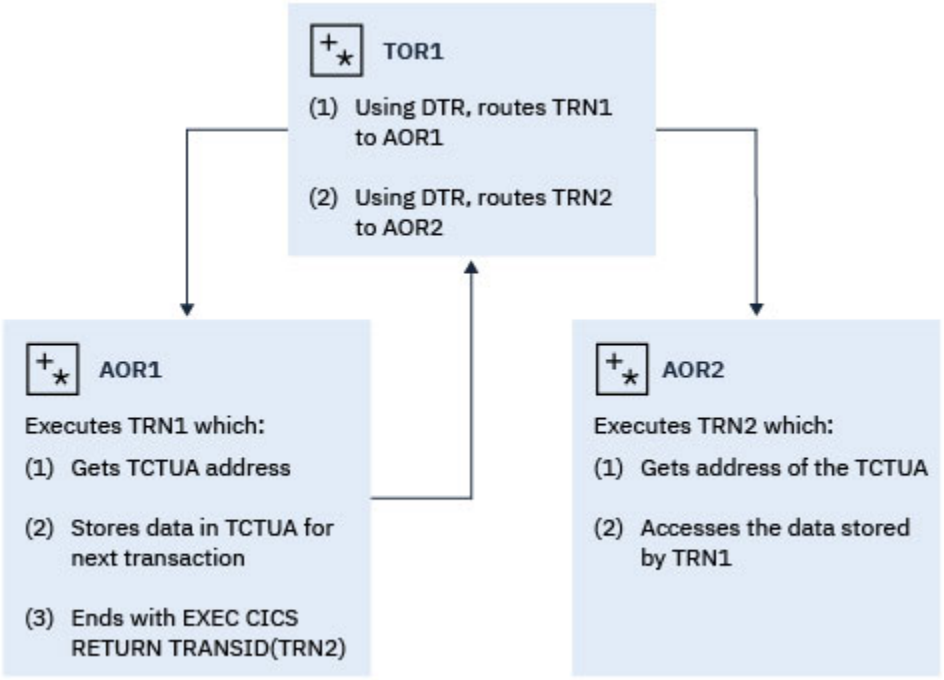


Figure 54. The use of a TCTUA by a pseudoconversation in a dynamic routing environment

Using the TCTUA in an unsafe way

The EXEC CICS ADDRESS TCTUA(*ptr-ref*) provides direct addressability to the TCTUA, and this is how each task requiring access to a TCTUA should obtain the TCTUA address. If tasks attempt to pass the address of their TCTUAs in some other way, such as in a temporary storage queue, or to use the TCTUA itself to pass addresses of other storage areas, the TCTUA ceases to provide a safe programming technique for use in a dynamic transaction routing environment.

It is also possible for a task to obtain the TCTUA of a principal facility other than its own, by issuing an INQUIRE TERMINAL command that names the terminal associated with another task (the INQUIRE TERMINAL command returns the TCTUA address of the specified terminal). Using the TCTUA address of a terminal other than a task's own principal facility is another example an unsafe use of the TCTUA facility. Depending on the circumstances, particularly in a dynamic routing environment , the TCTUA of a terminal that is not the inquiring task's principal facility could be deleted after the address has been obtained. For example, in a target region, an INQUIRE TERMINAL command could return the TCTUA address associated

with a surrogate terminal that is running a dynamically routed transaction. If the next transaction from the terminal is routed to a different target region, the TCTUA address ceases to be valid.

Using ENQ and DEQ commands with ENQMODEL resource definitions

The ENQ and DEQ commands are used to serialize access to a shared resource. CICS uses ENQMODEL resources with z/OS global resource serialization to provide sysplex-wide protection of resources that are used by multiple applications.

Sysplex enqueue eliminates a significant cause of inter-transaction affinity and allows the single-threading and synchronization of tasks across the sysplex. It enables serialization of concurrent updates to shared temporary storage queues, and makes it possible to prevent interleaving of records written by concurrent tasks in different CICS regions to a remote transient data queue. Sysplex-wide enqueue is supported only for a resource, and not for an enqueue on an address. It is not designed for the locking of recoverable resources.

Local enqueues within a single CICS region are managed in the CICS address space. Sysplex-wide enqueues that affect more than one CICS region are managed by z/OS global resource serialization. For more information on global resource serialization, see z/OS MVS Planning: Global Resource Serialization . For information on configuring z/OS global resource serialization for CICS , see Global CICS enqueue and dequeue: improving performance .

When the **EXEC CICS ENQ** and **EXEC CICS DEQ** commands are issued for a resource, CICS checks for a matching installed ENQMODEL definition. The ENQSCOPE attribute of an ENQMODEL resource defines the set of regions that share the same enqueue scope. If the ENQSCOPE attribute remains blank (the default value), CICS treats any matching ENQ or DEQ request as local to the issuing CICS region. If the ENQSCOPE is non-blank, CICS treats the ENQ or DEQ as sysplex-wide, and passes a queue name and the resource name to z/OS global resource serialization to manage the enqueue. The CICS regions that need to use sysplex-wide enqueue or dequeue function must all have the required ENQMODEL resources defined and installed. For more information on ENQMODEL resources, see ENQMODEL resources.

Applications use sysplex enqueues when you define appropriate ENQMODEL resources; no change is needed to the application programs. For applications where the resource name is configured dynamically, so is not known in advance, you can use the enqueue EXEC interface program exits XNQEREQ and XNQEREQC to supply characters at the start of the resource name that match a suitable ENQMODEL resource definition. For more information on these user exits, see Enqueue EXEC interface program exits XNQEREQ and XNQEREQC.

BTS containers

A container is owned by a BTS activity and can be used to pass data between BTS activities or between different activations of the same activity.

Containers cannot be used outside of an activity; for more information, see the *CICS Business Transaction Services* manual . An activity uses GET and PUT container to update the container's contents. CICS ensures that the appropriate containers are available to an activity by saving all the information (including containers) associated with a BTS activity in an RLS-enabled VSAM file. For this reason, note that a BTS environment cannot extend outside a sysplex (see *CICS Business Transaction Services*), but you can use dynamic routing within a sysplex passing data in containers.

Some general characteristics of containers are:

- An activity can own any number of containers; you are not limited to one.
- There is no size restriction.
- They are recoverable.
- They exist in main storage only while the associated activity is executing. Otherwise they are held on disk. Therefore, you do not need to be overly concerned with their storage requirements, unlike terminal COMMAREAs.

Programming techniques that create affinities

Some CICS application programming techniques, notably those that pass, or obtain, addresses to shared storage, create an affinity between transactions.

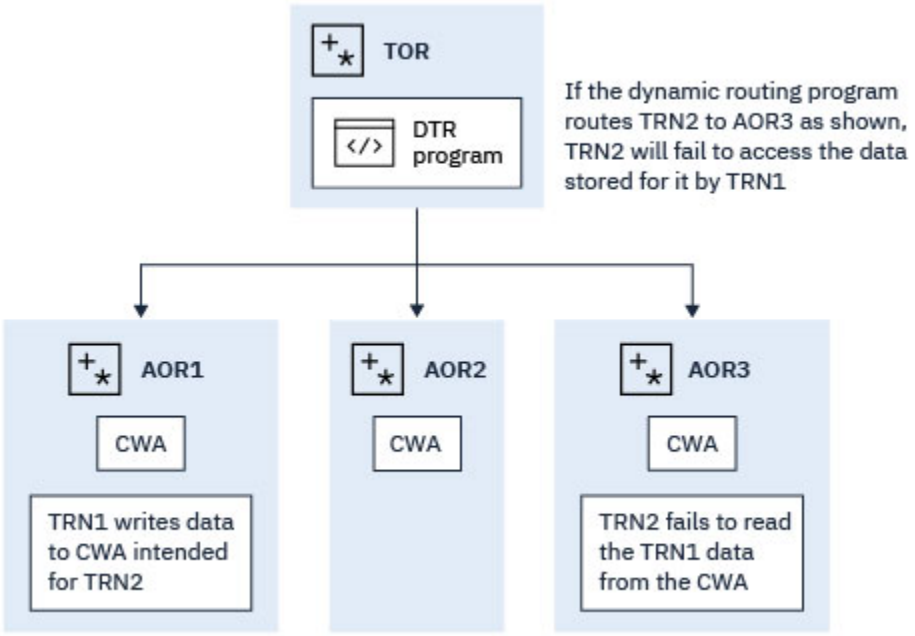
The programming techniques that are generally unsafe are described in the following sections.

- "Using the common work area" on page 163
- "Using GETMAIN SHARED storage" on page 164
- "Using the LOAD PROGRAM HOLD command" on page 165
- "Sharing task-lifetime storage" on page 166
- "Using the WAIT EVENT command" on page 167
- "Using ENQ and DEQ commands without ENQMODEL resource definitions" on page 168

Using the common work area

The CWA in a CICS region is created (optionally) during CICS initialization, exists until CICS terminates, and is not recovered on a CICS restart (warm or emergency). The ADDRESS CWA (*ptr-ref*) command provides direct addressability to the CWA.

A good example of how the use of long-life shared storage such as the CWA can create affinity is when one task stores data in the CWA, and a later task reads the data from it. Clearly, the task retrieving the data must run in the same target region as the task that stored the data, or it references a different storage area in a different address space. This restricts the workload balancing capability of the dynamic or distributed routing program, as shown in Figure 55 on page 163.



CWA

Figure 55. Illustration of inter-transaction affinity created by use of the CWA

However, if the CWA contains read-only data, and this data is replicated in more than one target region, it is possible to use the CWA and continue to have the full benefits of dynamic routing. For example, you can run a program during the post-initialization phase of CICS startup (a PLTPI program) that loads the CWA with read-only data in each of a number of selected target regions. In this way, all transactions routed to target regions loaded with the same CWA data have equal access to the data, regardless of which of the target regions to which the transactions are routed. With CICS subsystem storage protection, you can

ensure the read-only integrity of the CWA data by requesting the CWA from CICS-key storage, and define all the programs that read the CWA to execute in user key.

Using GETMAIN SHARED storage

Shared storage is allocated by a GETMAIN SHARED command, and remains allocated until explicitly freed by the same, or by a different, task.

Shared storage can be used to exchange data between any CICS tasks that run during the lifetime of the shared storage. Transactions designed in this way must execute in the same CICS region to work correctly. The dynamic or distributed routing program should ensure that transactions using shared storage are routed to the same target region.

Figure 56 on page 164 illustrates the use of shared storage.

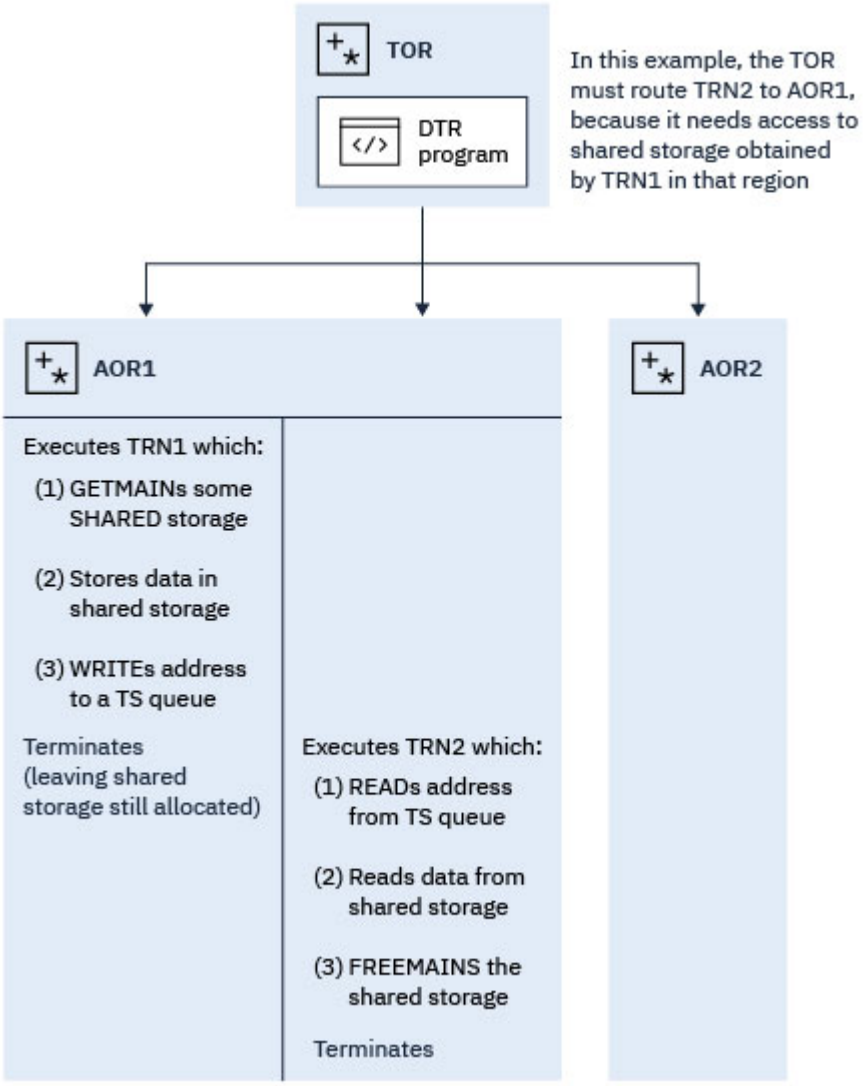


Figure 56. Illustration of inter-transaction affinity created by use of shared storage

If the two transactions shown in Figure 56 on page 164 are parts of a pseudoconversational transaction, the use of shared storage should be replaced by a COMMAREA (if the amount of storage fits within the COMMAREA size limits).

Using the LOAD PROGRAM HOLD command

A program (or table) that CICS loads in response to a LOAD PROGRAM HOLD command remains in directly addressable storage until explicitly released by the same, or by a different, task.

Any CICS tasks that run while the loaded program (table) is held in storage can use the loaded program's storage to exchange data, if:

- The program is not loaded into read-only storage, or
- The program is not defined to CICS with RELOAD(YES)

Although you could use a temporary storage queue to make the address of the loaded program's storage available to other tasks, the more typical method would be for other tasks to issue a LOAD PROGRAM command also, with the SET(*ptr_ref*) option so that CICS can return the address of the held program.

The nature of the affinity caused by the use of the LOAD PROGRAM HOLD command is identical to that caused by the use of GETMAIN SHARED storage (see Figure 56 on page 164 in "Using GETMAIN SHARED storage" on page 164 and Figure 57 on page 165 below), and the same rule applies: to preserve the application design, the dynamic or distributed routing program must ensure that all transactions that use the address of the loaded program (or table) are routed to the same target region.

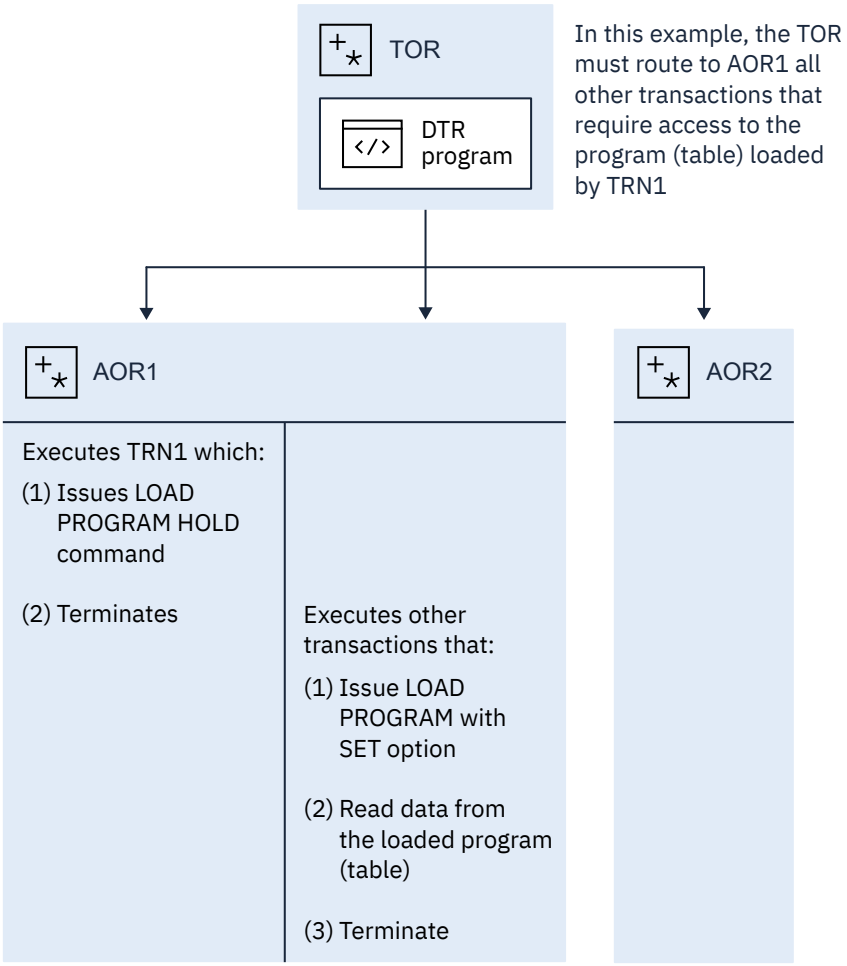


Figure 57. Illustration of inter-transaction affinity created by use of LOAD PROGRAM HOLD

Note: This rule applies also to programs defined with the RESIDENT option on the resource definition for the loaded program (whether the HOLD option is specified on the LOAD command). However, regardless of affinity considerations, it is unsafe to use the RESIDENT option to enable transactions to share data, because programs defined with RESIDENT are subject to SET PROGRAM(*program_name*) NEWCOPY commands, and can therefore be changed.

The rule also applies to a non-resident, non-held, loaded program where the communicating tasks are synchronized.

Sharing task-lifetime storage

The use of any task-lifetime storage belonging to one task can be shared with another task, provided the owning task can pass the address to the other task in the same CICS address space.

This technique creates an affinity among the communicating tasks, and requires that any task retrieving and using the passed address must execute in the same target region as the task owning the task-lifetime storage.

For example, it is possible to use a temporary storage queue to pass the address of a PL/I automatic variable, or the address of a COBOL working-storage structure (see Figure 58 on page 166 for an example).

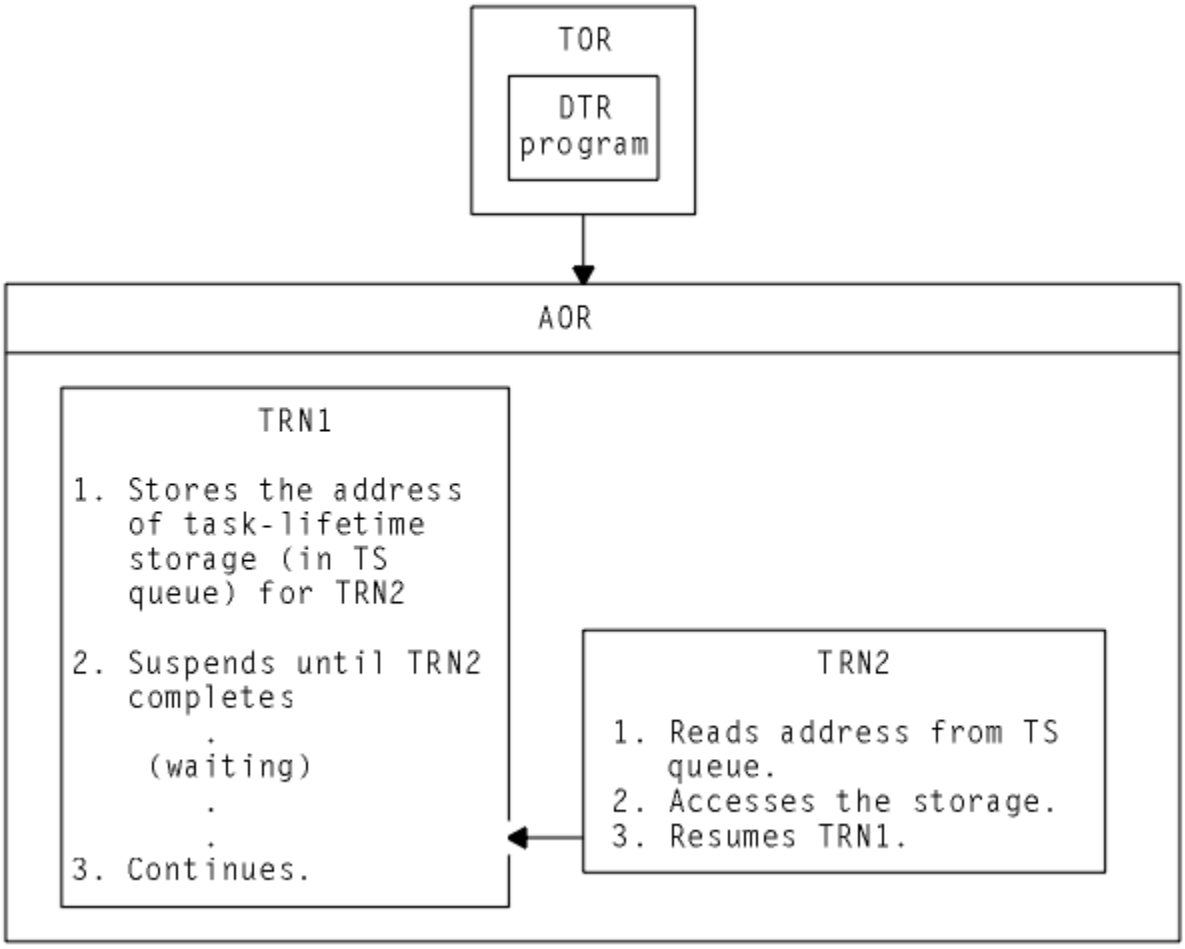


Figure 58. Illustration of inter-transaction affinity created by use of task-lifetime storage

For two tasks to share task-lifetime storage belonging to one of them requires that the tasks are synchronized in some way. See Table 22 on page 166 for commands that provide ways of suspending and resuming a task that passes the address of its local storage.

Table 22. Methods for suspending and resuming (synchronizing) tasks

Suspending operation	**Resuming operation**
WAIT EVENT, WAIT EXTERNAL, WAITCICS	POST
RETRIEVE WAIT	START
DELAY	CANCEL

Table 22. Methods for suspending and resuming (synchronizing) tasks (continued)	
Suspending operation	**Resuming operation**
POST	CANCEL
START	CANCEL
ENQ	DEQ

Some of these techniques themselves require that the transactions using them must execute in the same target region, and these are discussed later. However, even in those cases where tasks running in different target regions can be synchronized, it is not safe to pass the address of task-lifetime storage from one to the other. Even without dynamic routing, designs that are based on the synchronization techniques shown in Table 22 on page 166 are fundamentally unsafe because it is possible that the storage-owning task could be purged.

Note:

1. Using synchronization techniques, such as RETRIEVE WAIT/START, to allow sharing of task-lifetime storage is not recommended.
2. No inter-transaction affinity is caused in those cases where the task sharing another task's task-lifetime storage is started by an START command, except when the START command is function-shipped or routed to a remote system.

Using the WAIT EVENT command

The WAIT EVENT command is used to synchronize a task with the completion of an event performed by some other CICS or MVS task.

The completion of the event is signalled (posted) by the setting of a bit pattern into the event control block (ECB). Both the waiting task and the posting task must have direct addressability to the ECB, hence both tasks must execute in the same target region. The use of a temporary storage queue is one way that the waiting task can pass the address of the ECB to another task.

This synchronization technique is illustrated in Figure 59 on page 168.

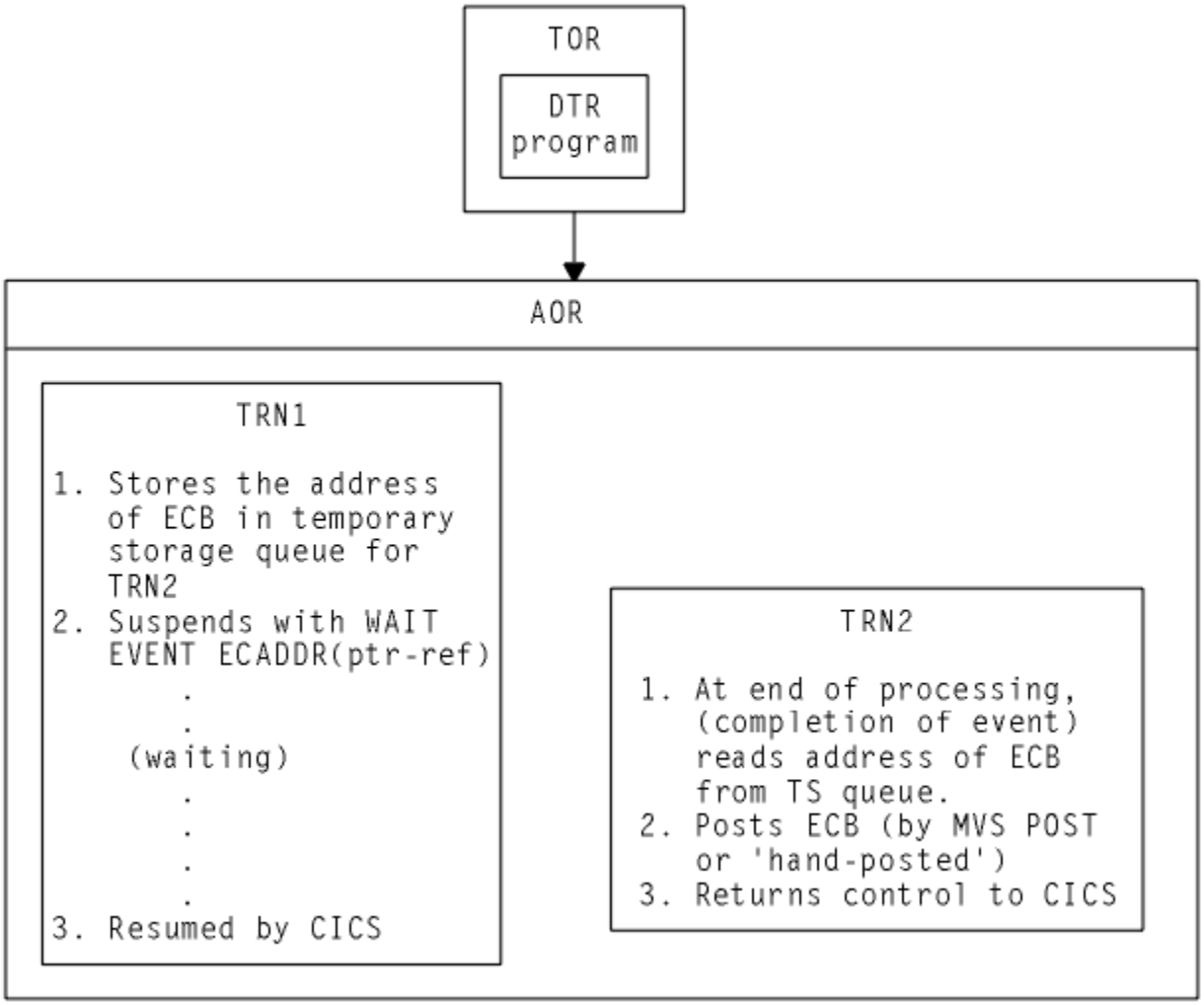


Figure 59. Illustration of inter-transaction affinity created by use of WAIT EXTERNAL command

If TRN2 shown in Figure 59 on page 168 executed in a different target region from TRN1, the value of *ptr-ref* would be invalid, the post operation would have unpredictable results, and the waiting task would never be resumed. For this reason, a dynamic or distributed routing program must ensure that a posting task executes in the same target region as the waiting task to preserve the application design. The same considerations apply to the use of WAIT EXTERNAL and WAITCICS commands for synchronizing tasks.

Using ENQ and DEQ commands without ENQMODEL resource definitions

The ENQ and DEQ commands are used to serialize access to a shared resource. These commands only work for CICS tasks running in the same region.

The ENQ and DEQ commands cannot be used to serialize access to a resource shared by tasks in different regions, unless they are supported by appropriate ENQMODEL resource definitions so that they have sysplex-wide scope. See “Using ENQ and DEQ commands with ENQMODEL resource definitions” on page 162 and ENQMODEL resources for a description of ENQMODELs.

Note that any ENQ that does not specify the LENGTH option is treated as an enqueue on an address and therefore has only local scope. The use of ENQ and DEQ for serialization (without ENQMODEL definitions to give sysplex-wide scope) is illustrated in Figure 60 on page 169.

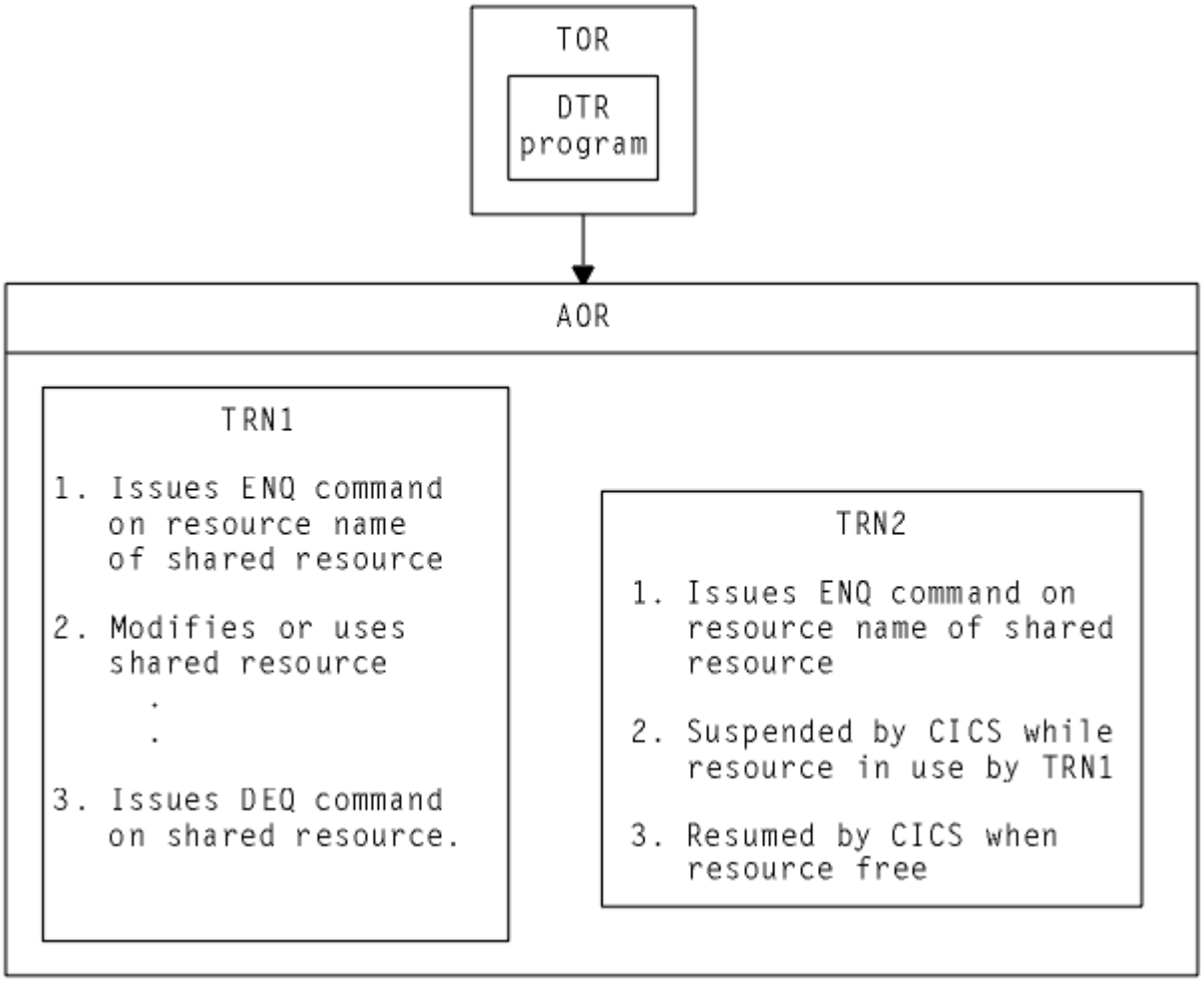


Figure 60. Illustration of inter-transaction affinity created by use of ENQ/DEQ commands

If TRN2 shown in Figure 60 on page 169 executed in a different target region from TRN1, TRN2 would not be suspended while TRN1 accessed the shared resource. For this reason, a dynamic or distributed routing program must ensure that all tasks that enqueue on a given resource name must execute in the same target region to preserve the application design. TRN2 would, of course, be serialized with other CICS tasks that issue ENQ commands on the same resource name in its target region.

Programming techniques that might create affinities

Some CICS application programming techniques might create an affinity between transactions depending on how they are implemented. You can adopt various approaches to avoid unwanted affinities.

- "Avoiding affinities when using temporary storage" on page 170
- "Using transient data" on page 172
- "Avoiding affinities when using the RETRIEVE WAIT and START commands" on page 173
- "Avoiding affinities when using the START and CANCEL REQID commands" on page 175
- "Avoiding affinities using the DELAY and CANCEL REQID commands" on page 176
- "Avoiding affinities using the POST and CANCEL REQID commands" on page 178

Avoiding affinities when using temporary storage

CICS application programs commonly use temporary storage queues to hold temporary application data, and to act as scratch pads.

Sometimes a temporary storage queue is used to pass data between application programs that execute under one instance of a transaction (for example, between programs that pass control by a LINK or XCTL command in a multi-program transaction). Such use of a temporary storage queue requires that the queue exists only for the lifetime of the transaction instance, and therefore it does not need to be shared between different target regions, because a transaction instance executes in one, and only one, target region.

Note: This latter statement is not strictly true in the case of a multi-program transaction, where one of the programs is linked by a distributed program link command and the linked-to program resides in a remote system. In this case, the program linked by a DPL command runs under another CICS task in the remote region. The preferred method for passing data to a DPL program is by a channel or COMMAREA, but if a temporary storage queue is used for passing data in a DPL application, the queue must be shared between the two regions.

Sometimes a temporary storage queue holds information that is specific to the target region, or holds read-only data. In this case the temporary storage queue can be replicated in each target region, and no sharing of data between target regions is necessary.

However, in many cases a temporary storage queue is used to pass data between transactions, in which case the queue must be globally accessible to enable the transactions using the queue to run in any dynamically selected target region. To make a temporary storage queue globally accessible while avoiding inter-transaction affinities, you can either use remote queues in a queue-owning region (QOR), or use shared temporary storage queues that reside in a temporary storage data-sharing pool in a coupling facility.

Remote temporary storage queues in a queue-owning region

To function ship temporary storage requests to a queue-owning region (QOR):

- In the application-owning regions, create TSMODEL resources using the REMOTESYSTEM and REMOTEPREFIX (or XREMOTEPFX) attributes to define the temporary storage queues as remote.
- In the queue-owning region, create corresponding TSMODEL resources to define the characteristics of the temporary storage queues.
- If you want to avoid duplication, you can create TSMODEL resources that can be installed in both a local and remote region, and share these resources between the application-owning regions and the queue-owning region.

If your temporary storage queue names are generated dynamically, you must follow a process that generates queue names that are unique for a given terminal. The usual convention is a 4-character prefix (for example, the transaction identifier) followed by a 4-character terminal identifier as the suffix. Generic queue names in this format can be defined as remote queues. If your naming convention does not conform to this rule, the queue cannot be defined as remote, and all transactions that access the queue must be routed to the target region where the queue was created.

You cannot use the global user exits for temporary storage requests to change a temporary storage queue name from a local to a remote queue name.

Figure 61 on page 171 illustrates the use of a remote queue-owning region.

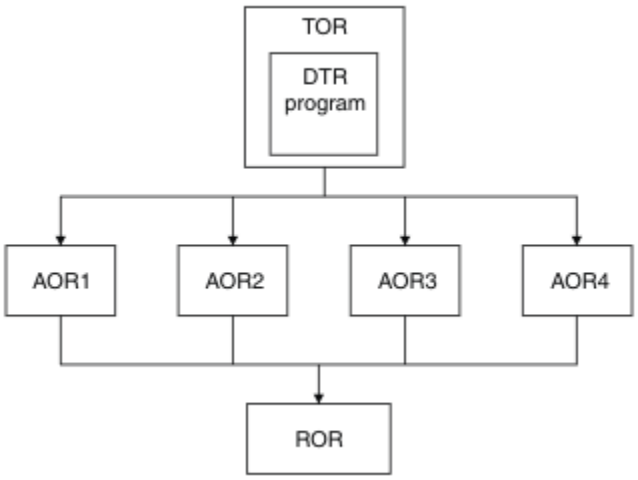


Figure 61. Using remote queues to avoid inter-transaction affinity relating to temporary storage

Shared temporary storage queues

To use shared temporary storage queues that reside in a temporary storage data-sharing pool in a coupling facility, in the application-owning regions, create TSMODEL resources that use the POOLNAME and PREFIX (or XPREFIX) attributes to map the temporary storage requests to a named pool of shared temporary storage queues.

You must set up a temporary storage server to support each pool of temporary storage queues that is defined in the coupling facility. For more information, see Setting up and running a temporary storage server.

Note that TSMODEL resource definitions do not support applications specifying an explicit SYSID for a queue that resides in a temporary storage data sharing pool. Applications must use the QUEUE or QNAME option instead. If your applications use an explicit SYSID for a shared queue pool, this requires the support of a temporary storage table (TST).

Figure 62 on page 172 shows four AORs that are using the same pool of shared temporary storage queues in a coupling facility. A temporary storage server is always managed by the queue server region program, DFHXQMN.

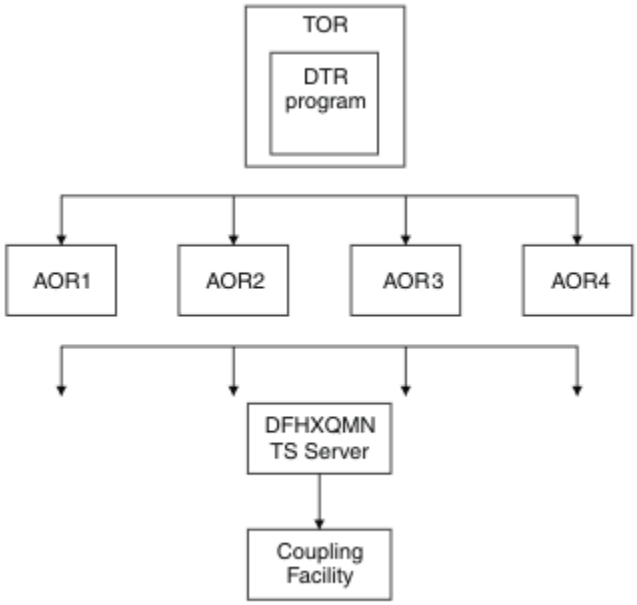


Figure 62. Example of the use of a temporary storage data-sharing server

Exception conditions for globally accessible queues

When you eliminate inter-transaction affinity relating to TS queues by the use of a global QOR, you must also take care to review exception condition handling. Some exception conditions can occur that previously were not possible when the transactions and the queue were local in the same region.

This situation arises because the target region and QOR can fail independently, causing circumstances where:

- The queue already exists, because only the target region failed while the QOR continued.
- The queue is not found, because only the QOR failed while the target region continued.

Using transient data

CICS application programs commonly use transient data queues (TD). To enable transactions that use a TD queue that must be shared, to be dynamically routed to a target region, you must ensure that the TD queues are globally accessible.

The dynamic transaction routing considerations for TD queues have much in common with those for temporary storage.

All transient data queues must be defined to CICS, and must be installed before the resources become available to a CICS region. These definitions can be changed to support a remote transient data queue-owning region (QOR).

However, there is a restriction for TD queues that use the trigger function. The transaction to be invoked when the trigger level is reached must be defined as a local transaction in the region where the queue resides (in the QOR). Thus the trigger transaction must execute in the QOR. However, any terminal associated with the queue need not be defined as a local terminal in the QOR. This does not create an inter-transaction affinity.

Figure 63 on page 173 illustrates the use of a remote transient data queue-owning region.

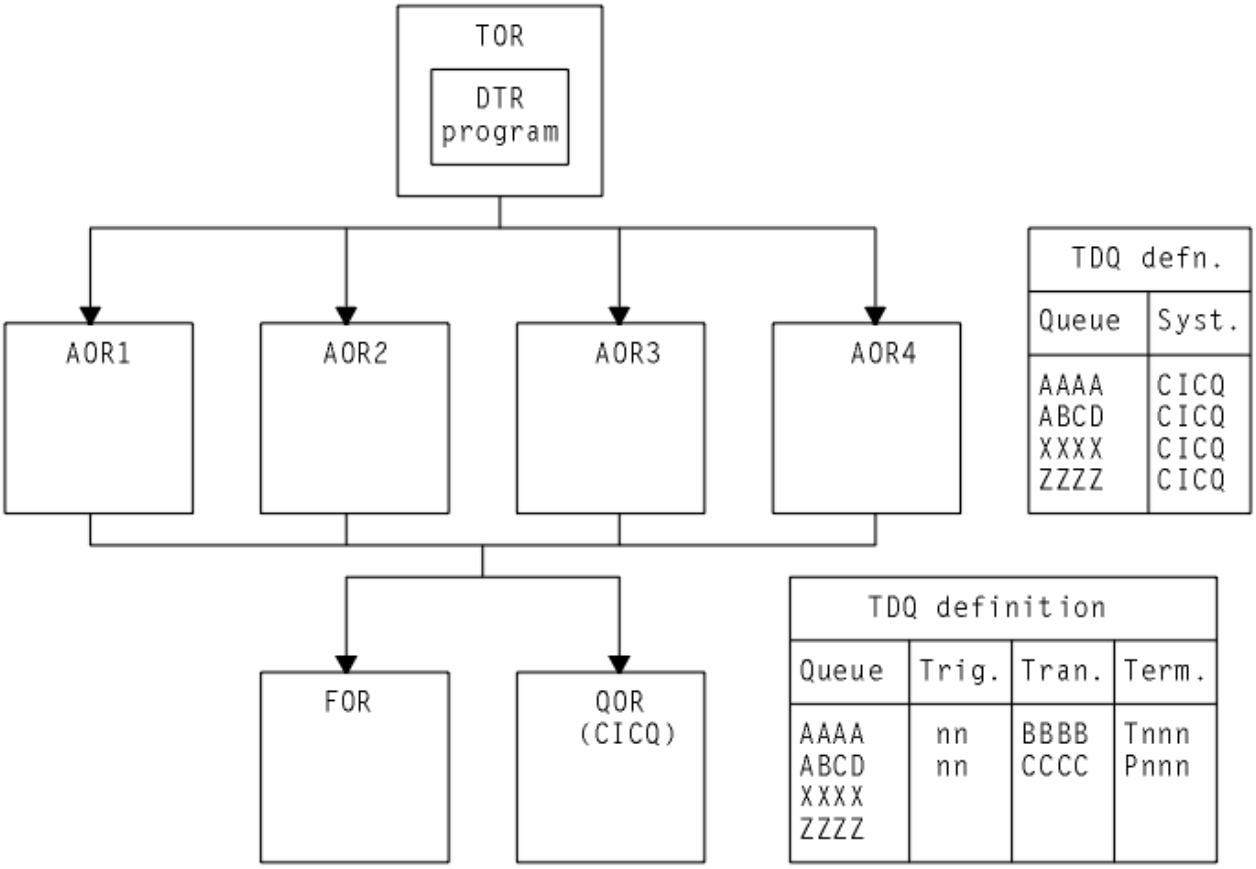


Figure 63. Using remote queues to avoid inter-transaction affinity relating to transient data

Exception conditions for globally accessible queues

When you eliminate inter-transaction affinity relating to TD queues by the use of a global QOR, there should not be any new exception conditions (other than SYSIDERR if there is a system definition error or failure).

Avoiding affinities when using the RETRIEVE WAIT and START commands

The use of some synchronization techniques permit the sharing of task-lifetime storage between two synchronized tasks. Use the RETRIEVE WAIT and START commands for this purpose.

In the example illustrated in Figure 64 on page 174 , TRN1 is designed to retrieve data from an asynchronous task, TRN2, and therefore must wait until TRN2 makes the data available. Note that for this mechanism to work, TRN1 must be a terminal-related transaction.

The steps are as follows:

1. TRN1 writes data to a TS queue, containing its TRANSID and TERMID.
2. To cause itself to suspend, TRN1 issues a RETRIEVE WAIT command, which causes CICS to suspend the task until the RETRIEVE can be satisfied, which is not until TRN2 issues a START command with data passed by the FROM parameter.
3. However, TRN2 can only issue a START command to resume TRN1 if it knows the TRANSID and TERMID of the suspended task (TRN1 in our example). Thus it reads the TS queue to obtain the information written by TRN1. Using a temporary storage queue is one way that this information can be passed by the suspending task.
4. Using the information from the TS queue, TRN2 issues the START command for TRN1, causing CICS to resume TRN1 by satisfying the outstanding RETRIEVE WAIT.

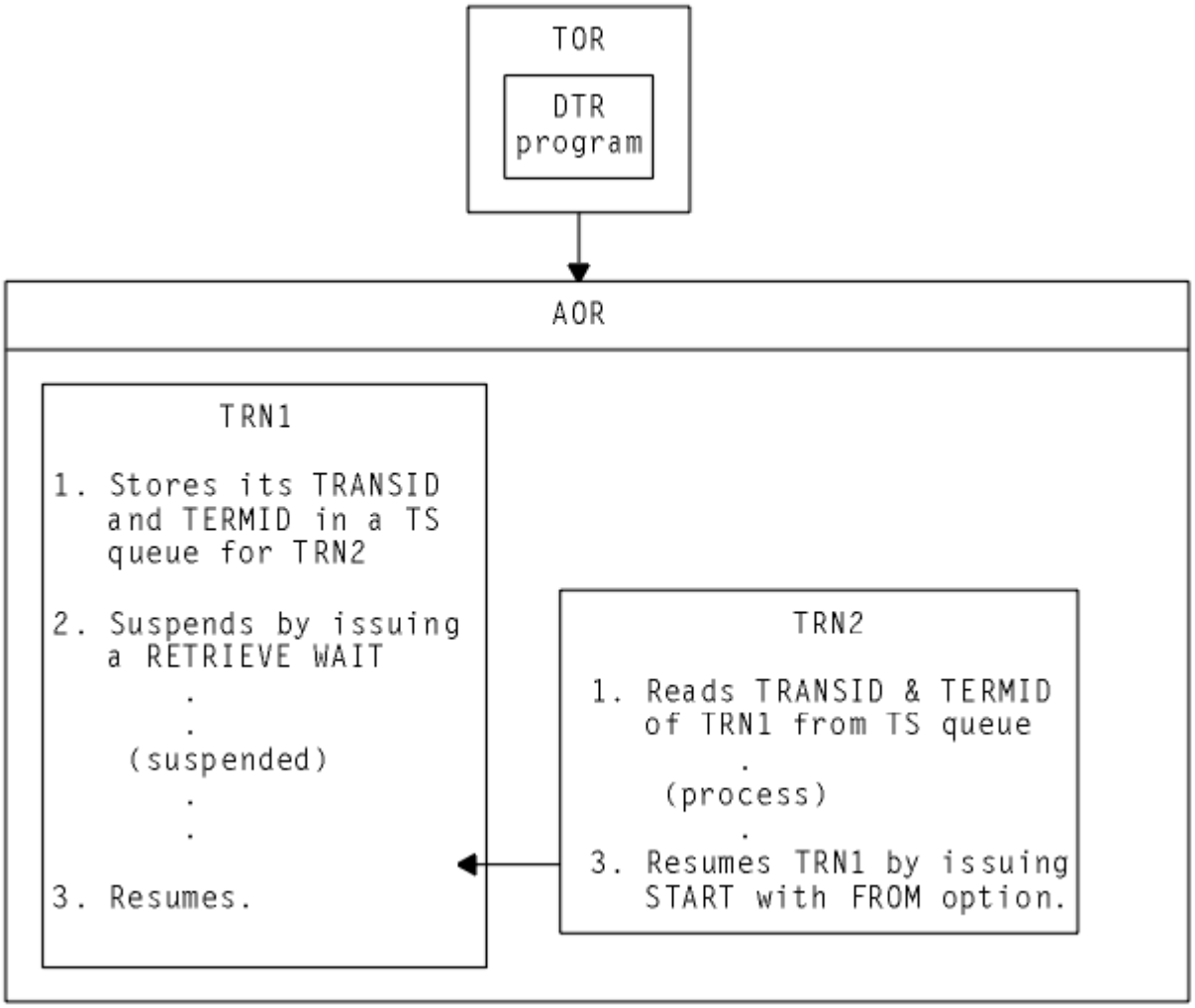


Figure 64. Illustration of task synchronization using RETRIEVE WAIT and START commands

In the example of task synchronization using RETRIEVE WAIT and START commands shown in Figure 64 on page 174 , the START command that satisfies the RETRIEVE WAIT must:

- Be issued in same target region as the transaction (TRN1 in our example) issuing the RETRIEVE WAIT command, or
- Specify the SYSID of the target region where the RETRIEVE WAIT command was executed, or
- Specify a TRANSID (TRN1 in our example) that is defined as a remote transaction residing on the target region that executed the RETRIEVE WAIT command.

An application design based on the remote TRANSID technique only works for two target regions. An application design using the SYSID option on the START command only works for multiple target regions if all target regions have connections to all other target regions (which may not be desirable). In either case, the application programs must be modified: there is no acceptable way to use this programming technique in a dynamic or distributed routing program except by imposing restrictions on the routing program. In general, this means that the dynamic or distributed routing program must ensure that TRN2 must execute in the same region as TRN1 to preserve the application design.

Avoiding affinities when using the START and CANCEL REQID commands

The CICS START command is used by a transaction to start another transaction. Another transaction can cancel this command using the CANCEL command and specifying the REQID associated with the START command.

With this CICS application programming technique, one transaction issues a START command to start another transaction after a specified interval. Another transaction (not the one requested on the START command) determines that it is no longer necessary to run the requested transaction (which is identified by the REQID parameter) and cancels the START request.

A temporary storage queue is one way that the REQID can be passed from task to task.

Note: The CANCEL command is effective only if the specified interval has not yet expired. To use this technique, the CANCEL command must specify the REQID option, but the START command need not. This is because, provided the NOCHECK option is not specified on the START command, CICS generates a REQID for the request and stores it in the EXEC interface block (EIB) in field EIBREQID.

Figure 65 on page 175 illustrates this programming technique.

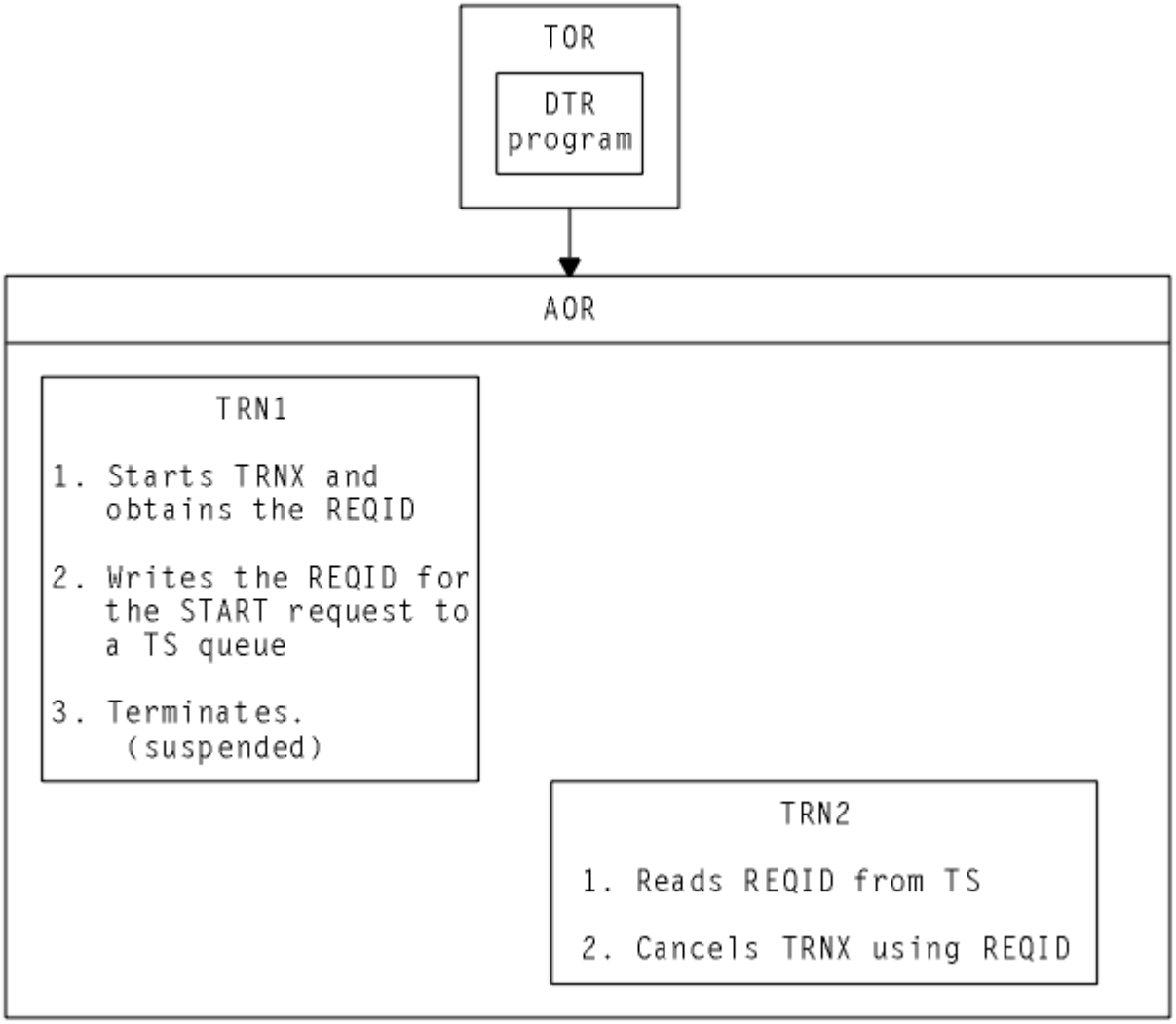


Figure 65. Illustration of the use of the START and CANCEL REQID commands

Using this application programming technique, the CANCEL command that cancels the START request must:

- Be issued in same target region that the START command was executed in, or
- Specify the SYSID of the target region where the START command was executed, or

- Specify a TRANSID (TRNX in our example) that is defined as a remote transaction residing on the target region where the START command was executed.

Note: A START command is not necessarily executed in the same region as the application program issuing the command. It can be function shipped (or, in the case of a non-terminal-related START, routed) and executed in a different CICS region. The previous rules apply to the region where the START command is finally executed.

An application design based on the remote TRANSID technique only works for two target regions. An application design using the SYSID option on the cancel command only works for multiple target regions if all target regions have connections to all other target regions. In either case, the application programs need to be modified: there is no acceptable way to use this programming technique in a dynamic or distributed routing program except by imposing restrictions on the routing program.

In general, this means that the dynamic or distributed routing program has to ensure that TRN2 executes in the same region as TRN1 to preserve the application design, and also that TRNX is defined as a local transaction in the same region.

Avoiding affinities using the DELAY and CANCEL REQID commands

The CICS DELAY command is used to suspend a task. Another task can cancel this command using the CANCEL command and specifying the REQID associated with the DELAY command.

With this CICS application programming technique, one task can resume another task that has been suspended by a DELAY command.

A DELAY request can be canceled only by a different task from the one issuing the DELAY command, and the CANCEL command must specify the REQID associated with DELAY command. Both the DELAY and CANCEL command must specify the REQID option to use this technique.

The steps involved in this technique using a temporary storage queue to pass the REQID are as follows:

1. A task (TRN1) writes a predefined DELAY REQID to a TS queue. For example:

   ```
   EXEC CICS WRITEQ TS
   QUEUE('DELAYQUE') FROM(reqid_value)
   ```

2. The task waits on another task by issuing a DELAY command, using the *reqid_value* as the REQID. For example:

   ```
   EXEC CICS DELAY
   INTERVAL(1000) REQID(reqid_value)
   ```

3. Another task, TRN2, reads the REQID of the DELAY request from TS queue called 'DELAYQUE'.
4. TRN2 completes its processing, and resumes TRN1 by canceling the DELAY request.

The process using a TS queue is illustrated in Figure 66 on page 177.

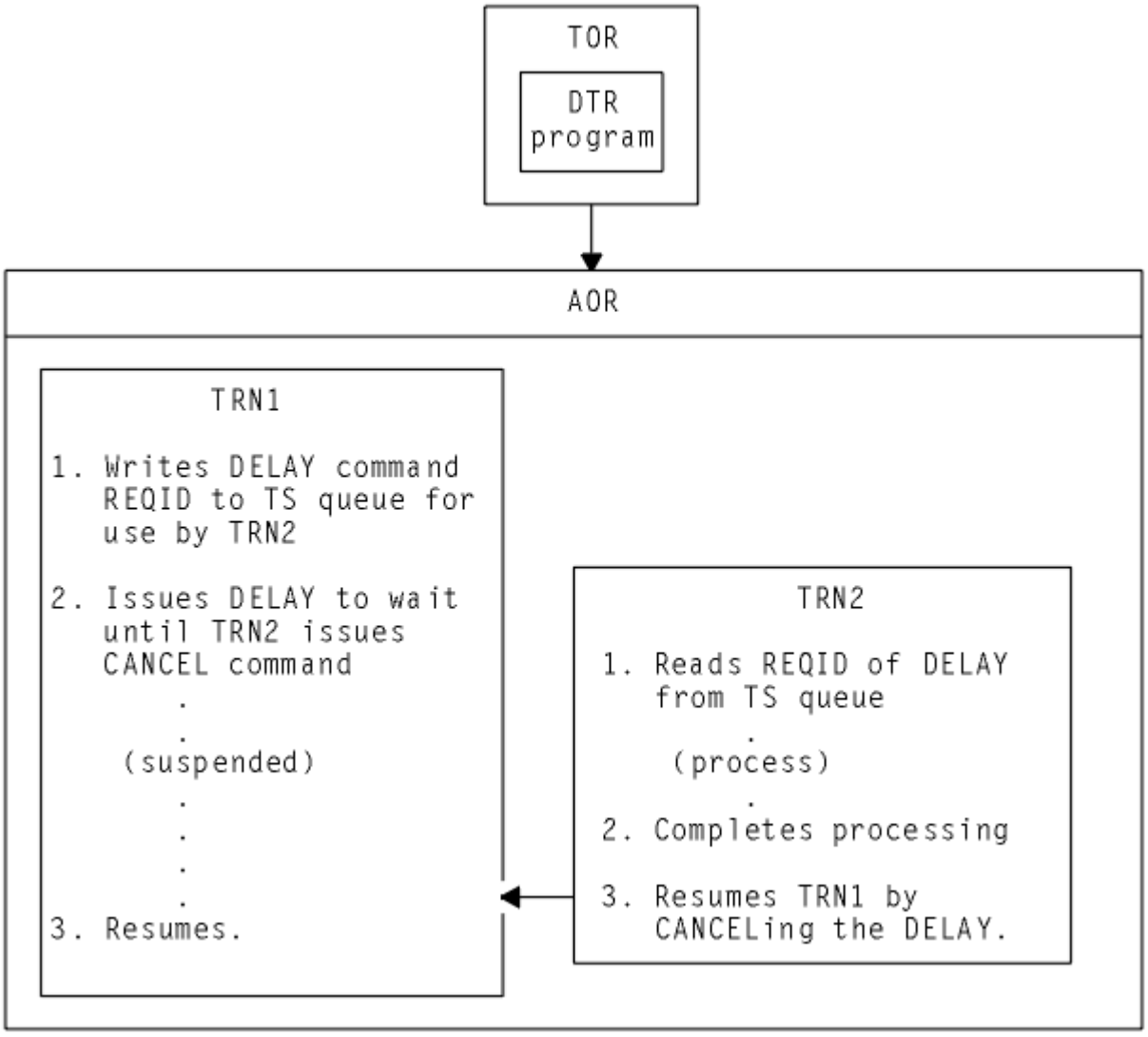


Figure 66. Illustration of the use of the DELAY and CANCEL REQID commands

Another way to pass the REQID when employing this technique would be for TRN1 to start TRN2 using the START command with the FROM and TERMID options. TRN2 could then obtain the REQID with the RETRIEVE command, using the INTO option.

Using this application programming technique, the CANCEL command that cancels the DELAY request must:

- Be issued in same target region as the DELAY command was executed in, or
- Specify the SYSID of the target region where the DELAY command was executed, or
- Specify a TRANSID (TRN1 in our example) that is defined as a remote transaction residing on the target region where the DELAY command was executed.

An application design based on the remote TRANSID technique only works for two target regions. An application design using the SYSID option on the cancel command only works for multiple target regions if all target regions have connections to all other target regions. In either case, the application programs need to be modified: there is no acceptable way to use this programming technique in a dynamic or distributed routing environment except by imposing restrictions on the routing program.

If the CANCEL command is issued by a transaction that is initiated from a terminal, it is possible that the transaction could be dynamically routed to the wrong target region.

Avoiding affinities using the POST and CANCEL REQID commands

The CICS POST command is used to request notification that a specified time has expired. Another transaction (TRN2) can force notification, as if the specified time has expired, by issuing a CANCEL of the POST request.

The time limit is signalled (posted) by CICS by setting a bit pattern in the event control block (ECB). To determine whether notification has been received, the requesting transaction (TRN1) either tests the ECB periodically, or issues a WAIT command on the ECB.

A TS storage queue can be used to pass the REQID of the POST request from task to task.

Figure 67 on page 178 illustrates this technique.

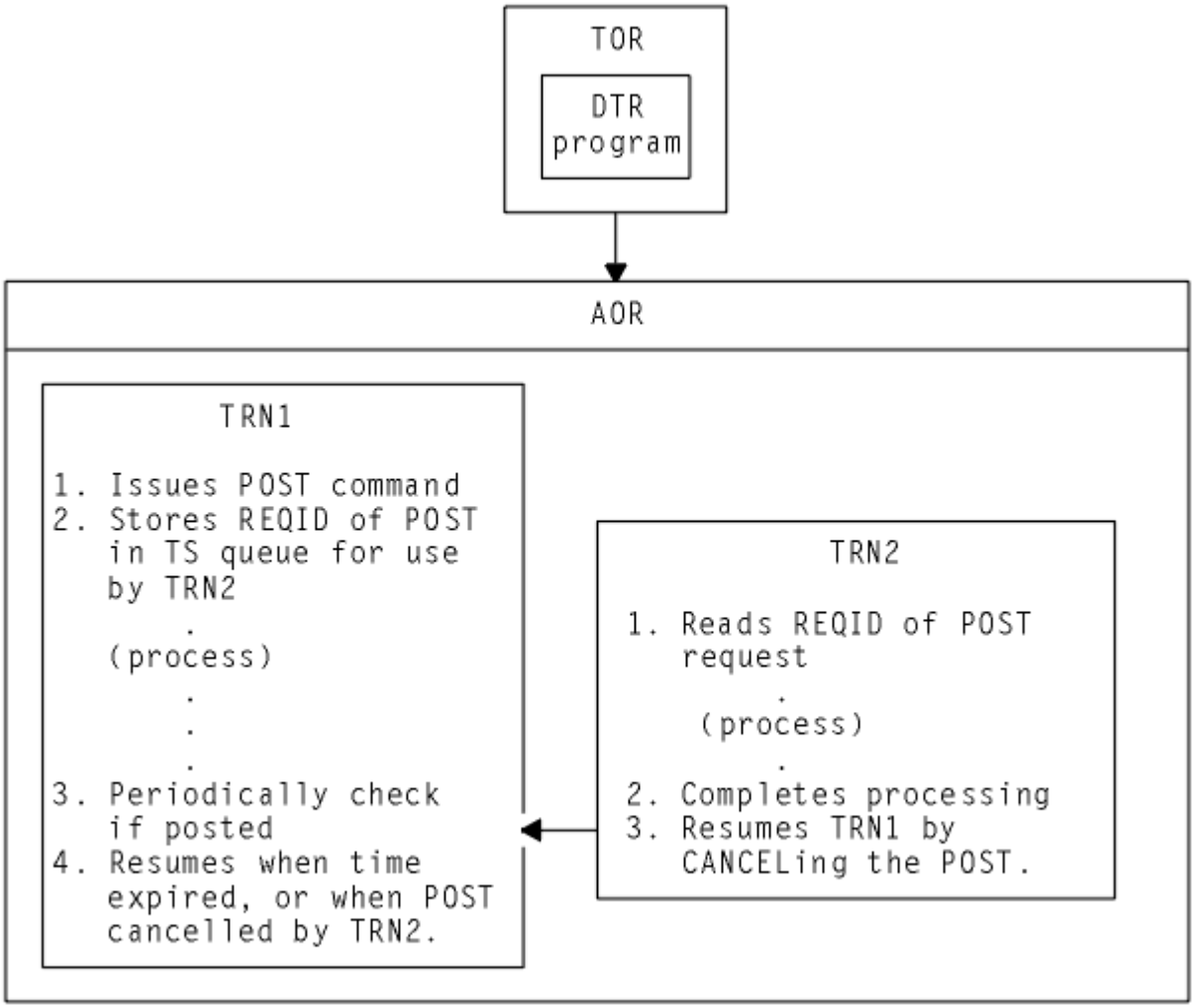


Figure 67. Illustration of the use of the POST command

If this technique is used, the dynamic or distributed routing program has to ensure that TRN2 executes in the same CICS region as TRN1 to preserve the application design.

The CANCEL command that notifies the task that issued the POST must:

- Be issued in same target region that the POST command was executed in, or
- Specify the SYSID of the target region where the POST command was executed, or
- Specify a TRANSID (TRN1 in our example) that is defined as a remote transaction residing on the target region where the POST command was executed.

An application design based on the remote TRANSID technique only works for two target regions. An application design using the SYSID option on the cancel command only works for multiple target regions if all target regions have connections to all other target regions. In either case, the application programs

need to be modified: there is no acceptable way to use this programming technique in a dynamic or distributed routing program except by imposing restrictions on the routing program.

In general, this means that the dynamic or distributed routing program has to ensure that TRN2 executes in the same region as TRN1 to preserve the application design.

Clearly, there is no problem if the CANCEL is issued by the same task that issued the POST. If a different task cancels the POST command, it must specify REQID to identify the particular instance of that command. Hence the CANCEL command with REQID is indicative of an inter-transaction affinity problem. However, REQID need not be specified on the POST command because CICS automatically generates a REQID and pass it to the application in EIBREQID.

Duration and scope of inter-transaction affinities

When planning your dynamic or distributed routing strategy, and planning how to manage inter-transaction affinities, it is important to understand the concepts of affinity relations and affinity lifetimes. The relations and lifetimes of inter-transaction affinities define the scope and duration of inter-transaction affinities.

Clearly, the ideal situation for a dynamic or distributed routing program is for there to be no inter-transaction affinities at all, which means there are no restrictions in the choice of available target regions for dynamic routing. However, even when inter-transaction affinities do exist, there are limits to the scope of these affinities, the scope of the affinity being determined by affinity relation and affinity lifetime.

Understanding the relations and lifetimes of transaction affinities is important in deciding how to manage them in a dynamic routing environment.

This section covers the following topics:

- "Affinity transaction groups" on page 179
- "Affinity relations and affinity lifetimes" on page 179

Affinity transaction groups

To manage affinities in a dynamic routing environment, you must first categorize transactions by their affinity. One way to do this is to place transactions in transaction groups, where a group is a set of transactions that have inter-transaction affinity.

Each affinity transaction group (or affinity group) represents a group of transactions that have an affinity with one another. Defining affinity groups is one way that a dynamic or distributed routing program can determine to which target region a transaction should be routed.

The more inter-transaction affinity you have in a CICS workload, the less effective a dynamic routing program can be in balancing the workload across a CICSplex. To minimize the impact of inter-transaction affinity, affinities in an affinity group are characterized by their affinity relation and affinity lifetime. These relation and lifetime attributes determine the scope and duration of an affinity. Therefore an affinity transaction group consists of an affinity group identifier, a set of transactions that constitute the affinity group, with the affinity relation and affinity lifetime associated with the group.

Affinity relations and affinity lifetimes

When you create a transaction group, you can assign to the group an affinity relation using the **AFFINITY** attribute and an affinity lifetime using the **AFFLIFE** attribute. The affinity relation determines how the dynamic or distributed routing program selects a target region for a transaction instance associated with the affinity. The affinity lifetime determines when the affinity ends.

The five possible affinity relations that you can assign to your affinity groups are BAPPL , `Global` , `LOCKED` , `LUname` , and `userid` . The affinity lifetime you assign to an affinity depends on the affinity relation. For example, if the affinity relation is `LOCKED` , the affinity lifetime must be UOW, whereas if the affinity relation is BAPPL, you can set the affinity lifetime to `Activity` , `Permanent` , `Process` or `System`.

The BAPPL relation

When you define an affinity relation of BAPPL to a transaction group, all instances of transactions that are associated with the same BTS process must run in the same target region for the lifetime of the affinity.

If the affinity relation is BAPPL , the affinity lifetime must have one of the following values:

Process
The affinity lasts for as long as the associated process exists.

Activity
The affinity lasts for as long as the associated activity exists.

System
The affinity lasts for as long as the target region exists, and ends whenever the target region terminates (at a normal, immediate, or abnormal termination).

Permanent
The affinity extends across all CICS restarts. If you are running CICSPlex SM , this affinity lasts for as long as any CMAS involved in managing the CICSplex using the workload is active.

A typical example of transactions that have a BAPPL relation is where a local temporary storage queue is used to pass data between the transactions within a BTS activity or process.

Figure 68 on page 180 shows an example of an affinity group with the BAPPL relation.

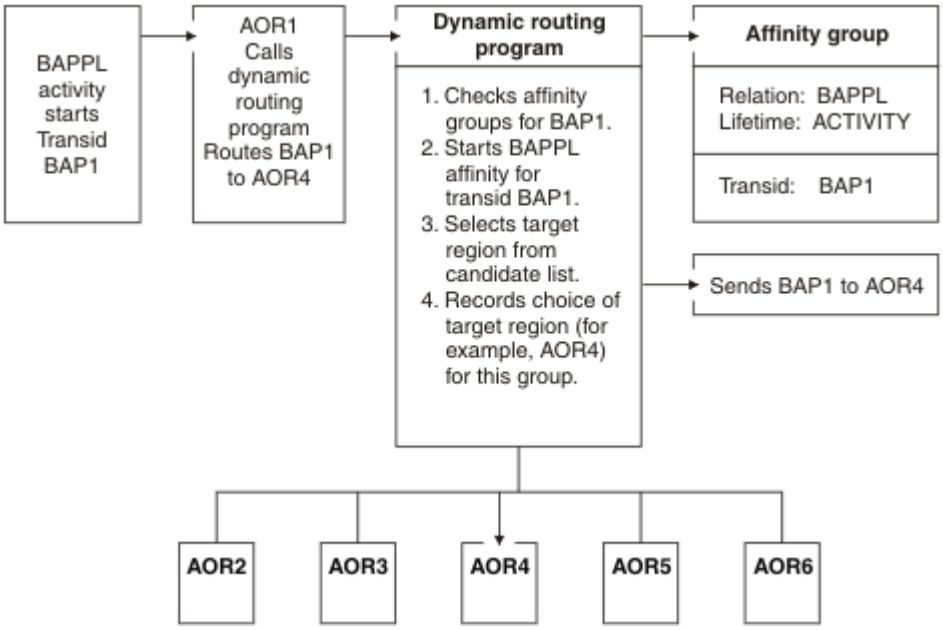


Figure 68. Managing inter-transaction affinity with BAPPL relation and activity lifetime

In this example, the first instance of BTS transaction BAP1 starts a BAPPL–activity affinity. The first instance of BAP1 can be routed to any suitable target region (AOR1 through AOR6), but all other instances of the activity must be routed to whichever target region is selected for BAP1.

Although BTS itself does not introduce any affinities, and discourages programming techniques that do, it does support existing code that might introduce affinities. You must define such affinities to workload management. It is particularly important to specify each affinity's lifetime. Failure to do this might restrict unnecessarily the workload management routing options.

It is important to note that a given activity can be run both synchronously and asynchronously. Workload management is able to honour only invocations that are made asynchronously. Furthermore, you are strongly encouraged not to create these affinities, particularly activity and process affinities, because these affinities are synchronized across the BTS-set. These affinities could have serious performance impacts on your systems.

You should also note that, with CICSPlex SM , the longest time that an affinity can be maintained is while a CMAS involved in the workload is active; that is, an affinity lifetime of PERMANENT. If a total system failure or a planned shutdown occurs, affinities are lost, but activities in CICS will be recovered from the BTS RLS data set.

The global relation

When you define an affinity relation of `global` to a transaction group, all instances of all the transactions started from any terminal, by any START command, or by any CICS BTS process, must run in the same target region for the lifetime of the affinity.

If the affinity relation is `global` , the affinity lifetime must have one of the following values:

System
The affinity lasts for as long as the target region exists, and ends whenever the target region terminates (at a normal, immediate, or abnormal termination).

Permanent
The affinity extends across all CICS restarts. This is the most restrictive of all the inter-transaction affinities. If you are running CICSPlex SM , this affinity lasts for as long as any CMAS involved in managing the CICSplex using the workload is active.

An example of a global inter-transaction affinity with a lifetime of permanent is where the transaction uses (reads or writes) a local, recoverable, temporary storage queue, and where the TS queue name is not derived from the terminal. (You can only specify that a TS queue is recoverable in the CICS region in which the queue is local.)

Generally, transactions in this affinity category are not suitable candidates for dynamic routing and you should consider making them statically routed transactions.

Figure 69 on page 181 shows an example of a global relation.

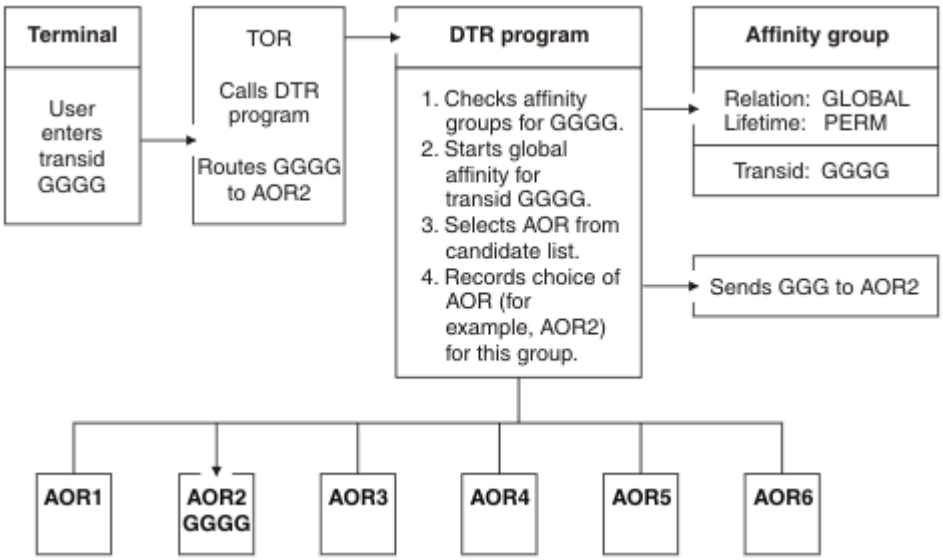


Figure 69. Managing inter-transaction affinity with global relation and permanent lifetime

In this example, the transaction GGGG is defined in a group with a permanent global affinity relation. The first instance of transid GGGG, from any terminal, starts a permanent-lifetime affinity. The first instance of GGGG can be routed to any suitable target region. In this example, AOR2 is selected from the possible range AOR1 through AOR6, but all other instances, from any terminal, must also be routed to the same region, AOR2.

The LOCKED relation

When you define an affinity relation of LOCKED to a transaction group, all instances of transactions in the group that are associated with dynamically-linked programs that have the same unit of work must run in the same target region for the lifetime of the unit of work.

The LOCKED relation is associated with a unit of work (UOW). The affinity lifetime for the LOCKED relation is always UOW . The unit of work ends either when a CICS **SYNCPOINT** or **ROLLBACK** request is run, or when the originating task terminates. UOW is valid only for LOCKED affinities between dynamic program link requests.

A typical example of transactions that have a LOCKED relation is where multiple invocations of the same dynamically-routed program access a common resource. Each invocation of the program has a transaction code under which the called program runs on a target system. By specifying that transaction code as part of a transaction group that designates a LOCKED affinity, all instances of dynamic program links for that program name cause the associated remote transaction to be rerouted to the same target region until the caller's unit of work ends. If those subsequent program links were routed to different regions, deadlocks could occur. Therefore work must not be routed away from the region that locked the resource. For further information, see Multiple links to the same server region.

Figure 70 on page 183 shows an example of an affinity with the LOCKED relation.

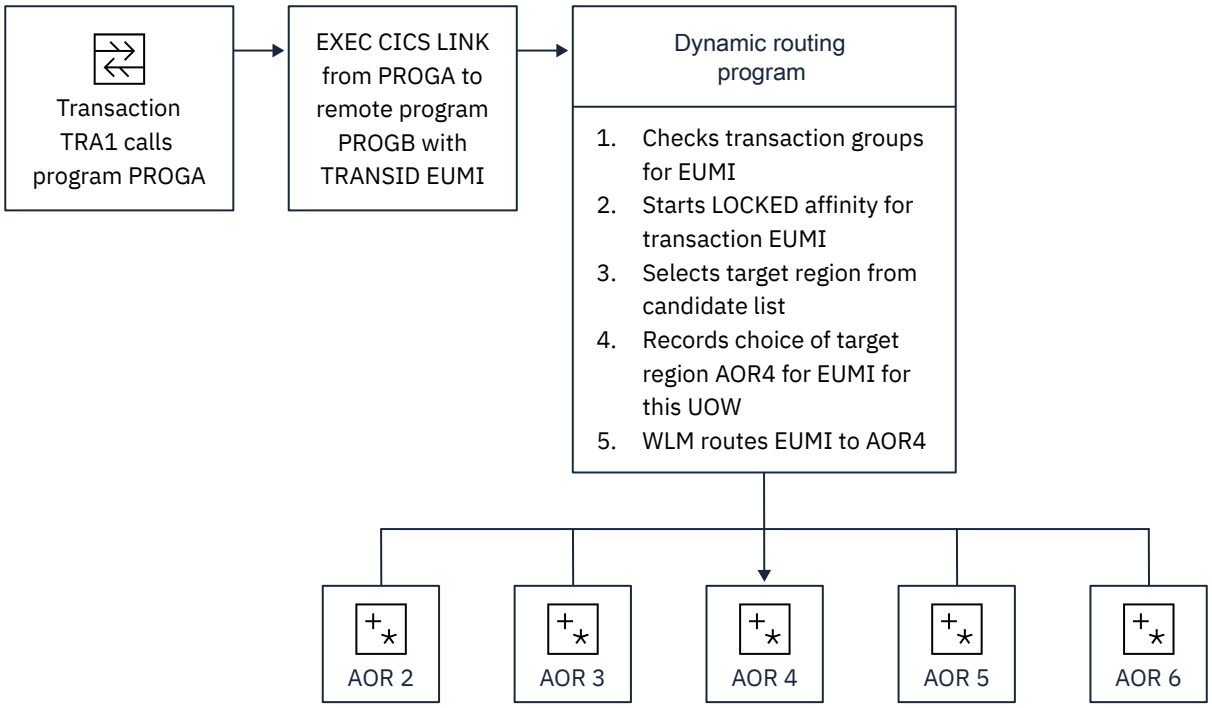


Figure 70. Managing inter-transaction affinity with LOCKED relation and UOW lifetime

In this example, transaction TRA1 calls program PROGA as part of a unit of work identified by the **NETUOWID** attribute of the WLMATAFF resource table. PROGA makes an **EXEC CICS LINK** request to program PROGB on a remote system. PROGB is defined as dynamic and its associated transaction, defined by the **TRANSID** attribute in the program definition, is EUMI. EUMI is a user-defined mirror transaction belonging to a transaction group that is defined with an affinity relation of LOCKED and an affinity lifetime of UOW. Workload management routes this transaction to CICS region AOR4. The dynamic routing program, after checking whether an affinity already exists, starts a new affinity for EUMI. All subsequent instances of EUMI share this affinity for the lifetime of the unit of work, and are routed to AOR4 until the unit of work ends.

The LUname (terminal) relation

When you define an affinity relation of LUname to a transaction group, all instances of all the transactions in the group that are associated with the same terminal must run in the same target region for the lifetime of the affinity.

If the affinity relation is LUname , the affinity lifetime must have one of the following values:

Pseudoconversation
The affinity lasts for the whole pseudoconversation, and ends when the pseudoconversation ends at the terminal. Each transaction must end with EXEC CICS RETURN TRANSID, not with the pseudoconversation mode of END.

Logon
The affinity lasts for as long as the terminal remains logged on to CICS, and ends when the terminal logs off.

System
The affinity lasts for as long as the target region exists, and ends whenever the target region terminates (at a normal, immediate, or abnormal termination).

Permanent
The affinity extends across all CICS restarts. If you are running CICSPlex SM , this affinity lasts for as long as any CMAS involved in managing the CICSplex using the workload is active.

Delimit
The affinity continues until a transaction with a pseudoconversation mode of END is encountered.

A typical example of transactions that have an LUname relation are those that:

- Use a local TS queue to pass data between the transactions in a pseudoconversation, and
- Use a TS queue name that is derived, in part, from the terminal name (see “Avoiding affinities when using temporary storage” on page 170 for information about TS queue names).

These types of transactions can be placed in an affinity group with a relation of terminal and a lifetime of pseudoconversation. When the dynamic routing program detects the first transaction in the pseudoconversation initiated by a specific terminal (LUname), it is free to route the transaction to any target region that is a valid candidate for that transaction. However, any subsequent transaction in the affinity group that is initiated at the same terminal must be routed to the same target region as the transaction that started the pseudoconversation. When the affinity ends (at the end of the pseudoconversation on a given terminal), the dynamic routing program is again free to route the first transaction to any candidate target region.

This form of affinity is manageable and does not impose too severe a constraint on dynamic transaction routing, and might occur in many CICSplexes. It can be managed easily by a dynamic routing program, and should not inhibit the use of dynamic routing.

Figure 71 on page 185 shows this type of example.

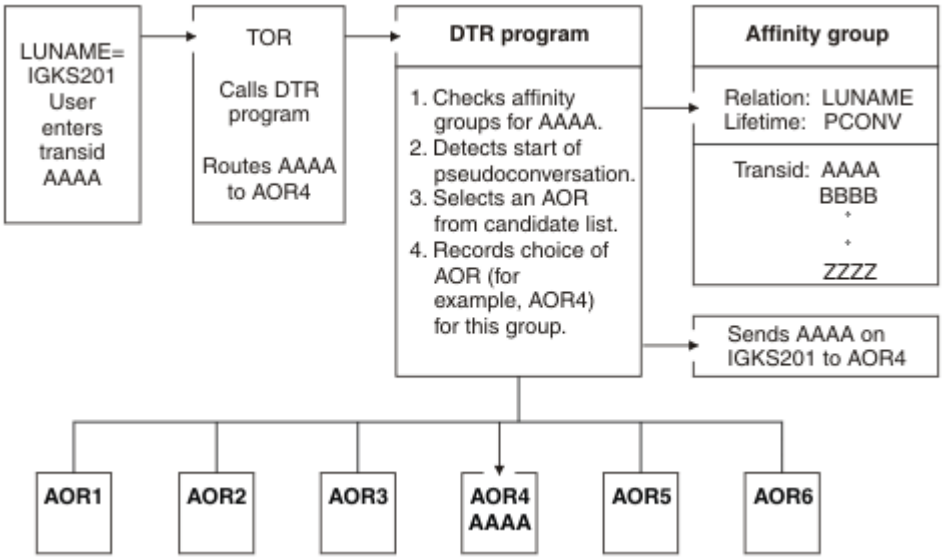


Figure 71. Managing inter-transaction affinity with LUname relation and pseudoconversation lifetime

In this example, each instance of transid AAAA from a terminal starts a pseudoconversational-lifetime affinity. AAAA can be routed to any suitable target region (AOR1 through AOR6), but other transactions in the group, in the same pseudoconversation with the same terminal (IGKS201 in this example), must be routed to whichever target region is selected for AAAA.

The userid relation

When you define an affinity relation of `userid` to a transaction group, all instances of the transactions that are initiated from a terminal, by a **START** command, or by a CICS BTS activity, and executed on behalf of the same user ID, must execute in the same target region for the lifetime of the affinity.

If the affinity relation is `userid` , the affinity lifetime must have one of the following values:

Pseudoconversation
The affinity lasts for the whole pseudoconversation, and ends when the pseudoconversation ends for that user ID. Each transaction must end with **`EXEC CICS RETURN TRANSID`** , not with the pseudoconversation mode of END.

Signon
The affinity lasts for as long as the user is signed on, and ends when the user signs off. This lifetime is possible only in those situations where only one user per user ID is permitted. Signon lifetime cannot be detected if multiple users are permitted to be signed on with the same user ID at the same time (at different terminals).

System
The affinity lasts for as long as the target region exists, and ends whenever the target region terminates (at a normal, immediate, or abnormal termination).

Permanent
The affinity extends across all CICS restarts. If you are running CICSPlex SM , this affinity lasts for as long as any CMAS involved in managing the CICSplex using the workload is active.

Delimit
The affinity continues until a transaction with a pseudoconversation mode of END is encountered.

A typical example of transactions that have a `userid` relation is where the user ID is used dynamically to identify a resource, such as a TS queue. The least restrictive of the affinities in this category is one that lasts only for as long as the user remains signed on.

Figure 72 on page 186 shows an example of an affinity group with the userid relation and a signon lifetime.

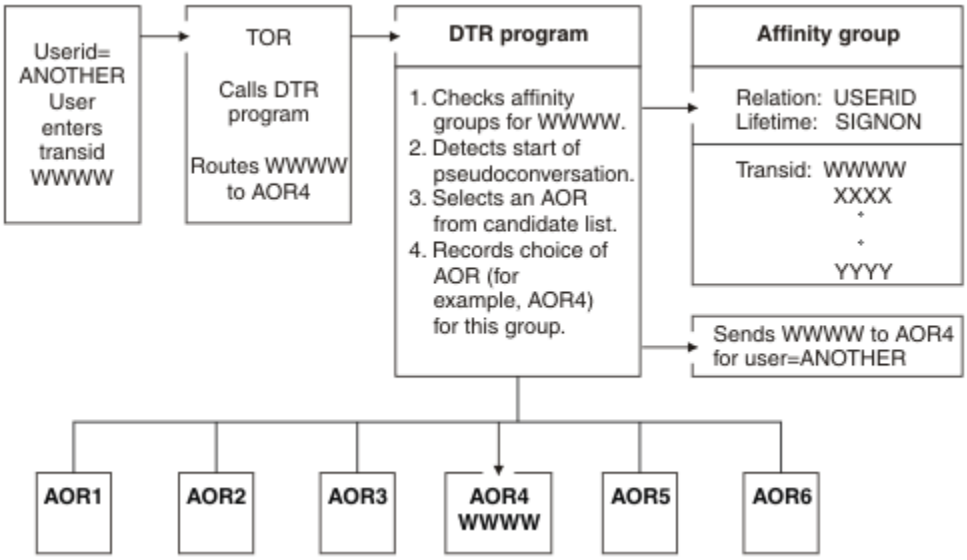


Figure 72. Managing inter-transaction affinity with `userid` relation and `signon` lifetime

In this example, any instance of a transaction from a terminal starts a `signon` lifetime affinity. It can be routed to any suitable target region (AOR1 through AOR6), but other transactions in the group for the same user (ANOTHER in this example) must be routed to whichever target region is selected for the first instance of a transaction in the group.

Recovery design

CICS provides two techniques - journaling and syncpointing - that can help you to recover or reconstruct events or data changes during CICS execution.

Techniques for named counter recovery are described in “Named counter recovery” on page 344.

Journaling

CICS provides facilities for creating and managing *journals* during CICS processing. Journals can contain any data the user needs to facilitate subsequent reconstruction of events or data changes.

For example, a journal might act as an audit trail, a change-file of database updates and additions, or a record of transactions passing through the system (often referred to as a *log*). Each journal can be written from any task.

Journal control commands are provided to allow the application programmer to:

- Create a journal record (WRITE JOURNALNAME or WRITE JOURNALNUM command)
- Synchronize with (wait for completion of) journal output (WAIT JOURNALNAME or WAIT JOURNALNUM command)

Exception conditions that occur during execution of a journal control command are handled as described in “Dealing with exception conditions” on page 189 . (The earlier JFILEID option is supported for compatibility purposes only.)

Journal records

Each journal is identified by a name or number known as the journal identifier. This number can range from 1 through 99. The name DFHLOG is reserved for the journal known as the system log.

When a journal record is built, the data is moved to the journal buffer area. All buffer space and other work areas needed for journal operations are acquired and managed by CICS. The user task supplies only the data to be written to the journal. Log manager is designed so that the application programmer requesting that output services does not have to be concerned with the detailed layout and precise

contents of journal records. The programmer has to know only which journal to use, what user data to specify, and which user-identifier to supply.

Journal output synchronization

A synchronous journal record is created by issuing the WRITE JOURNALNAME or WRITE JOURNALNUM command with the WAIT option, the requesting task can wait until the output has been completed.

By specifying this command, the application programmer ensures that the journal record is written on the external storage device associated with the journal before processing continues; the task is said to be *synchronized* with the output operation.

The application programmer can also request asynchronous journal output. This causes a journal record to be created in the journal buffer area but allows the requesting task to retain control and thus to continue with other processing. The task can check and wait for output completion (that is, synchronize) later by issuing the WAIT JOURNALNAME or WAIT JOURNALNUM command.

Note: In some cases, a SHUTDOWN IMMEDIATE can cause user journal records to be lost, if they have been written to a log manager buffer but not to external storage. This is also the case if the CICS shut-down assist transaction (CESD) forces SHUTDOWN IMMEDIATE during a normal shutdown, because normal shutdown is hanging. To avoid the risk of losing journal records, you are recommended to issue CICS WAIT JOURNALNUM requests periodically, and before ending your program.

Without WAIT, CICS does not write data to the log stream until it has a full buffer of data, or until some other unrelated activity requests that the buffer be hardened, thus reducing the number of I/O operations. Using WAIT makes it more difficult for CICS to calculate accurately log structure buffer sizes. For CF log streams, this could lead to inefficient use of storage in the coupling facility.

The basic process of building journal records in the CICS buffer space of a given journal continues until one of the following events occurs:

- For system logs:
  - Whenever the system requires it to ensure integrity and to permit a future emergency restart
  - The log stream buffer is filled
- For user journals:
  - The log stream buffer is filled (or, if the journal resides on SMF, when the journal buffer is filled)
  - A request specifying the WAIT option is made (from any task) for output of a journal record
  - An EXEC CICS SET JOURNALNAME command is issued
  - An EXEC CICS DISCARD JOURNALNAME command is issued
  - Any of the previously listed events occurring for any other journal which maps onto the same log stream
  - On a normal shutdown
- For forward recovery logs:
  - The log stream buffer is filled
  - At syncpoint (first phase)
  - On file closure
- For autojournals:
  - The log stream buffer is filled
  - A request specifying the WAIT option is made (from any task) for output of a journal record
  - On file closure
- For the log-of-logs (DFHLGLOG):
  - On file OPEN and CLOSE requests

When any one of these occurs, all journal records present in the buffer, including any deferred output resulting from asynchronous requests, are written to the log stream as one block.

The advantages that may be gained by deferring journal output are:

- Transactions may get better response times by waiting less.
- The load of physical I/O requests on the host system may be reduced.
- Log streams may contain fewer but larger blocks and so better utilize primary storage.

However, these advantages are achievable only at the cost of greater programming complexity. It is necessary to plan and program to control synchronizing with journal output. Additional decisions that depend on the data content of the journal record and how it is to be used must be made in the application program. In any case, the full benefit of deferring journal output is obtained only when the load on the journal is high.

If the journal buffer space available at the time of the request is not sufficient to contain the journal record, the NOJBUFSP condition occurs. If no HANDLE CONDITION command is active for this condition, the requesting task loses control, the contents of the current buffer are written, and the journal record is built in the resulting freed buffer space before control returns to the requesting task.

If the requesting task is not willing to lose control (for example, if some housekeeping must be performed before other tasks get control), a HANDLE CONDITION command should be issued. If the NOJBUFSP condition occurs, no journal record is built for the request, and control is returned directly to the requesting program at the location provided in the HANDLE CONDITION command. The requesting program can perform any housekeeping needed before reissuing the journal output request.

Journal commands can cause immediate or deferred output to the journal. System log records are distinguished from all other records by specifying JOURNALNAME(DFHLOG) on the request. User journal records are created using some other JOURNALNAME or a JOURNALNUM. All records must include a journal type identifier, (JTYPEID). If the user journaling is to the system log, the journal type identifier (according to the setting of the high-order bit) also serves to control the presentation of these to the global user exit XRCINPT at a warm or emergency restart. Records are presented during the backward scan of the log as follows:

- For in-flight or indoubt tasks only (high-order bit off)
- For all records encountered until the scan is terminated (high-order bit on)

Syncpointing

To facilitate recovery in the event of abnormal termination of a CICS task or of failure of the CICS system, the system programmer can, during CICS table generation, define specific resources (for example, files) as recoverable. If a task is terminated abnormally, these resources are restored to the condition they were in at the start of the task, and can then be rerun.

The process of restoring the resources associated with a task is termed **backout**.

If an individual task fails, backout is performed by the dynamic transaction backout program. If the CICS system fails, backout is performed as part of the emergency restart process. See Starting CICS with the START=INITIAL parameter which describes these facilities, which in general have no effect on the coding of application programs.

However, for long-running programs, it may be undesirable to have a large number of changes, accumulated over a period of time, exposed to the possibility of backout in the event of task or system failure. This possibility can be avoided by using the SYNCPOINT command to split the program into logically separate sections known as units of work (UOWs); the end of an UOW is referred to as a synchronization point (**syncpoint**). For more information about syncpoints, see Troubleshooting for recovery processing.

If failure occurs after a syncpoint but before the task has been completed, only changes made after the syncpoint are backed out.

Alternatively, you can use the SAA Resource Recovery interface instead of the SYNCPOINT command. This provides an alternative API to existing CICS resource recovery services. You may want to use the SAA

Resource Recovery interface in networks that include multiple SAA platforms, where the consistency of a common API is seen to be of benefit. In a CICS system, the SAA Resource Recovery interface provides the same function as the EXEC CICS API.

Restriction: Full SAA Resource Recovery provides some return codes that are not supported in its CICS implementation. (See the CICS appendix in Systems Application Architecture Common Programming Interface Resource Recovery Reference.

The SAA Resource Recovery interface is implemented as a call interface, having two call types:

SRRCMIT
Commit—Equivalent to SYNCPOINT command.

SRRBACK
Backout—Equivalent to SYNCPOINT ROLLBACK command.

For further information about the SAA Resource Recovery interface, see Systems Application Architecture Common Programming Interface Resource Recovery Reference.

UOWs should be entirely logically independent, not merely with regard to protected resources, but also with regard to execution flow. Typically, an UOW comprises a complete conversational operation bounded by SEND and RECEIVE commands. A browse is another example of an UOW; an ENDBR command must therefore precede the syncpoint.

In addition to a DL/I termination call being considered to be a syncpoint, the execution of a SYNCPOINT command causes CICS to issue a DL/I termination call. If a DL/I PSB is required in a subsequent UOW, it must be rescheduled using a program control block (PCB) call or a SCHEDULE command.

With distributed program link (DPL), it is possible to specify that a syncpoint is taken in the server program, to commit the server resources before returning control to the client. This is achieved by using the SYNCONRETURN option on the LINK command. For programming information about the SYNCONRETURN option, see "The SYNCONRETURN option for the server program" in “Examples of distributed program link” on page 214 and CICS command summary.

A BMS logical message, started but not completed when a SYNCPOINT command is processed, is forced to completion by an implied SEND PAGE command. However, you should not rely on this because a logical message whose first page is incomplete is lost. You should also code an explicit SEND PAGE command before the SYNCPOINT command or before termination of the transaction.

Consult your system programmer if syncpoints are to be issued in a transaction that is eligible for transaction restart.

Dealing with exception conditions

Every time you process an EXEC CICS command in one of your applications, CICS automatically raises a condition, or return code, to tell you what happened.

You can choose that the CICS EXEC interface program passes this condition back to your application. This condition is also called a RESP value, because you can obtain it by using the RESP option in your command. Alternatively, you can obtain this value by reading it from the EXEC interface block (EIB). The condition is usually NORMAL.

If something unusual happens, you get an *exception condition* , that is, a condition other than NORMAL. By testing this condition, you can find out what has happened and, possibly, why.

Many exception conditions have an additional value (RESP2) associated with them, which gives further information. To obtain this RESP2 value, you can use the RESP2 option (in addition to the RESP option) in your command, or you can read it from the EIB.

Some conditions, even though they are not NORMAL, do not indicate an error situation. For example, an ENDFILE condition on a READNEXT command during a file browse, might indicate expected behavior. For information about all possible conditions and the commands on which they can occur, see Application development reference.

This section describes:

- “Default CICS exception handling” on page 190
- “Handling exception conditions by inline code” on page 190
- “ Modifying default CICS exception handling ” on page 193

Default CICS exception handling

For COBOL, PL/I, and assembler language applications (but not AMODE(64) assembler language applications), unless you specify otherwise, CICS uses its built-in exception handling whenever an exception condition occurs. For AMODE(64) assembler language applications and applications written in C or C++, CICS takes no action when an exception condition occurs. The application must handle the exception condition.

See “Handling exception conditions by inline code” on page 190 for information about handling exception conditions.

With CICS default exception handling, when an exception condition occurs, the most frequent action is to cause an abend. For details about the specific behaviors for each condition and for each command, refer to Application development reference and Introduction to System programming commands.

If the CICS default exception handling does not meet your requirements, you can specify other actions in the following ways:

- Turn off the CICS default exception handling on a specific EXEC CICS command call by specifying the NOHANDLE option.
- Turn off the CICS default exception handling by specifying the RESP option on the command. This option switches off the default CICS exception handling in the same way as the NOHANDLE command. It also updates the variable named by the argument of the RESP option with the value of the condition returned by the command. For details, see “Handling exception conditions by inline code” on page 190.

If you turn off the default CICS exception handling, you must ensure that your program copes with anything that might happen in the command call.

It is possible to use combinations of the HANDLE ABEND, HANDLE CONDITION, and IGNORE CONDITION commands to modify the default CICS exception handling, but this is no longer recommended. For details, see “ Modifying default CICS exception handling ” on page 193.

Handling exception conditions by inline code

If your program is written in C or C++, or is an AMODE(64) assembler language application, inline code is the only technique available to handle exception conditions. If your program is not written in C or C++ and is not an AMODE(64) assembler language application, handling exception conditions involves either using the NOHANDLE option or specifying the RESP option on **EXEC CICS** commands, which prevents CICS from performing its default exception handling.

About this task

The RESP option makes the value of the exception condition directly available to your program, for it to take remedial action.

If you use the NOHANDLE or RESP option, ensure that your program can cope with whatever condition might arise in the course of executing the commands. The RESP value is available to enable your program to decide what to do and more information which it might need to use is carried in the EXEC interface block (EIB). In particular, the RESP2 value is contained in one of the fields of the EIB. For more information about EIB, see EIB fields . Alternatively, if your program specifies RESP2 in the command, the RESP2 value is returned by CICS directly.

The DFHRESP built-in translator function makes it easy to test the RESP value, because you can examine RESP values symbolically. This technique is easier than examining binary values that are less meaningful to someone reading the code.

How to use the RESP and RESP2 options

The argument of RESP is a user-defined fullword binary data area (long integer). On return from the command, it contains a value that corresponds to the condition that might have been raised. Normally, its value is DFHRESP(NORMAL).

Use of RESP and DFHRESP in COBOL and PL/I

The following example shows an EXEC CICS call in COBOL that uses the RESP option. A PL/I example is similar, but ends with a semicolon (;) rather than END-EXEC.

```
EXEC CICS WRITEQ TS FROM(abc)
QUEUE(qname)
NOSUSPEND
RESP(xxx)
END-EXEC.
```

The following code is an example of using DFHRESP to test for the RESP value:

```
IF xxx=DFHRESP(NOSPACE) THEN …
```

Use of RESP and DFHRESP in C and C++

The following example shows an EXEC CICS call in C that uses the RESP option, including the declaration of the RESP variable:

```
long response;
.
.
.
EXEC CICS WRITEQ TS FROM(abc)
QUEUE(qname)
NOSUSPEND
RESP(response);
```

The following code is an example of using DFHRESP to test for the RESP value:

```
if (response == DFHRESP(NOSPACE))
{
.
.
.
}
```

Use of DFHRESP in assembler language

The following code is an example of a test for the RESP value in assembler language:

```
 CLC xxx,DFHRESP(NOSPACE)
BE ...
```

An example of exception handling in C

The following example is a typical function which could be used to receive a BMS map and to cope with exception conditions.

```
int ReadAccountMap(char *mapname, void *map)
{
long response;
int ExitKey;
EXEC CICS RECEIVE MAP(mapname)
MAPSET("ACCOUNT")
INTO(map)
RESP(response);
switch (response)
{
case DFHRESP(NORMAL):
ExitKey = dfheiptr->eibaid;
ModifyMap(map);
break;
case DFHRESP(MAPFAIL):
ExitKey = dfheiptr->eibaid;
break;
default:
ExitKey = DFHCLEAR;
break;
}
return ExitKey;
}
```

Figure 73. An example of exception handling in C

The *ReadAccountMap* function has two arguments:

1. *mapname* is the variable which contains the name of the map which is to be received.
2. *map* is the address of the area in memory to which the map is to be written.

The RESP value will be returned in *response* . The declaration of *response* sets up the appropriate type of automatic variable.

The EXEC CICS statement asks for a map of the name given by *mapname* , of the mapset ACCOUNT, to be read into the area of memory to which the variable *map* points, with the value of the condition being held by the variable *response*.

The condition handling can be done by using if statements. However, to improve readability, it is often better, as here, to use a switch statement, instead of compound if ... else statements. The effect on program execution time is negligible.

Specific cases for two conditions:

1. A condition of NORMAL is what is normally expected. If a condition of NORMAL is detected in the example here, the function then finds out what key the user pressed to return to CICS and this value is passed to ExitKey. The program then makes some update to the map held in memory by the ModifyMap function, which need not concern us further.
2. A condition of MAPFAIL, signifying that the user has made no updates to the screen, is also fairly normal and is specifically dealt with here. In this case the program again updates ExitKey but does not call ModifyMap.

In this example, any other condition is held to be an error. The example sets ExitKey to DFHCLEAR—the same value that it would have set if the user had cleared the screen—which it then returns to the calling program. By checking the return code from ReadAccountMap, the calling program would know that the map had not been updated and that some remedial action is required.

An example of exception handling in COBOL

The following example is a typical function which could be used to receive a BMS map and to cope with exception conditions.

```
03 RESPONSE PIC S9(8) BINARY.
03 EXITKEY PIC X.

EXEC CICS RECEIVE MAP(MAPNAME)
MAPSET('ACCOUNT')
INTO(MAP)
RESP(RESPONSE)
END-EXEC.
IF (RESPONSE NOT = DFHRESP(NORMAL)) AND
(RESPONSE NOT = DFHRESP(MAPFAIL))
MOVE DFHCLEAR TO EXITKEY
ELSE
MOVE EIBAID TO EXITKEY
IF RESPONSE = DFHRESP(NORMAL)
GO TO MODIFYMAP
END-IF
END-IF.

MODIFYMAP.
```

Figure 74. An example of exception handling in COBOL

MAPNAME is the variable which contains the name of the map which is to be received.

The RESP value is returned in RESPONSE. RESPONSE is declared as a fullword binary variable in the data section.

The EXEC CICS statement asks for a map of the name given by *MAPNAME* , of the mapset ACCOUNT, to be read, with the value of the condition being held by the variable *RESPONSE*.

The condition handling is done by using IF ... statements. If the condition is neither NORMAL nor MAPFAIL the program behaves as if the user had cleared the screen.

If the condition is either NORMAL or MAPFAIL the program saves the value of the key which the user pressed to exit the screen in EXITKEY. In addition, if the condition is NORMAL, the program branches to MODIFYMAP to perform some additional function.

Modifying default CICS exception handling

You can use the **HANDLE CONDITION** , **IGNORE CONDITION** , and **HANDLE ABEND** commands in your application program to modify how CICS handles exceptions and abends.

About this task

This information applies only to COBOL, PL/I, or assembler language applications (but not AMODE(64) assembler language applications). For any other supported high level language, these commands are not supported.

A summary of the commands is as follows:

HANDLE CONDITION
Specify the label to which control is to be passed if a condition occurs.

IGNORE CONDITION
Specify that no action is to be taken if a condition occurs.

HANDLE ABEND
Activate, cancel, or reactivate an exit for abnormal termination processing.

An abend is the most frequent way in which CICS handles exception conditions.

The current effect of the **IGNORE CONDITION** , **HANDLE ABEND** , and **HANDLE CONDITION** commands can be suspended by using the **PUSH HANDLE** command, and reinstated by using the **POP HANDLE** command.

You can pass control to a specified label in the following ways:

- Use a **HANDLE CONDITION** condition(label) command, where condition is the name of an exception condition.
- Use a **HANDLE CONDITION** ERROR(label) command.

The **HANDLE CONDITION** command sets up CICS code to name specific conditions, and then uses this code to pass control to appropriate sections of your application if those conditions arise. With an active **HANDLE CONDITION** command, control goes to the label that you specified for that specific condition.

The same condition might occur on many different commands, and for different reasons. For example, an IOERR condition can occur during file control operations, interval control operations, and others. Therefore, you need to identify the command that has raised a specific condition before you can investigate why it has happened. This is a good reason for using the RESP option in new CICS applications. Although you need only one **HANDLE CONDITION** command to set error-handling for several conditions, sometimes it is awkward to identify exactly which of several **HANDLE CONDITION** commands is currently active when a CICS command fails somewhere in your code.

If a condition that you have not named occurs, CICS takes the default action, unless this action is to abend the task, in which case it raises the ERROR condition. If you name the condition but leave out its label, any **HANDLE CONDITION** command for that condition is deactivated, and if the condition occurs, CICS takes the default action.

A common source of errors with the **HANDLE CONDITION** command is not dealing with all conditions. If you use an unfamiliar command, check the command description to find out which exception conditions are possible (see Application development reference). Even when you issue HANDLE commands for all the exception conditions, the error-handling code might not be finished. Sometimes the result is an error-handling routine that, by issuing a RETURN command, allows incomplete or incorrect data changes to be committed.

A good approach is to use the **HANDLE CONDITION** command, but to let the system default action proceed if there is no obvious way round a specific problem.

Bearing in mind the distinction between an error condition, a condition that merely causes a wait (see “How CICS keeps track of what to do” on page 196 for examples of conditions that cause a wait), and the special case of the **SEND MAP** command overflow processing, a **HANDLE CONDITION** command is active after a **HANDLE CONDITION** condition(label), or **HANDLE CONDITION** ERROR(label) command has been run in your application.

If no **HANDLE CONDITION** command is active for a condition, but one is active for ERROR, control passes to the label for ERROR, if the condition is an error, not a wait.

If you use **HANDLE CONDITION** commands, or are maintaining an application that uses them, do not include any commands in your error routine that can cause the same condition that gave you the original branch to the routine, because you will cause a loop.

Take special care not to cause a loop on the ERROR condition itself. You can avoid a loop by reverting temporarily to the system default action for the ERROR condition. Do this by coding a **HANDLE CONDITION ERROR** command with no label specified. At the end of your error processing routine, you can reinstate your error action by including a **HANDLE CONDITION ERROR** command with the appropriate label. If you know the previous **HANDLE CONDITION** state, you can do this explicitly. In a general subroutine, which might be called from several different points in your code, the **PUSH HANDLE** and **POP HANDLE** commands might be useful. See “Using PUSH HANDLE and POP HANDLE commands” on page 198.

Using the HANDLE CONDITION command

Use the **HANDLE CONDITION** command to specify the label to which control is to be passed if a condition occurs.

About this task

When you use this command, you must include the name of the condition and ensure that the **HANDLE CONDITION** command is executed before the command that might give rise to the associated condition.

Restriction: This command is supported only in COBOL, PL/I, and assembler language applications (but not AMODE(64) assembler language applications). It is not supported in all other supported high level languages.

You cannot include more than 16 conditions in the same command. You must specify any additional conditions in further **HANDLE CONDITION** commands. You can also use the ERROR condition in the same list to specify that all other conditions cause control to be passed to the same label.

The **HANDLE CONDITION** command for a given condition applies only to the program in which it is specified. The command remains active while the program is running, or until one of the following events:

- An **IGNORE CONDITION** command for the same condition is met. The **HANDLE CONDITION** command is overridden.
- Another **HANDLE CONDITION** command for the same condition is met. The new command overrides the previous one.

The **HANDLE CONDITION** command is temporarily deactivated by the NOHANDLE or RESP option on a command.

When control passes to another program, by a **LINK** or **XCTL** command, the **HANDLE CONDITION** commands that were active in the calling program are deactivated. When control returns to a program from a program at a lower logical level, the **HANDLE CONDITION** commands that were active in the higher-level program before control was transferred from it are reactivated, and those in the lower-level program are deactivated. (See “Program linking” on page 147 for information about logical levels.)

The following example shows how to handle conditions, for example DUPREC and LENGERR, that can occur when you use a **WRITE** command to add a record to a data set. In the example, DUPREC is handled as a special case. Standard system action is taken for LENGERR; that is, terminate the task abnormally. All other conditions are handled by the error routine ERRHANDL.

```
 EXEC CICS HANDLE CONDITION
ERROR(ERRHANDL)
DUPREC(DUPRTN) LENGERR
END-EXEC.
```

In a PL/I application program, a branch to a label in an inactive procedure or in an inactive begin block, caused by a condition, produces unpredictable results.

In an assembler language application program, when a branch to a label is caused by a condition, the registers in the application program are restored to their values in the program at the point where the command that caused the condition is issued.

RESP and NOHANDLE options

You can temporarily deactivate the effect of any HANDLE CONDITION command by using the RESP or NOHANDLE option on a command.

The way to use these options is described in “Handling exception conditions by inline code” on page 190 . If you do this, you lose the ability to use any system default action for that command. In other words, you have to do your own "catch-all" error processing.

How CICS keeps track of what to do

CICS has a table of the conditions referred to by HANDLE CONDITION and IGNORE CONDITION commands in your application.

Each execution of one of these commands either updates an existing entry in this table, or causes CICS to make a new entry if this is the first time the condition has been quoted in such a command. Each entry tells CICS what to do by indicating one of the three exception-handling states your application can be in, namely:

1. **Let the program continue**, with control coming straight back from CICS to the next instruction following the command that has failed in your program. You can then find out what happened by testing, for example, the RESP value that CICS returns after executing a command. The result of this test enables you decide what to do next. For details, see “Handling exception conditions by inline code” on page 190.

   This is the recommended method, which is the approach taken in the "File A" sample programs referred to in Samples. It is also the recommended approach for any new CICS applications. It lends itself to structured code and removes the need for implied GOTOs that CICS required in the past.
2. **Pass control to a specified label** if a named condition arises. You do this by using a HANDLE CONDITION command or HANDLE CONDITION ERROR command to name both the condition and the label of a routine in your code to deal with it. For details, see “Using the HANDLE CONDITION command” on page 195 and “Using the HANDLE CONDITION ERROR command” on page 196.
3. **Taking the CICS system default action** , where for most conditions, this is to terminate the task abnormally and means that you do nothing by way of testing or handling conditions.

For the conditions ENQBUSY, NOJBUFSP, NOSTG, QBUSY, SESSBUSY, and SYSBUSY, the normal default is to force the task to wait until the required resource (for example, storage) becomes available, and then resume processing the command. You can change this behavior to ignoring the condition by using the NOSUSPEND option. For the condition NOSPACE, the normal default is to wait if processing a WRITEQ TS command, but to abend the task if processing a WRITEQ TD, WRITE, or REWRITE command. Coding the **WRITEQ TS** command with the NOSUSPEND option makes it ignore any NOSPACE condition that arises. For more information see WRITEQ TS .

CICS keeps a table of these conditions for each link level. Essentially, therefore, each program level has its own HANDLE state table governing its own condition handling.

This behavior is modified by HANDLE CONDITION ERROR and IGNORE CONDITION.

Using the HANDLE CONDITION ERROR command

The following example shows you how to trap unexpected errors with the HANDLE CONDITION ERROR command.

About this task

Figure 75 on page 196 shows the first of only two HANDLE CONDITION commands used in program ACCT01:

```
 PROCEDURE DIVISION.
*
* INITIALIZE.
* TRAP ANY UNEXPECTED ERRORS.
EXEC CICS HANDLE CONDITION
ERROR(OTHER-ERRORS)
END-EXEC.
*
```

Figure 75. Trapping the unexpected with the HANDLE CONDITION ERROR command

The HANDLE CONDITION ERROR command passes control to the paragraph at label OTHER-ERRORS if any condition arises for a command that does not specify NOHANDLE or RESP.

This command is the first command executed in the procedure division of this COBOL program. This is because a HANDLE CONDITION command must be processed before any CICS command is processed that can raise the condition being handled. However, your program does not see the effects when it processes the HANDLE CONDITION command; it only sees them later, if and when it issues a CICS command that raises one of the named conditions.

In this, and the other ACCT programs, you generally use the RESP option. All the commands that specify the RESP option are written with a "catch-all" test (`IF RESPONSE NOT = DFHRESP(NORMAL) GO TO OTHER-ERRORS`) **after** any explicit tests for specific conditions. Therefore, other than the exceptions that you specifically anticipate, any exceptions take control to the paragraph at OTHER-ERRORS in each program. Those relatively few commands that do not have RESP on them take control to the same place if they result in any condition other than NORMAL because of this HANDLE CONDITION ERROR command.

Using the IGNORE CONDITION command

You can use the IGNORE CONDITION command to specify that a program continues when a specific condition occurs (in contrast to the HANDLE CONDITION command, which specifies that control passes to a specified label when a specific condition occurs).

About this task

Restriction: This command is supported only in COBOL, PL/I, and assembler language applications (but not AMODE(64) assembler language applications). It is not supported in all other supported high level languages.

You can set up an IGNORE CONDITION command to ignore one or more of the conditions that can potentially arise on a command. The IGNORE CONDITION command means that no action is to be taken if a condition occurs; control returns to the instruction that follows the command and return codes are set in the EIB. The following example ignores the MAPFAIL condition:

```
 EXEC CICS IGNORE CONDITION MAPFAIL
END-EXEC.
```

When a single EXEC CICS command is processed, it might raise several conditions. For example, a file control command might be invalid and might also apply to a file that is not defined in the file control table. CICS checks these conditions and passes the first one that is not ignored (by your IGNORE CONDITION command) back to your application program. CICS passes back only one exception condition at a time to your application program.

An IGNORE CONDITION command for a given condition applies only to the program you put it in. It remains active while the program is running, or until a later HANDLE CONDITION command that names the same condition is met, in which case the IGNORE CONDITION command is overridden.

You can use an IGNORE CONDITION command for a program that reads records that might be longer than the space you provided, but you do not consider this as an error and do not want anything done about it. You might, therefore, code IGNORE CONDITION LENGERR before issuing READ commands.

You can also use an IGNORE CONDITION ERROR command to catch any condition considered as an error for which there is no currently active HANDLE CONDITION command that includes a label. When an error occurs, control is passed to the next statement and the program must check for return codes in the EIB. See “How CICS keeps track of what to do” on page 196 for examples of conditions that are not considered as errors.

You can also switch from ignoring a condition to handling it, or to using the system default action. For example, you could code:

```
* MIXED ERROR PROCESSING
EXEC CICS IGNORE CONDITION LENGERR
END-EXEC.

EXEC CICS HANDLE CONDITION DUPREC(DUPRTN)
LENGERR
```

```
ERROR(ERRHANDL)
END-EXEC.
```

Because this code initially ignores condition LENGERR, nothing happens if the program raises a LENGERR condition; the application continues its processing. Of course, if the fact that LENGERR has arisen means that the application cannot sensibly continue, you have a problem.

Later in the code, you can explicitly set condition LENGERR to the system default action by naming it in a HANDLE CONDITION command without a label. When this command has been executed, the program no longer ignores condition LENGERR, and if it subsequently occurs, it now causes the system default action. The point about mixing methods is that you can, and that each condition is treated separately.

You cannot code more than 16 conditions in the same command. You must specify any additional conditions in further IGNORE CONDITION commands.

Using the HANDLE ABEND command

The **HANDLE ABEND** command activates or reactivates a program-level abend exit in your application program. You can also use this command to cancel a previously activated exit.

For certain programming languages, you can use the HANDLE ABEND command to supply your own code to be executed when an abend is processed. This means that your application can cope with the abnormal situation in an orderly manner and continue to run. You provide the user exit programs and rely on CICS to call those programs when required.

The flow of control during abend processing is shown in Figure 76 on page 207.

Restrictions

- The **HANDLE ABEND** command does not apply to Java programs.
- Because exception conditions in C and C++ programs do not cause abends, you cannot use the **HANDLE ABEND** command in this way with these programs. However, the **HANDLE ABEND** command is supported in C and C++ when used with the PROGRAM option.
- You cannot use **HANDLE ABEND LABEL** in Assembler programs that do not use DFHEIENT and DFHEIRET. Assembler programs that use the Language Environment stub CEESTART should either use HANDLE ABEND PROGRAM or a Language Environment service such as CEEHDLR.
- You cannot use **HANDLE ABEND LABEL** in AMODE(64) programs.

Using PUSH HANDLE and POP HANDLE commands

The **PUSH HANDLE** and **POP HANDLE** commands suspend and reinstate, respectively, the effects of **HANDLE CONDITION** , **IGNORE CONDITION** , **HANDLE ABEND** , and **HANDLE AID** commands.

PUSH HANDLE
Suspends the current effect of **HANDLE CONDITION** , **IGNORE CONDITION** , **HANDLE ABEND** , and **HANDLE AID** commands.

POP HANDLE
Reinstates the effect of **HANDLE CONDITION** , **IGNORE CONDITION** , **HANDLE ABEND** , and **HANDLE AID** commands to what they were before the previous **PUSH HANDLE** was called.

Restriction: These commands are supported only in COBOL, PL/I, and assembler language applications (but not AMODE(64) assembler language applications). They are not supported for all other supported high level languages.

CICS also keeps a table of conditions for each **PUSH HANDLE** command that has not been countermanded by a matching **POP HANDLE** command.

When each condition occurs, CICS uses the following sequence of tests:

1. If the command has the RESP or NOHANDLE option, control returns to the next instruction in your application program. Otherwise, CICS scans the condition table.

2. If an entry for the condition exists, this entry determines the action.
3. If no entry exists and the default action for this condition is to suspend execution, the action is as follows:
   a. If the command has the NOSUSPEND or NOQUEUE option, control returns to the next instruction.
   b. If the command does not have the NOSUSPEND or NOQUEUE option, the task is suspended.
4. If no entry exists and the default action for this condition is to abend, CICS searches for the ERROR condition:
   a. If the ERROR condition is found, this entry determines the action.
   b. If ERROR cannot be found, the task is abended. You can choose to handle abends.

Note: The OVERFLOW condition on a **SEND MAP** command is an exception to the rules.

The **ALLOCATE** , **ENQ** , **GETMAIN** , **WRITE JOURNALNAME** , **WRITE JOURNALNUM** , **READQ TD** , and **WRITEQ TS** commands can all raise conditions for which the default action is to suspend your application program until the specified resource becomes available. Therefore, on these commands, you can use the NOSUSPEND option to inhibit this wait and return immediately to the next instruction in your application program.

Some conditions can occur during the execution of a number of unrelated commands. If you want the same action for all occurrences, code a single **HANDLE CONDITION** command at the start of your program.

Note: Because using RESP implies NOHANDLE, be careful when using RESP with the **RECEIVE** command, because it overrides the **HANDLE AID** command as well as the **HANDLE CONDITION** command. This means that function key responses are ignored, and is the reason for testing them earlier in the ACCT code. See “Using the HANDLE AID command” on page 401.

Parsing SOAP fault messages

If you use the **EXEC CICS INVOKE WEBSERVICE** or **EXEC CICS INVOKE SERVICE** commands to call a remote web service, you might receive an INVREQ response with a RESP2 value of 6, indicating that a SOAP fault message was returned.

Problem

You want to know more about a SOAP fault message from an application. For example, you want to know whether the fault message originated in CICS or in the remote application; or you want to access some embedded data in the fault message.

Response

You can use the XML parsing application programming interface (API) command, **TRANSFORM** . (Previously, you read the DFHWS-BODY container returned by CICS to access the XML representation of the SOAP fault message, then parsed the XML data by using a mechanism of your choice.) You can use a similar approach to process SOAP V1.2 fault messages.

Use DFHSC2LS to build COBOL bindings for SOAP faults

Download a copy of the XML schema for SOAP Envelopes from the following location: `http://schemas.xmlsoap.org/soap/envelope/`. For example, save the schema to a location in the UNIX file system called `/u/example/source/SOAP11.xsd`. Use DFHSC2LS to process the XML schema and create as output a set of COBOL bindings for the schema, and an XSDBind file in a bundle directory. You could use JCL similar to the following example:

```
//EXAMPLE EXEC DFHSC2LS,
//INPUT.SYSUT1 DD *
MAPPING-LEVEL=3.0
ELEMENTS=Body,Fault
SCHEMA=/u/example/source/SOAP11.xsd
```

```
LANG=COBOL
PDSLIB=//EXAMPLE.COBOL.LIBRARY
PDSMEM=SOAP11
XSDBIND=SOAP11.xsdbind
BUNDLE=/u/example/output/bundle/SOAP11
LOGFILE=/u/example/output/logfile.log
*/
```

DFHSC2LS creates several COBOL language structures, as follows:

- Bindings for the body of the SOAP envelope:

```
03 Body.
06 Body-num PIC S9(9) COMP-5 SYNC.
06 Body-cont PIC X(16).

01 SOAP1101-Body.
03 Body-xml-cont PIC X(16).
03 Body-xmlns-cont PIC X(16).
```

This language structure contains bindings to allow any number of XML tags to appear in the SOAP body. CICS stores the number of tags found in the `Body-num` field, and information about the data in the container named by the `Body-cont` field. Each XML tag from the body has two fields associated with it that provide the XML for the tag in a container named in `Body-xml-cont` and the in-scope XML namespace declarations in a container named in `Body-xmlns-cont`.

- Bindings for the fault in the body of the SOAP envelope:

```
03 Fault.
06 faultcode-length PIC S9999 COMP-5 SYNC.
06 faultcode PIC X(255).
06 faultstring-length PIC S9999 COMP-5 SYNC.
06 faultstring PIC X(255).
06 faultactor-num PIC S9(9) COMP-5 SYNC.
06 faultactor.
09 faultactor2-length PIC S9999 COMP-5 SYNC.
09 faultactor2 PIC X(255).
06 detail3-num PIC S9(9) COMP-5 SYNC.
06 detail2.
09 Xdetail-num PIC S9(9) COMP-5 SYNC.
09 Xdetail-cont PIC X(16).

01 SOAP1102-Xdetail.
03 detail-xml-cont PIC X(16).
03 detail-xmlns-cont PIC X(16).
```

This language structure contains bindings to allow a single SOAP fault to be parsed. It provides access to the `faultcode` , `faultstring` and `faultactor` fields, together with structures to map any number of XML tags found within the `detail` section of the SOAP fault.

Install the bundle into CICS

Create and install a BUNDLE definition such as the following example:

```
GROUP: EXAMPLE

DESCRIPTION: Bundle for mapping SOAP 1.1 SOAP Faults

BUNDLEDIR: /u/example/output/bundle/SOAP11
```

`BUNDLEDIR` points to the location that you specified by using the `BUNDLE` parameter of DFHSC2LS. If you run DFHSC2LS on a different z/OS image from the one that CICS uses, you might need to copy the bundle directory to the target machine. In this situation, use a different directory path and set the value of `BUNDLEDIR` accordingly. You can set any name for the bundle; it does not need to be S0AP11.

When the bundle is installed into CICS , you have a BUNDLE resource called SOAP11 and an XMLTRANSFORM resource also called SOAP11. The XMLTRANSFORM name is derived from the value of the XSDBIND parameter of DFHSC2LS.

Example SOAP fault message

The following example SOAP fault message might be found in the DFHWS-BODY container following an **EXEC CICS INVOKE WEBSERVICE** command:

```
SOAP-ENV:Body>
<SOAP-ENV:Fault xmlns="">
<faultcode>SOAP-ENV:Server</faultcode>
<faultstring>Conversion to SOAP failed</faultstring>
<detail>
<CICSFault xmlns="http://www.ibm.com/software/htp/cics/WSFault">
DFHPI1010 *** XML generation failed. A conversion error
INVALID_PACKED_DEC) occurred when converting field 'example' for
WEBSERVICE 'testWebservice'.
</CICSFault>
</detail>
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
```

This example is a fault message that CICS creates when a conversion error occurs. When the **TRANSFORM** command processes the fault message, `Body-num` is set to 1 to indicate that there is a single XML tag in the Body tag. `Body-cont` is set to the name of a container, for example DFHPICC-00000001.

The names of two further containers are place inside container DFHPICC-00000001, for example DFHPICC-00000002 and DFHPICC-00000003.

Container DFHPICC-00000002 contains the first tag in the Body tag, for example:

```
SOAP-ENV:Fault xmlns=">
<faultcode>SOAP-ENV:Server</faultcode>
<faultstring>Conversion to SOAP failed</faultstring>
<detail>
<CICSFault xmlns="http://www.ibm.com/software/htp/cics/WSFault">
DFHPI1010 *** XML generation failed. A conversion error
INVALID_PACKED_DEC) occurred when converting field 'example' for
WEBSERVICE 'testWebservice'.
</CICSFault>
</detail>
</SOAP-ENV:Fault>
```

Container DFHPICC-00000003 contains any in-scope namespace declarations, for example:

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance "
```

If the DFHPICC-00000002 container is then parsed through a second **EXEC CICS TRANSFORM** command, further output is created. The `faultcode` and `faultcode-length` fields are set to SOAP-ENV:Server and 15. The `faultstring` and `faultstring-length` fields are set to "Conversion to SOAP failed" and 25. The `faultactor-num` field is set to 0. The `detail3-num` field is set to 1 to indicate that the optional `detail` tag is present in the fault. The `detail2-num` field is set to 1 to indicate that there is one sub-tag in the optional `detail` tag. The `detail2-cont` field is set to the name of a container, for example DFHPICC-00000004.

Container DFHPICC-00000004 contains the names of two further containers, for example DFHPICC-00000005 and DFHPICC-00000006.

Container DFHPICC-00000005 contains the first XML tag found in the `detail` section of the SOAP fault, for example:

```
CICSFault
xmlns="http://www.ibm.com/software/htp/cics/WSFault ">
DFHPI1010 *** XML generation failed. A conversion error
(INVALID_PACKED_DEC) occurred when converting field 'example'
for WEBSERVICE 'testWebservice'.
</CICSFault>
```

Container DFHPICC-00000006 contains the in-scope namespace declarations, for example:

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "
```

Application Code

To implement an application to parse the SOAP fault, use the following procedure:

1. Call the **TRANSFORM** command to query the contents of the DFHWS-BODY container. For example:

   ```
   EXEC CICS TRANSFORM XMLTODATA CHANNEL(channel-name)
   XMLCONTAINER('DFHWS-BODY') NSCONTAINER('DFHWS-XMLNS')
   ELEMNAME(element-name) ELEMNAMELEN(element-name-len)
   END-EXEC
   ```

2. If the element-name is set to Body, parse the container. If it is not, an error occurred. To parse the Body, use the following commands:

   ```
   EXEC CICS TRANSFORM XMLTODATA CHANNEL(channel-name)
   XMLTRANSFORM('SOAP11') XMLCONTAINER('DFHWS-BODY')
   NSCONTAINER('DFHWS-XMLNS') DATCONTAINER('PARSEDBODY')
   END-EXEC

   EXEC CICS GET CONTAINER('PARSEDBODY') SET(body-ptr) END-EXEC
   ```

3. Address the parsed data. For example:

   ```
   SET ADDRESS OF Body TO body-ptr
   ```

4. Check Body-num to ensure there is at least one entry in the Body. If so, read the container that lists the details. For example:

   ```
   EXEC CICS GET CONTAINER(Body-cont) SET(body-cont-ptr)
   END-EXEC

   SET ADDRESS OF SOAP1101-Body TO body-cont-ptr
   ```

5. Call the **TRANSFORM** command a second time to query the first tag in the Body:

   ```
   EXEC CICS TRANSFORM XMLTODATA CHANNEL(channel-name)
   XMLCONTAINER(Body-xml-cont) NSCONTAINER(Body-xmlns-cont)
   ELEMNAME(element-name) ELEMNAMELEN(element-name-len)
   END-EXEC
   ```

6. If the element-name is set to Fault, parse the container:

   ```
   EXEC CICS TRANSFORM XMLTODATA CHANNEL(channel-name)
   XMLTRANSFORM('SOAP11') XMLCONTAINER(Body-xml-cont)
   NSCONTAINER(Body-xmlns-cont) DATCONTAINER('PARSEDFAULT') END-EXEC

   EXEC CICS GET CONTAINER('PARSEDFAULT') SET(fault-ptr) END-EXEC

   SET ADDRESS OF Fault TO fault-ptr
   ```

7. You can now query the data from the fault. For example, you might find the `faultstring` field useful. To parse application specific details from the detail' section of the fault, you an build further application-specific COBOL bindings by using DFHSC2LS and issuing further **TRANSFORM** commands in the application.

Note: You can combine steps 1 and 2 into a single **TRANSFORM** command (and also combine steps 5 and 6).

Avoiding the storm drain effect

If an application avoids making calls to resource managers because it knows the connection to the resource manager is not active, or it processes an error return code as a result of the connection being

unavailable, and proceeds to issue an error message and return normally rather than abend, it could delude the workload manager into routing more work to the CICS region. This situation is called the *storm drain effect*.

Because the application did not abend, the workload manager believes that good response times are being achieved by this CICS region for work involving the resource manager, and therefore routes more work down the *storm drain*. This situation can arise for applications that use connections from CICS to Db2, IMS , IBM MQ and VSAM RLS.

To avoid the *storm drain effect*, the resource manager adapters need to inform the z/OS Workload Manager that the request has failed. This information can then in turn be used by CICSPlex SM Workload Management. See Abend probabilities and workload management

The resource manager adapters for the following resources inform z/OS Workload Manager to allow CICSPlex SM Workload Management to use its abend probability functionality, if activated, to avoid routing more work to the affected CICS region:

Db2

For connections between CICS and Db2 at the time of returning the -923 SQL return code, the CICS Db2 attachment facility informs the z/OS Workload Manager that the request has failed. The DB2CONN definition should be configured with attributes STANDBYMODE=RECONNECT and CONNECTERROR=SQLCODE. Applications should unconditionally issue EXEC SQL requests and test for an SQLCODE of -923 rather than using an EXEC CICS EXTRACT EXIT or EXEC CICS INQUIRE EXITPROGRAM command beforehand to test to see if CICS is connected to Db2. Only when an -923 SQLCODE is returned will z/OS Workload Manager be informed.

IMS

For connections between CICS and IMS, if the interface between CICS and IMS DBCTL is not active, applications that issue **CALLDLI** requests receive return codes 08FF in the UIB. At this time, the CICS-DBCTL interface informs the z/OS Workload Manager that the request has failed. For **EXEC DLI** requests, typically the request results in an application abend, unless the **NODHABEND** keyword has been used. In this situation, z/OS Workload Manager is also informed in order to avoid the storm drain effect.

IBM MQ

For connections between CICS and IBM MQ, if CICS has never been connected to a queue manager, then the request is rejected prior to entering the CICS-MQ adapter, so it is not possible to avoid the storm drain effect.

When the CICS-MQ adapter is active, if a subsequent connection failure occurs and any of the following MQI reason codes is returned, then z/OS Workload Manager is informed that the request has failed:

- mqrc_connection_broken
- mqrc_q_mgr_quiescing
- mqrc_connection_quiescing
- mqrc_connection_not_authorized
- mqrc_q_mgr_name_error
- mqrc_q_mgr_not_available
- mqrc_q_mgr_stopping
- mqrc_connection_stopping
- mqrc_adapter_not_available

VSAM RLS

For connections between CICS and VSAM RLS, if CICS receives a response indicating an RLS failure, or RLS is disabled, or a previous RLS failure has occurred, the normal result is an AFCR, AFCS, or AFCT abend

respectively. In case the abend is handled, CICS File Control informs z/OS Workload Manager that the request has failed.

Abnormal termination recovery

CICS provides a program-level abend exit facility so that you can write your own exits (programs or routines) that can receive control during abnormal termination of a task. An example of a function performed by such an abend exit is the clean up of a program that has started but has not completed normally.

Some causes of abnormal terminations are as follows:

- A user request, for example, by the following code:

```
EXEC CICS ABEND ABCODE(...)
```

- A CICS request as a result of an invalid user request. For example, an invalid FREEMAIN request results in the transaction abend code ASCF.
- A program check. The system recovery program (DFHSRP) is driven and the task abends with code ASRA.
- An operating system abend. The DFHSRP program is driven and the task abends with code ASRB.
- A looping task. The DFHSRP program is driven and the task abends with code AICA.

Note: If an ASRB or ASRA is detected in CICS code, CICS produces a dump before calling your HANDLE ABEND exit.

See Troubleshooting and support for full details about fixing problems. See CICS messages for information about the transaction abend codes for abnormal terminations that are initiated by CICS , their meanings, and your responses.

The **HANDLE ABEND** command activates or reactivates a program-level abend exit in your application program; you can also use this command to cancel a previously activated exit.

When activating an exit, you must do one of the following:

- Use the PROGRAM option to specify the name of a program to receive control.
- For COBOL or assembler applications only (but not AMODE(64) assembler applications), use the LABEL option to specify a routine label to which control branches when an abnormal termination condition occurs.

  You cannot use the **HANDLE ABEND LABEL** command with C, C++, PL/I or AMODE(64) applications. In PL/I, the equivalent is using an ON ERROR block.

A **HANDLE ABEND** command overrides any preceding such command in any application program at the same logical level. Each application program of a transaction can have its own abend exit, but only one abend exit at each logical level can be active. (Logical levels are explained in “Program control” on page 146 .)

When a task terminates abnormally, CICS searches for an active abend exit, starting at the logical level of the application program in which the abend occurred, and proceeding to successively higher levels. The first active abend exit found, if any, is given control. This procedure is shown in Figure 76 on page 207 , which also shows how subsequent abend processing is determined by the user-written abend exit.

If CICS finds no abend exit, it passes control to the abnormal condition program to terminate the task abnormally. This program invokes the user replaceable program error program, DFHPEP. For programming information about how to customize DFHPEP, see Writing a program error program.

CICS deactivates the exit upon entry to the exit routine or program to prevent recursive abends in an abend exit. If you want to try the operation again, you can branch to a point in the program that was in control at the time of the abend and issue a **HANDLE ABEND RESET** command to reactivate the abend exit. You can also use this command to reactivate an abend exit (at the logical level of the issuing program) that was canceled previously by a **HANDLE ABEND CANCEL** command. You can suspend the

HANDLE ABEND command with the **PUSH HANDLE** and **POP HANDLE** commands, as described in "Using PUSH HANDLE and POP HANDLE commands" on page 198.

When an abend is handled, the dynamic transaction backout program is not invoked. If you need the dynamic transaction backout program, you take an implicit or explicit sync point, or issue SYNCPOINT ROLLBACK, or issue an **ABEND** command.

Where the abend is the result of a failure in a transaction running in an IRC-connected system, for example AZI2, the sync point processing can abend ASP1 if it attempts to use the same IRC connection during its backout processing.

The **HANDLE ABEND** command cannot intercept ASPx or APSJ abend codes.

Creating a program-level abend program or routine

You can create a program-level abend program. For some programming languages, you can create a program-level abend routine.

Program-level abend program

You can define an abend program by using RDO or by using the program autoinstall exit.

If you use the autoinstall method, the program definition is not available at the time of the HANDLE ABEND. This might mean that a program functions differently the first time it is invoked. If the program is not defined at the time the HANDLE ABEND is issued, and program autoinstall is active, the security check on the name of the program is the only one which takes place. Other checks occur at the time the abend program is invoked. If the autoinstall fails, the task abends APCT and control is passed to the next higher level.

Abend exit programs can be coded in any supported language.

On entry to an abend exit program, no addressability can be assumed other than that normally assumed for any application program coded in that language.

Program-level abend routine

Restriction: This section does not apply to C, C++, PL/I, and AMODE(64) applications, because the **HANDLE ABEND LABEL** command is not supported.

Abend exit routines must be coded in the same language as their program.

For abend exit routines, the addressing mode and execution key are set to the addressing mode and execution key in which the **HANDLE ABEND** command has been issued.

On entry to an abend exit routine, the register values are as follows:

COBOL
Control returns to the **HANDLE ABEND** command with the registers restored; a COBOL GOTO is then executed.

Assembler (but not AMODE(64) assembler)

Reg 15
Abend label.

Reg 0-14
Contents at the time of the last CICS service request.

Termination of processing

You can terminate processing in an abend exit routine or program in one of the following ways. When abend routines and programs are called by CICS internal logic, they should terminate with an abend because further processing will probably cause more problems.

- Use a **RETURN** command to indicate that the task continues to run with control passed to the program on the next higher logical level. If no such program exists, the task is terminated normally, and any recoverable resources are committed.
- Use an **ABEND** command to indicate that the task is abnormally terminated with control passed either to an abend exit specified for a program on a higher logical level or, if there is not one, to the abnormal condition program for abnormal termination processing.
- Branch to retry an operation. When you use this method to retry an operation, and you want to reenter the original abend exit routine or program if a second failure occurs, the abend exit routine or program should issue the **HANDLE ABEND RESET** command before branching. This is because CICS has disabled the exit routine or program to prevent it from reentering the abend exit.

For an abend that is caused by a timeout on an outstanding **RECEIVE** command, it is important to let the CICS abend continue, so that CICS can cancel the **RECEIVE**.

Retrying operations

If an abend occurs during the invocation of a CICS service, issuing a further request for the same service can cause unpredictable results. Because the reinitialization of pointers and work areas, and the freeing of storage areas in the exit routine, might not have been completed.

You should not try to recover from ATNI or ATND abends by attempting further I/O operations. Either of these abends results in a TERMERR condition, requiring the session to be terminated in all cases. You should not try to issue terminal control commands while recovering from an AZCT abend, or an AZIG abend, as CICS has not fully cleaned up from the RTIMOUT, and an indefinite wait can occur.

If intersystem communication is being used, an abend in the remote system might cause a branch to the specified program or label, but subsequent requests to use the same resource in the remote system might fail. If an abend occurs as a result of a failure in the connection to the remote system, subsequent requests to use *any* resources in the remote system might fail.

If an abend occurs as a result of a BMS command, control blocks are not tidied up before control is returned to the BMS program, and results are unpredictable if the command is retried.

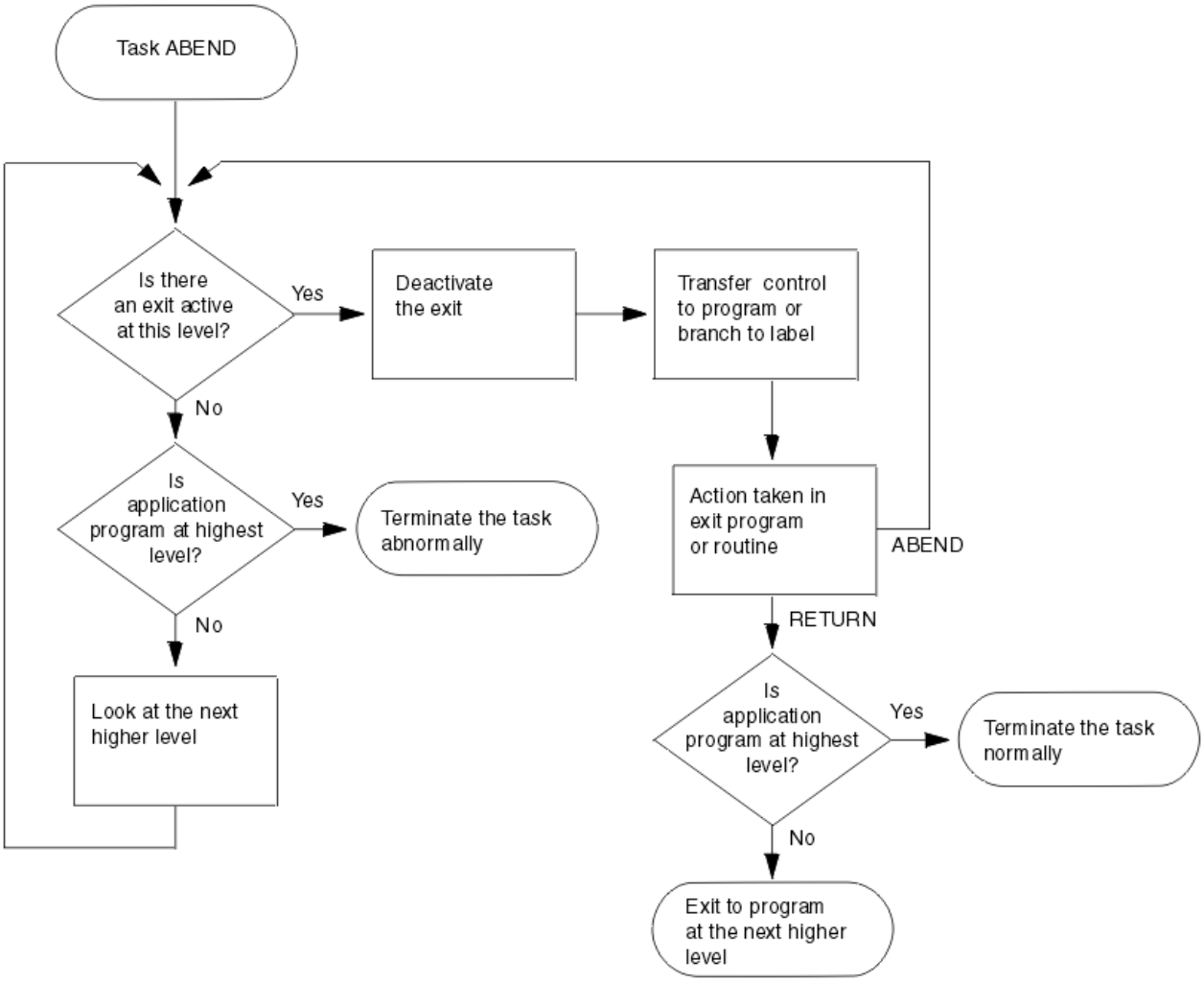


Figure 76. ABEND exit processing

Monitoring application performance

CICS monitoring provides information about the performance of your application transactions. Use the MONITOR command for user event monitoring points.

In addition to the monitoring data collected from a system defined elsewhere, monitoring points (EMPs) in CICS, a user application program can contribute data to user fields in the CICS monitoring records. You can do this by using the MONITOR POINT command to invoke user-defined EMPs. At each of these EMPs, you can add or change up to 16384 bytes of your own data in each performance monitoring record. In those 16384 bytes, you can have any combination of the following:

- In the range 0 - 256 counters
- In the range 0 - 256 clocks
- A single 8192-byte character string

For example, you could use these user EMPs to count the number of times a certain event occurs, or to time the interval between two events.

Determining a user's access to resources by using the QUERY SECURITY command

You can use the **EXEC CICS QUERY SECURITY** command to determine whether a terminal user has access to resources that are defined to an external security manager such as RACF.

You can use **QUERY SECURITY** to query the security authorization of the user that invokes a transaction containing the **QUERY SECURITY** command. In this case, the USERID is not specified.

You can also allow a transaction containing the **QUERY SECURITY** command and running under one user ID to query the security authorization of another user ID. In this case, the user ID that is subject to the query must be specified in the USERID option of the **QUERY SECURITY** command. Note that CICS performs a surrogate user check to verify whether the user invoking the transaction is authorized to the user specified in USERID.

In response to a **QUERY SECURITY** command, CICS returns information about the user's security authorizations. CICS obtains this information from the external security manager. You can code the application to proceed in different ways depending on the user's permitted accesses.

You specify the type of resource that you are querying by the CICS resource type name. For example, if you want to query a user's authorization to access a file, you can specify RESTYPE('FILE'). To identify a particular file within the type, you specify the **RESID** parameter.

A typical use case: Controlling access to data at the record or field level

The normal CICS resource security checking for file resources, for example, works only at the file level. To control access to individual records, or even fields within records, you can use **QUERY SECURITY**. For this purpose, your security administrator must define resource profile names, with appropriate access authorizations, for the records or fields that you want to protect. These profiles are defined in user resource classes defined by the administrator, not in CICS resource classes.

To query these classes and resources, the **QUERY SECURITY** command uses the RESCLASS and RESID options (RESCLASS and RESTYPE are mutually exclusive options). You can use the CVDA values returned by **QUERY SECURITY** to determine whether to access the record or field.

Programming hint

A transaction can use the **QUERY SECURITY** command to query a number of resources in order to prepare a list of resources to which the terminal user has access. The use of this technique could generate up to four resource violation messages for each query on a resource that the transaction is not authorized to access. These messages appear on the system console, the CSCS TD queue, and the SMF log data set. If you want to suppress these messages, code NOLOG in the **QUERY SECURITY** command.

Other considerations

CICS-defined resource identifiers

In all cases except for the SPCOMMAND resource type, the resource identifiers are user-defined. However, for the SPCOMMAND type, the identifiers are fixed by CICS. RESID values details the possible RESID values for the SPCOMMAND resource type.

SEC system initialization parameter

The setting of the **SEC** system initialization parameter affects the CVDA values returned by the **QUERY SECURITY** command.

Find out more

Security checking using the QUERY SECURITY command gives you details on how to use **QUERY SECURITY** in an application program to determine the level of access that a user has to a particular resource.

Designing applications to use the user-defined resources

This topic gives an example of how you might design applications to make use of the user-defined resources.

Your applications use CICS file control in the normal way to read records from the pay and personal details file. Because you are controlling individual fields within each record, you may not need to apply resource security at the file level, so your transactions can be defined with RESSEC(NO). After reading the file record, but before displaying the results, you use QUERY SECURITY to determine whether the user has the authority to access the particular field within the record. For instance, before displaying the salary amount, you issue:

```
EXEC CICS QUERY SECURITY RESCLASS('$FILEREC')
                         RESID('PAYFILE.SALARY')
                         RESIDLENGTH(14)
                         READ(read_cvda)
```

Then, depending on the value returned in `read_cvda`, your application either displays the salary or a message stating that the user is not authorized to display it. Likewise, as part of a transaction that updates a person's telephone number, you issue:

```
EXEC CICS QUERY SECURITY RESCLASS('$FILEREC')
                         RESID('PERSONAL.PHONE')
                         RESIDLENGTH(14)
                         UPDATE(update_cvda)
```

If the value returned in `update_cvda` indicates that the user has UPDATE access, the transaction can continue and update the telephone number in the file. Otherwise, it should indicate that the user is not authorized to update the telephone number.

CICS intercommunication

The areas to consider when you write applications that communicate with other CICS systems are summarized.

For further information about CICS intercommunication, see Getting started with intercommunication.

You can run application programs in a CICS intercommunication environment using one or more of the following facilities:

Transaction routing
Transaction routing enables a terminal in one CICS system to run a transaction in another CICS system. See “Transaction routing” on page 210.

Function shipping
Function shipping enables your application program to access resources in another CICS system. See “Function shipping” on page 210.

Distributed program link (DPL)
DPL enables an application program running in one CICS region to link to another application program running in a remote CICS region. See “Distributed program link (DPL)” on page 211.

Asynchronous processing
Asynchronous processing enables a CICS transaction to start another transaction in a remote system and optionally pass data to it. See “Asynchronous processing” on page 222.

Distributed transaction processing (DTP)
DTP enables a CICS transaction to communicate with a transaction running in another system. Two interfaces are available for DTP; command-level EXEC CICS and the SAA interface for DTP, known as Common Programming Interface Communications (CPI Communications). See “Distributed transaction processing (DTP)” on page 222.

Common Programming Interface Communications (CPI-C)
CPI-C provides DTP on APPC connections and defines an API that can be used on multiple system platforms. See “Common Programming Interface Communications (CPI Communications)” on page 223.

External CICS interface (EXCI)
EXCI enables a non-CICS program running in MVS to allocate and open sessions to a CICS system, and to issue DPL requests on these sessions. CICS supports MVS resource recovery services (RRS) in applications that use the external CICS interface. See “External CICS interface (EXCI)” on page 223.

For details about the intercommunication aspects of the CICS Front End Programming Interface (FEPI), see Introduction to FEPI.

Design considerations

If your application program uses more than one of the facilities listed earlier, you must allow for the design considerations for each one. Also, if your program uses more than one intersystem session for distributed transaction processing, it must control each session according to the rules for that type of session.

Programming language

Generally, you can use COBOL, C, C++, PL/I, or assembler language to write application programs that use CICS intercommunication facilities. However, for DTP application programs that hold APPC unmapped conversations using the EXEC CICS API, you can use only C, C++, or assembler language.

Transaction routing

Transactions that can be invoked from a terminal owned by another CICS system, or that can acquire a terminal owned by another CICS system during transaction initiation, must be able to run in a transaction routing environment.

Generally, you can design and code such a transaction just like one used in a local environment. However, there are a few restrictions related to basic mapping support (BMS), pseudoconversational transactions, and the terminal on which your transaction is to run. All programs, tables, and maps that are used by a transaction ***must*** reside on the system that owns the transaction. (You can duplicate them in as many systems as you need.)

Some CICS transactions are related to one another, for example, through common access to the CWA or through shared storage acquired using a GETMAIN command. When this is true, the system programmer must ensure that these transactions are routed to the same CICS system. You should avoid (where possible) any techniques that might create inter-transaction affinities that could adversely affect your ability to perform dynamic transaction routing.

To help you identify potential problems with programs that issue these commands, you can use the CICS Interdependency Analyzer. For more information about this utility, see CICS Interdependency Analyzer for z/OS Overview . For more information about transaction affinity, see “Affinity” on page 156.

When a request to process a transaction is transmitted from one CICS system to another, transaction identifiers can be translated from local names to remote names. However, a transaction identifier specified in a RETURN command is not translated when it is transmitted from the transaction-owning system to the terminal-owning system.

Function shipping

You code a program to access resources in a remote system in much the same way as if they were on the local system.

You can use:

DL/I calls (EXEC DLI commands)
to access data associated with a remote CICS system.

File control commands
to access files on remote systems. Note that requests which contain the TOKEN keyword cannot be function-shipped.

Temporary storage commands
to access data from temporary storage queues on remote systems.

Transient data commands
to access transient data queues on remote systems.

Three additional exception conditions can occur with remote resources. They occur if the remote system is unavailable (SYSIDERR), if a request is invalid (ISCINVREQ), or if the mirror transaction abends (AIPM abend for IPIC, ATNI for ISC connections, and AZI6 for MRO).

Distributed program link (DPL)

The distributed program link function enables a CICS program (the client program) to call another CICS program (the server program) in a remote CICS region.

There are several reasons why you might want to design your application to use distributed program link. Some of these are:

- To separate the end-user interface (for example, BMS screen handling) from the application business logic, such as accessing and processing data, to enable parts of the applications to be ported from host to workstation more readily
- To obtain performance benefits from running programs closer to the resources they access, and thus reduce the need for repeated function shipping requests
- To offer a simple alternative, in many cases, to writing distributed transaction processing (DTP) applications

There are several ways in which you can specify that the program to which an application is linking is remote:

1. By specifying the remote system name on a LINK command
2. By specifying the remote system name on the installed program resource definition.
3. By specifying the remote system name using the dynamic routing program (if the installed program definition specifies DYNAMIC(YES) or there is no installed program definition)
4. By specifying the remote system name in a XPCREQ global user exit

The basic flow in distributed program link is described in CICS distributed program link . The following terms, illustrated in Figure 77 on page 212 , are used in the discussion of distributed program link:

Client region
The CICS region running an application program that issues a link to a program in another CICS region.

Server region
The CICS region to which a client region ships a link request.

Client program
The application program that issues a remote link request.

Server program
The application program specified on the link request, and which is executed in the server region.

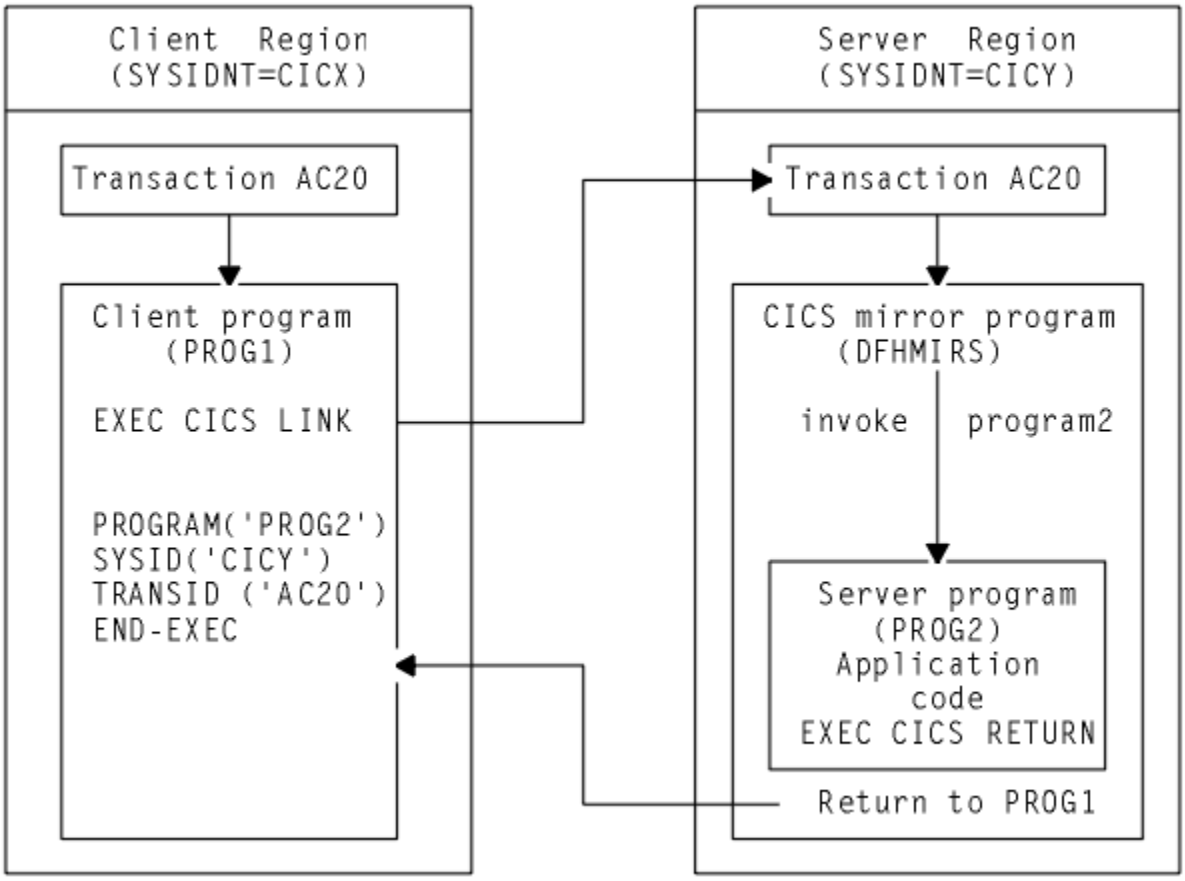


Figure 77. Illustration of distributed program link

Using the distributed program link function

Use these tables to understand the options on the distributed program link function.

You can specify:

- The name of the remote system (the server region).
- The name of the server program, if it is known by a different name in the server region.
- That you want to run the linked program locally, but restrict it to the distributed program link subset of the application programming interface (API) for testing purposes. (Server programs cannot use the entire CICS API when executed remotely; the restrictions are listed in Table 25 on page 221.
- That the server program takes a syncpoint independently from the client.
- The name of the transaction you want the program to run under in the server region.
- The data length of the COMMAREA being passed.

A server program can itself issue a distributed program link and act as a client program with respect to the program it links to.

The options shown in Table 23 on page 213 are used on the LINK command, and the options shown in Table 24 on page 213 are used in the PROGRAM resource definition in support of the distributed program link facility.

Table 23. Options on LINK command to support DPL	
Keyword	**Description**
DATALENGTH	Specifies the length of the contiguous area of storage (from the start of the COMMAREA) that the application is sending to a server program.
LENGTH	Specifies a halfword binary value that is the length in bytes of the COMMAREA. Specify a valid value (not zero) for the LENGTH option, because a length of zero when used on a DPL request can lead to unpredictable results and possibly a program failure.
SYSID	Specifies the name of the connection to the server region to which you want the client region to ship the program link request. **Note:** A remote SYSID specified on the LINK command overrides a REMOTESYSTEM name specified on the program resource definition or a sysid returned by the dynamic routing program.
SYNCONRETURN	Specifies that you want the server region to take a sync point on successful completion of the server program. **Note:** This option is unique to the LINK command and cannot be specified on the program resource definition.
TRANSID	Specifies the name of the transaction that the server region is to attach for execution of the server program. **Note:** TRANSID specified on the LINK command overrides any TRANSID specified on the program resource definition.

Note: Programming information, including the full syntax of the LINK command, is in CICS command summary , but note that for a distributed program link you cannot specify the INPUTMSG or INPUTMSGLEN options.

Table 24. Options in the PROGRAM resource definition to support DPL	
Keyword	**Description**
REMOTESYSTEM	Specifies the name of the connection to the server region (SYSID) to which you want the client region to ship the program link request.
REMOTENAME	Specifies the name by which the program is known in the server region (if different from the local name).
DYNAMIC	Specifies whether the program link request can be dynamically routed. For detailed information about the dynamic routing of DPL requests, see Dynamically routing DPL requests.
EXECUTIONSET	Specifies whether the program is restricted to the distributed program link subset of the CICS API. **Note:** This option is unique to the program definition and cannot be specified on the LINK command.

Table 24. Options in the PROGRAM resource definition to support DPL (continued)	
Keyword	**Description**
TRANSID	Specifies the name of the transaction that the server region is to attach for execution of the server program.

Examples of distributed program link

A COBOL example of a distributed program link command shows the syntax of the command.

Important: If the SYSID option of the LINK command specifies the name of a remote region, any REMOTESYSTEM, REMOTENAME, or TRANSID attributes specified on the program definition, or returned by the dynamic routing program, have no effect.

```
EXEC CICS LINK PROGRAM('DPLPROG')
 1
COMMAREA(DPLPRO-DATA-AREA)
 2
LENGTH(24000)
 2
DATALENGTH(100)
 2
SYSID('CICR')
 3
TRANSID('AC20')
 4
SYNCONRETURN
 5
```

Figure 78. COBOL example of a distributed program link

1. The program name of the server program.

   A program might have different names in the client and server regions. The name you specify on the LINK command depends on whether you specify the SYSID option.

   If you specify the name of a remote region on the SYSID option of the LINK command, CICS ships the link request to the server region without reference to the REMOTENAME attribute of the program resource definition in the client region, or to any program name returned by the dynamic routing program. In this situation, the PROGRAM name you specify on the LINK command must be the name by which the program is known in the server region.

   If you do not specify the SYSID option on the LINK command, or you specify the name of the local client region, the PROGRAM name you specify on the LINK command must be the name by which the program is known in the client region. CICS looks up the program resource definition in the client region. Assuming that the REMOTESYSTEM option of the installed program definition specifies the name of a remote region, the name of the server program on the remote region is obtained from:

   a. The REMOTENAME attribute of the program definition

   b. If REMOTENAME is not specified, the PROGRAM option of the LINK command.

   If the program definition specifies DYNAMIC(YES), or there is no installed program definition, the dynamic routing program is invoked and can accept or change the name of the server program.

2. The communication data area (COMMAREA).

   To improve performance, you can specify the DATALENGTH option on the LINK command. You can use this option to specify the amount of COMMAREA data you want the client region to pass to the server program. Typically, you use this option when a large COMMAREA is required to hold data that the server program is to return to the client program, but only a small amount of data needs to be sent to the server program by the client program, as in the example.

   If more than one server program updates the same COMMAREA before it is passed back to the client program, use the DATALENGTH option to specify the length of the COMMAREA. Ensure that if any of the server programs use an XCTL command to pass the COMMAREA to the next server program, they

specify the same length and address for it. This ensures that the original COMMAREA is returned to the client program. If a different length or address are specified, the invoked program will receive a copy of the COMMAREA, rather than the original COMMAREA, and so the original COMMAREA will not be returned to the client program. “Passing data to other programs by using COMMAREA” on page 149 has more information about using COMMAREAs to pass data to other programs.

Ensure that you specify a valid value (not zero) for the LENGTH option, because a length of zero when used on a DPL request can lead to unpredictable results and possibly a program failure.

When you use a COMMAREA, the program that is linked to must verify that the EIBCALEN field in the EIB of the task matches what the program expects. For more information, see “Passing data to other programs by using COMMAREA” on page 149.

3. The remote system ID (SYSID).

   You can specify the 4-character name of the server region to which you want the application region to ship a program link request using any of the following:

   - The SYSID option of the LINK command
   - The REMOTESYSTEM option of the program resource definition
   - The dynamic routing program.

   The rules of precedence are as follows:

   a. If the SYSID option on the EXEC CICS LINK command specifies a remote CICS region, CICS ships the request to the remote region.

      If the program definition specifies DYNAMIC(YES), or there is no program definition (the dynamic routing program is invoked for notification only) it cannot re-route the request.

   b. If the SYSID option is not specified or specifies the same name as the local CICS region:

      i) If the program definition specifies DYNAMIC(YES), or there is no installed program definition, the dynamic routing program is invoked, and can route the request.

      The REMOTESYSTEM option of the program definition, if specified, names the default server region passed to the dynamic routing program.

      Note: If the REMOTESYSTEM option names a remote region, the dynamic routing program cannot route the request locally.

      ii) If the program definition specifies DYNAMIC(NO), CICS ships the request to the remote system named on the REMOTESYSTEM option. If REMOTESYSTEM is not specified, CICS runs the program locally.

   The name you specify is the name of the connection definition installed in the client region defining the connection with the server region. (CICS uses the connection name in a table lookup to obtain the netname (z/OS Communications Server APPLID) of the server region.) The name of the server region you specify can be the name of the client region, in which case the program is run locally.

   If the server region cannot load or run the requested program (DPLPROG in our example), CICS returns the PGMIDERR condition to the client program in response to the link request. Note that EIBRESP2 values are not returned over the link for a distributed program link request where the error is detected in the server region. For errors detected in the client region, EIBRESP2 values are returned.

   You can also specify, or modify, the name of a server region in an XPCREQ global user exit program. See Enabling for specific invocation types for programming information about the XPCREQ global user exit point.

4. The remote transaction (TRANSID) to be attached.

   The TRANSID option is available on both the LINK command and the program resource definition. This enables you to tell the server region the transaction identifier to use when it attaches the mirror task under which the server program runs. If you specify the TRANSID option, you must define the transaction in the server region, and associate it with the supplied mirror program, DFHMIRS. This option allows you to specify your own attributes on the transaction definition for the purpose of

performance and fine tuning. For example, you could vary the task priority and transaction class attributes.

If the installed program definition specifies DYNAMIC(YES), or there is no installed program definition, the dynamic routing program is invoked and (provided that the SYSID option of the LINK command did not name a remote region) can change the value of the TRANSID attribute.

The order of precedence is:

a. If the SYSID option of the LINK command specified a remote region, a TRANSID supplied on the LINK
b. A TRANSID supplied by the dynamic routing program
c. A TRANSID supplied on the LINK command
d. The TRANSID attribute of the program definition.
e. The mirror TRANSID, CSMI.

You are recommended to specify the transaction identifier of the client program as the transaction identifier for the server program. This enables any statistics and monitoring data you collect to be correlated correctly under the same transaction.

The transaction identifier used on a distributed link program request is passed to the server program as follows:

- If you specify your own transaction identifier for the distributed link program request, this is passed to the server program in the EIBTRNID field of the EIB.
- EIBTRNID is set to the TRANSID value as specified in the DPL API or server resource definition. Otherwise, it defaults to the client's transaction code, which is the same value that is in the client's EIBTRNID.

Note: If the mirror transaction accesses Db2, EIBTRNID is used to determine which DB2ENTRY definition to use.

5. The SYNCONRETURN option for the server program.

   When you specify the SYNCONRETURN option, it means that the resources on the server are committed in a separate logical unit of work immediately before returning control to the client; that is, an implicit syncpoint is issued for the server just before the server returns control to the client. Figure 79 on page 217 provides an example of using distributed program link with the SYNCONRETURN option. The SYNCONRETURN option is intended for use when the client program is not updating any recoverable resources, for example, when performing screen handling. However, if the client does have recoverable resources, they are not committed at this point. They are committed when the client itself reaches a syncpoint or in the implicit syncpoint at client task end. You must ensure that the client and server programs are designed correctly for this purpose, and that you are not risking data integrity. For example, if your client program has shipped data to the server that results in the server updating a database owned by the server region, you only specify an independent syncpoint if it is safe to do so, and when there is no dependency on what happens in the client program. This option has no effect if the server program runs locally in the client region unless EXECUTIONSET(DPLSUBSET) is specified. In this case, the syncpoint rules governing a local link apply.

   Without the SYNCONRETURN option, the client commits the logical unit of work for both the client and the server resources, with either explicit commands or the implicit syncpoint at task end. Thus, in this case, the server resources are committed as part of the same syncpoint when the client resources are committed. Figure 80 on page 218 shows an example of using distributed program link without the SYNCONRETURN option.

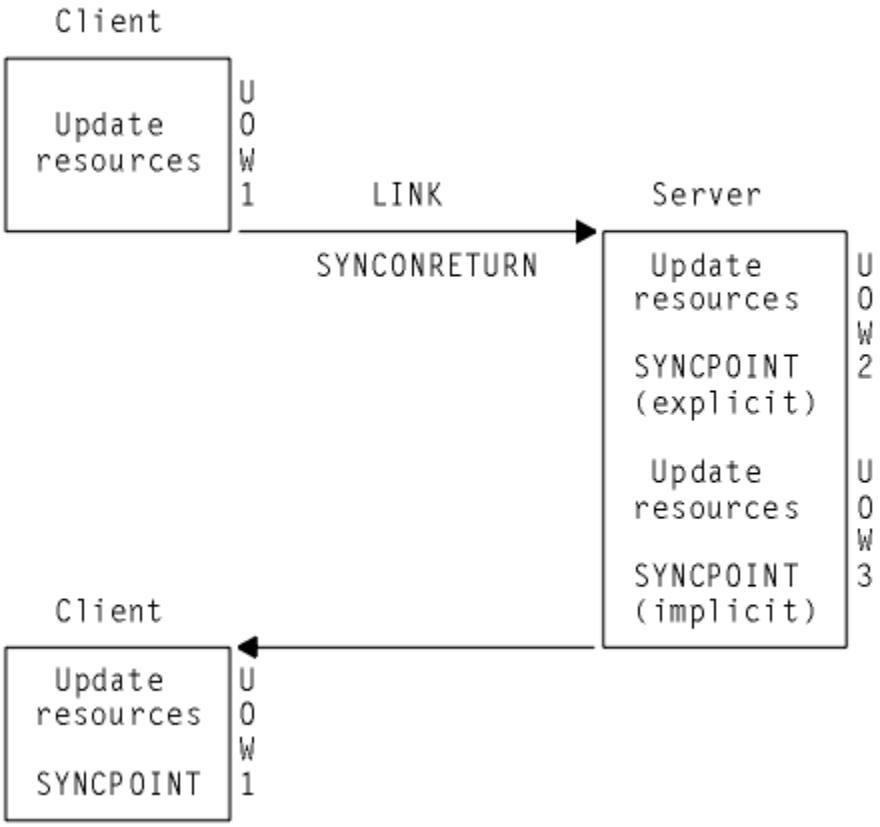


Figure 79. Using distributed program link with the SYNCONRETURN option

Note: This includes three logical units of work: one for the client and two for the server. The client resources are committed separately from the server.

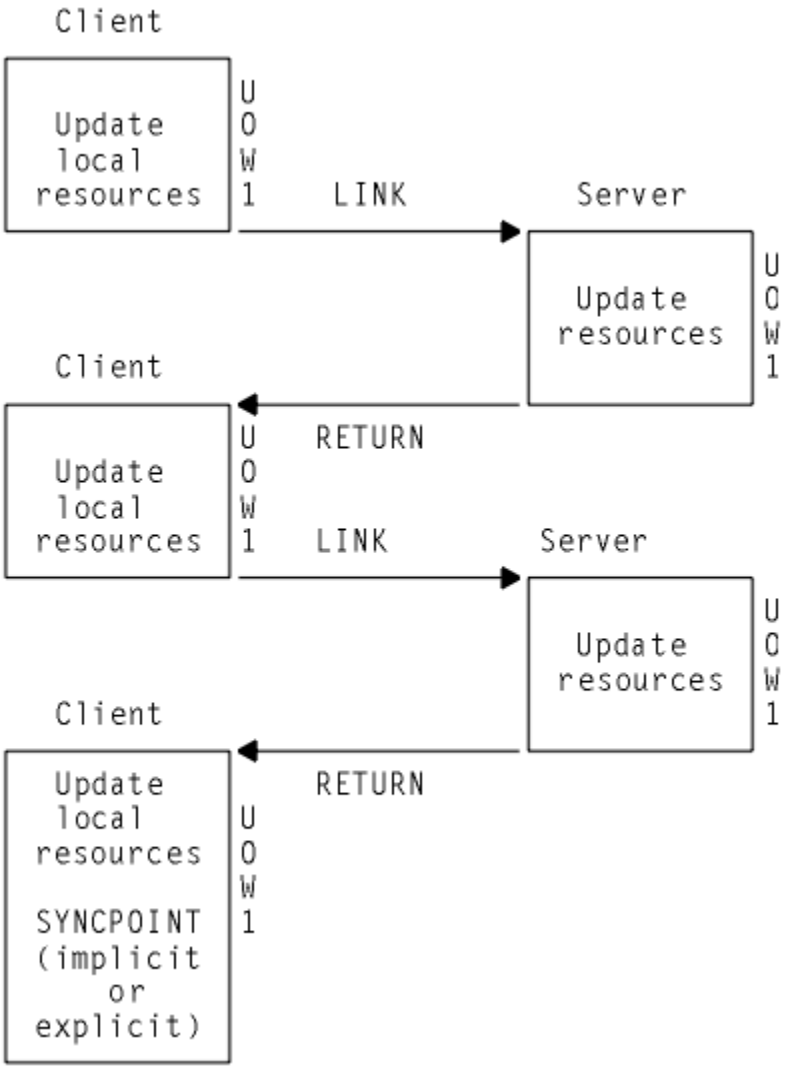


Figure 80. Using distributed program link without the SYNCONRETURN option

Note: The implicit or explicit syncpoint causes all client and server resources to be committed. There is only one logical unit of work because the client is responsible for determining when both the client and server resources are committed.

You need to consider the case when the client has a HANDLE ABEND command. When the client is handling abends in the server, the client gets control when the server abends. This is also true when the SYNCONRETURN option has been specified on the LINK command. In this case, it is recommended that the client issues an abend after doing the minimum of clean up. This causes both the client logical unit of work and the server logical unit of work to be backed out.

Programming considerations for distributed program link

Consider the following factors when writing application programs that use distributed program link.

Issuing multiple distributed program links from the same client task

A client task cannot request distributed program links to a single CICS server region using more than one transaction code in a single client unit of work unless the SYNCONRETURN option is specified. It can issue multiple distributed program links to one CICS server system with the same or the default transaction code.

Sharing resources between client and server programs

The server program does not have access to the lifetime storage of tasks on the client, for example, the TWA. Nor does it necessarily have access to the resources that the client program is using, for example, files, unless the file requests are being function shipped.

Mixing DPL and function shipping to the same CICS system

Great care should be taken when mixing function shipping and DPL to the same CICS system, from the same client task. These are some considerations:

- A client task cannot function ship requests and then use distributed program link with the SYNCONRETURN option in the same session (same logical unit of work or system initialization parameter MROFSE=YES specified or IPCONN MIRRORLIFE set to UOW or TASK). The distributed program link fails with an INVREQ response. In this case EIBRESP2 is set to 14.
- A client task cannot function ship requests and then use distributed program link with the TRANSID option in the same client logical unit of work. The distributed program link fails with an INVREQ response. In this case, EIBRESP2 is set to 15.
- Any function-shipped requests that follow a DPL request with the SYNCONRETURN option runs in a separate logical unit of work from the server logical unit of work.
- Any function-shipped requests running that follow a DPL request with the TRANSID option to the same server region runs under the transaction code specified on the TRANSID option, instead of under the default mirror transaction code. The function-shipped requests are committed as part of the overall client logical unit of work when the client commits.
- Any function-shipped requests running before or after a DPL request without the SYNCONRETURN or TRANSID options are committed as part of the overall client logical unit of work when the client commits.

See CICS function shipping for more information about function shipping.

Mixing DPL and DTP to the same CICS system

Care should be taken when using both DPL and DTP in the same application, particularly using DTP in the server program. For example, if you have not used the SYNCONRETURN option, you must avoid taking a syncpoint in the DTP partner which requires the DPL server program to syncpoint.

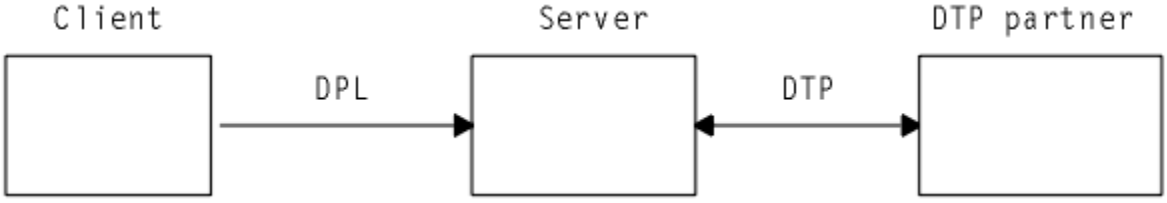


Figure 81. Example of mixing DPL and DTP

Restricting a program to the distributed program link subset

When a program executes as the result of a distributed program link, it is restricted to a subset of the full CICS API called the distributed program link subset. The commands that are prohibited in a server program are summarized in Table 25 on page 221.

You can specify, in the program resource definition only, that you want to restrict a program invoked by a local LINK command to this subset with the EXECUTIONSET(DPLSUBSET) option. The use of any prohibited commands can then be detected before an application program is used in a distributed environment.

When the server program is running locally the following considerations apply:

- If EXECUTIONSET(DPLSUBSET) is specified on the server program then the SYNCONRETURN option causes an implicit syncpoint to be taken in the local server program, before returning control to

the client program. In this case, because the server program is running locally, both the client and server resources are committed. However, SYNCONRETURN is intended for use when the client has no recoverable resources.

- If EXECUTIONSET(FULLAPI) is specified on the server program, the SYNCONRETURN option is ignored.
- The TRANSID and DATALENGTH options are ignored when processing the local link, but the format of the arguments is checked, for example, the TRANSID argument cannot be all blank.

Determining how a program was invoked

The 2-byte values returned on the STARTCODE option of the **ASSIGN** command are extended in support of the distributed program link function enabling the server program to find out that it is restricted to the distributed program link subset. See CICS API commands for programming information about EXEC CICS commands.

Accessing user-related information with the ASSIGN command

The values returned with the USERID and OPID keywords of the **ASSIGN** command in the server program depend on the way the ATTACHSEC option is defined for the connection being used between the client CICS region and the server CICS region. For example, the system could be defined so that the server program could access the same USERID and OPID values as the client program or could access different values determined by the ATTACHSEC option.

If ATTACHSEC(LOCAL) is specified, the userid to which the OPID and USERID parameters correspond is one of the following, in the order shown:

1. The user ID specified on the **USERID** parameter (for preset security) of the SESSIONS resource definition, if present
2. The user ID specified on the **SECURITYNAME** parameter of the connection resource definition, if present and no preset security user ID is defined on the sessions
3. The user ID specified on the **DFLTUSER** system initialization parameter of the server region, if neither the sessions nor connection definitions specify a user ID

If any value other than LOCAL is specified for ATTACHSEC , the signed-on user ID is the one received in the attach request from the client region.

For more information, see Intercommunication security

Another security-related consideration concerns the use of the CMDSEC and RESSEC options of the **ASSIGN** command. These are attributes of the transaction definition for the mirror transaction in the server region. They can be different from the definitions in the client region, even if the same TRANSID is used.

Exception conditions for LINK command

There are error conditions introduced in support of DPL which are returned to the client and server programs.

Exception conditions returned to the client program

Condition codes returned to a client program describe such events as "remote system not known" or "failure to commit" in the server program. There are different reasons, identified by EIBRESP2 values, for raising the INVREQ and LENGERR conditions on a LINK command. The ROLLEDBACK, SYSIDERR, and TERMERR conditions can also be raised. See CICS API commands for programming information about these commands.

If the mirror transaction in the remote region fails, the application program that issued the DPL request can handle the abend of the mirror, and commit its own local resources, *only if both the following are true* :

1. The application program explicitly handles the abend caused by the failure of the mirror, and either:
   - Takes an implicit sync point by normal transaction termination

- *or* Issues an explicit sync point request.

2. The remote mirror transaction performed no recoverable work within the scope of the unit of work of the application program. That is, the mirror was invoked only for a distributed program link (DPL) request with SYNCONRETURN.

In all other cases—that is, if the application program does not handle the abend, or the mirror does any recoverable work (for example, a file update, even to an unrecoverable file) CICS forces the transaction to be backed out.

The PGMIDERR condition is raised on the HANDLE ABEND PROGRAM, LOAD, RELEASE, and XCTL commands if the local program definition specifies that the program is remote. This exception is qualified by an EIBRESP2 value of 9.

Exception conditions returned to the server program

INVREQ is returned, qualified by an EIBRESP2 value of 200, to a server program if it issues one of the prohibited commands summarized in Table 25 on page 221 . If the server program does not handle the INVREQ condition, the default action is to abend the mirror transaction under which the server program is running with abend code ADPL.

If you attempt to use any of the EXEC CICS WEB commands *as a server*, those commands fail with INVREQ and a RESP2 value of 1. The following three commands fail with INVREQ and a RESP2 value of 5: EXEC CICS WEB EXTRACT, EXTRACT TCPIP and EXTRACT CERTIFICATE. There is no issue with using these commands where CICS is acting as a client.

For programming information about the DPL-related exception conditions, see LINK.

Table 25. API commands prohibited in programs invoked by DPL

Command	**Options**
ASSIGN	ALTSCRNHT ALTSCRNWD APLKYBD APLTEXT BTRANS COLOR DEFSCRNHT DEFSCRNWD DELIMITER DESTCOUNT DESTID DESTIDLENG DS3270 DSSCS EWASUPP EXTDS FACILITY FCI GCHARS GCODES GMMI HILIGHT INPARTN KATAKANA LDCMNEM LDCNUM MAPCOLUMN MAPHEIGHT MAPLINE MAPWIDTH MSRCONTROL NATLANGINUSE NEXTTRANSID NUMTAB OPCLASS OPSECURITY OUTLINE PAGENUM PARTNPAGE PARTNS PARTNSET PS QNAME SCRNHT SCRNWD SIGDATA SOSI STATIONID TCTUALENG TELLERID TERMCODE TERMPRIORITY TEXTKYBD TEXTPRINT UNATTEND USERNAME USERPRIORITY VALIDATION
CONNECT PROCESS	all
CONVERSE	all
EXTRACT ATTRIBUTES	all
EXTRACT PROCESS	all
FREE	all
HANDLE AID	all
ISSUE	ABEND CONFIRMATION ERROR PREPARE SIGNAL PRINT ABORT ADD END ERASE NOTE QUERY RECEIVE REPLACE SEND WAIT
LINK	INPUTMSG INPUTMSGLEN
PURGE MESSAGE	all
RECEIVE	all
RETURN	INPUTMSG INPUTMSGLEN

Table 25. API commands prohibited in programs invoked by DPL (continued)	
Command	**Options**
ROUTE	all
SEND	CONTROL MAP PARTNSET TEXT TEXT(MAPPED) TEXT(NOEDIT) PAGE
SIGNOFF	all
SIGNON	all
START	TERMID, where its value is the ID of the intersystem session. (That is, where the issuing task's principal facility is a session rather than a terminal.)
START CHANNEL	TERMID, where its value is the ID of the intersystem session. (That is, where the issuing task's principal facility is a session rather than a terminal.)
SYNCPOINT	Can be issued in server region if SYNCONRETURN specified on LINK
SYNCPOINT ROLLBACK	Can be issued in server region if SYNCONRETURN specified on LINK
WAIT TERMINAL	all
XCTL	INPUTMSG INPUTMSGLEN

The following commands are also restricted but can be used in the server region if SYNCONRETURN is specified on the LINK:

- CPIRR COMMIT
- CPIRR BACK
- EXEC DLI TERM
- CALL DLI TERM

Where only certain options are prohibited on the command, they are shown. All the APPC commands listed are prohibited only when they refer to the principal facility. One of these, the CONNECT PROCESS command, causes an error even if it refers to the principal facility in a non-DPL environment. It is included here because, if a CONNECT PROCESS command refers to its principal facility in a server program, the exception condition raised indicates a DPL error.

Asynchronous processing

The response from a remotely initiated transaction is not necessarily returned to the task that initiated the transaction, which is why the processing is referred to as *asynchronous* .

Asynchronous processing is useful when you do not need or want to tie up local resources while having a remote request processed. For example, with online inquiry on remote databases, terminal operators can continue entering inquiries without having to wait for an answer to the first one.

You can start a transaction on a remote system using a START command just like a local transaction. You can use the RETRIEVE command to retrieve data that has been stored for a task as a result of a remotely issued START, CANCEL, SEND, or RECEIVE command, as if it were a local transaction.

Distributed transaction processing (DTP)

The main advantage of DTP is that two transactions can have exclusive control of a session and can "converse" . DTP is useful when you need remote resources to be processed remotely or if you need to transfer data between systems.

By using DTP, you can design flexible and efficient applications. DTP can be used with either EXEC CICS or CPI Communications. You can use C, C++, and assembler language in DTP application programs that

hold LU type 6.2 unmapped conversations using the EXEC CICS API and applications that use the CICS intercommunication facilities.

DTP can be used with various partners, including both CICS and non-CICS platforms, as long as they support APPC. For further information about DTP, see Distributed transaction processing overview and Getting started with intercommunication.

Common Programming Interface Communications (CPI Communications)

CPI Communications provides an alternative API to existing CICS APPC support. CPI Communications provides DTP on APPC connections and can be used in COBOL, C, C++, PL/I, and assembler language.

CPI Communications defines an API that can be used in APPC networks that include multiple system platforms, where the consistency of a common API is of benefit.

Common Programming Interface Communications (CPI Communications) is the communication element of the Systems Applications Architecture (SAA) Common Programming Interface (CPI).

The CPI Communications interface can converse with applications on any system that provides an APPC API. This includes applications on CICS platforms. You can use EXEC CICS APPC API commands on one end of a conversation and CPI Communications commands on the other.

CPI Communications requires specific information (side information) to begin a conversation with partner program. CICS implementation of side information is achieved using the partner resource, which your system programmer is responsible for maintaining.

Calls from the application to the CPI Communications interface are resolved by link-editing the application with the CICS CPI Communications stub (DFHCPLC). For information about how to do this, see “Including the CICS-supplied interface modules” on page 691.

The CPI Communications API is defined as a general call interface. See z/VM: CPI Communications User's Guide.

External CICS interface (EXCI)

The external CICS interface is an application programming interface that enables a non-CICS program (a client program) running in MVS to call a program (a server program) running in a CICS region and to pass and receive data with a communications area. The CICS program is invoked as if linked-to by another CICS program.

This programming interface allows a user to allocate and open sessions (pipes) to a CICS system and to pass distributed program link (DPL) requests over them. CICS interregion communication (IRC) supports these requests and each pipe maps onto one MRO session.

For programming information about EXCI, see Introduction to the external CICS interface.

A client program that uses the external CICS interface can operate multiple sessions for different users (either under the same or separate TCBs) all coexisting in the same MVS address space without knowledge of, or interference from, each other.

The external CICS interface provides two forms of programming interface:

- The EXCI CALL interface consists of six commands that allow you to:
  - Allocate and open sessions to a CICS system from non-CICS programs running under MVS
  - Issue DPL requests on these sessions from the non-CICS programs
  - Close and de-allocate the sessions on completion of the DPL requests
- The EXEC CICS interface provides:
  - A single composite command (LINK PROGRAM) that performs all six commands of the EXCI CALL interface in one invocation

  The command takes the same form as the distributed program link command of the CICS command-level application programming interface.

CICS supports MVS resource recovery services (RRS) in applications that use the external CICS interface. This means that:

- The unit of work within which the CICS server program changes recoverable resources can now become part of the MVS unit of recovery associated with the EXCI client program.
- The CICS server unit of work can be committed when the server program returns control to the client or continues over multiple EXCI DPL calls, until the EXCI client decides to commit or backout the unit of recovery.

CICS services for application programs

CICS provides a number of services that enable you to construct business-critical applications.

Understanding file control

CICS data management services have traditionally been known as CICS file control. CICS file control offers you access to data sets that are managed by either the Virtual Storage Access Method (VSAM) or the basic direct-access method (BDAM).

CICS file control lets you read, update, add, and browse data in VSAM and BDAM data sets and delete data from VSAM data sets. You can also access CICS shared data tables and coupling facility data tables using file control.

A CICS application program reads and writes its data in the form of individual records. Each read or write request is made by a CICS command.

To access a record, the application program must identify both the record and the data set that holds it. It must also specify the storage area into which the record is to be read or from which it is to be written.

VSAM data sets: KSDS, ESDS, RRDS

CICS supports access to the following types of data set, Key-sequenced data set (KSDS), Entry-sequenced data set (ESDS) and Relative record data set (RRDS) (both fixed and variable record lengths).

VSAM data sets are held on direct-access storage devices (DASD) auxiliary storage. VSAM divides its data set storage into control areas (CA), which are further divided into control intervals (CI). Control intervals are the unit of data transmission between virtual and auxiliary storage. Each one is of fixed size and, in general, contains a number of records. A KSDS or ESDS can have records that extend over more than one control interval. These are called spanned records.

Key-sequenced data set (KSDS)

A **key-sequenced data set** has each of its records identified by a key. (The **key** of each record is a field in a predefined position within the record.) Each key must be unique in the data set.

When the data set is initially loaded with data, or when new records are added, the logical order of the records depends on the collating sequence of the key field. This also fixes the order in which you retrieve records when you browse through the data set.

To find the physical location of a record in a KSDS, VSAM creates and maintains an **index**. This relates the key of each record to the record's relative location in the data set. When you add or delete records, this index is updated accordingly.

CICS supports, in both RL and non-RLS mode, KSDS data sets that are defined with extended format and extended addressability attributes.

Entry-sequenced data set (ESDS)

An **entry-sequenced data set** is one in which each record is identified by its **relative byte address** (RBA).

Records are held in an ESDS in the order in which they were first loaded into the data set. New records added to an ESDS always go after the last record in the data set. You cannot delete records or alter their lengths. After a record has been stored in an ESDS, its RBA remains constant. When browsing, records are retrieved in the order in which they were added to the data set.

A standard RBA is an unsigned 32 bit number. The use of a 32 bit RBA means that a standard ESDS cannot contain more than 4 GB of data. However, there is a different ESDS that supports 64 bit **extended relative byte addresses** (**XRBA** s) and which is therefore not subject to the 4 GB limit. This type of ESDS is called an "extended format, extended addressing ESDS data set" . For brevity, we will refer to it as an "extended addressing ESDS" , or as an " **extended ESDS** " . CICS supports 64-bit XRBAs and extended ESDS data sets.

Relative record data set (RRDS)

A **relative record data set** has records that are identified by their relative record number (RRN). The first record in the data set is RRN 1, the second is RRN 2, and so on.

Records in an RRDS can be fixed or variable length records, and the way in which VSAM handles the data depends on whether the data set is a fixed or variable RRDS. A fixed RRDS has fixed-length slots predefined to VSAM, into which records are stored. The length of a record on a fixed RRDS is always equal to the size of the slot. VSAM locates records in a fixed RRDS by multiplying the slot size by the RRN (which you supply on the file control request), to calculate the byte offset from the start of the data set.

A variable RRDS, can accept records of any length up to the maximum for the data set. In a variable RRDS VSAM locates the records with an index.

A fixed RRDS generally offers better performance. A variable RRDS offers greater function.

CICS supports access to extended RRDS or VRRDS data sets if you use an RRN that can be specified in a four byte RRN field to access the records that reside beyond the 4 GB boundary.

Empty data sets

An empty data set is a data set that has not yet had any records written to it. VSAM imposes several restrictions on an empty data set that is opened in non-RLS access mode. However, CICS hides all these restrictions from you, allowing you to use an empty data set in the same way as a data set that contains data, regardless of the access mode.

VSAM alternate indexes

Sometimes you want to access the same set of records in different ways. For example, you can have records in a personnel data set that have as their key an employee number. No matter how many Smiths you have, each of them has a unique employee number. Think of this as the primary key.

If you were producing a telephone directory from the data set, you would want to list people by name rather than by employee number. You can identify records in a data set with a secondary (alternate) key instead of the primary key described previously. So the primary key is the employee number, and the employee name is the **alternate key**. Alternate keys are just like the primary key in a KSDS—fields of fixed length and fixed position within the record. You can have any number of alternate keys per base file and, unlike the primary or base key, alternate keys need not be unique.

To continue the personnel example, the employee's department code might be defined as a further alternate key.

VSAM allows KSDS and ESDS (but not RRDS or extended ESDS) data sets to have alternate keys. When the data set is created, one secondary or **alternate index** is built for each alternate key in the record and is related to the primary or base key. To access records using an alternate key, you must define a further VSAM object, an **alternate index path** . The path then behaves as if it were a KSDS in which records are accessed using the alternate key.

When you update a record by way of a path, the corresponding alternate index is updated to reflect the change. However, if you update the record directly by way of the base, or by a different path, the alternate

index is only updated if it has been defined to VSAM (when created) to belong to the **upgrade set** of the base data set. For most applications, you probably want your alternate index to be in the upgrade set.

A CICS application program disregards whether the file it is accessing is a path or the base. In a running CICS system, access to a single base data set can be made by way of the base and by any of the paths defined to it, if each access route is defined to CICS.

It is also possible for a CICS application program to access a file that has been directly defined as an alternate index rather than a path. This results in index data being returned to the application program rather than file data. This operation is not supported for files opened in record-level sharing (RLS) mode.

Accessing files in RLS mode

Record-level sharing (RLS) is a VSAM function that enables VSAM data to be shared, with full update capability, between many applications running in many CICS regions.

With RLS, CICS regions that share VSAM data sets can reside in one or more MVS images within an MVS parallel sysplex. RLS also provides some benefits when data sets are being shared between CICS regions and batch jobs.

If you open a file in RLS mode, locking takes place at the record level instead of the Control-Interval level, thus reducing the risk of deadlocks.

CICS supports record-level sharing (RLS) access to the following types of VSAM data set:

- Key sequenced data sets (KSDS). Note that if you are using KSDS, you cannot use the relative byte address (RBA) to access files.
- Entry sequenced data sets (ESDS). Note that although RLS access mode is permitted for entry sequenced data sets (ESDS), it is not recommended, as it can have a negative effect on the performance and availability of the data set when you are adding records. (See Using VSAM record-level sharing).
- Relative record data sets (RRDS), for both fixed and variable length records.

Note: If you issue the SET FILE EMPTY command for a file that specifies RLS mode, the request is accepted but is ignored all the time the file is opened in RLS mode. If you close and switch the file to non-RLS mode, the data set is then reset to empty (provided it is defined as reusable on its IDCAMS definition).

Most types of data set are eligible to participate in VSAM record level sharing and most CICS applications can benefit from this mode of access. However, there are some limitations that could affect some applications. The following types of file, data set, or method of access are **not** supported in RLS mode:

- RBA access to a KSDS
- Key-range data sets
- Temporary data sets
- VSAM clusters with the IMBED attribute
- Direct opening of an alternate index
- Opening individual components of a cluster
- Access to catalogs or to VVDS data sets
- CICS-maintained data tables
- Hiperbatch

Encrypting user data sets

You can encrypt CICS user data sets for which z/OS data set encryption is supported.

Before you begin

Check the data set types for which z/OS data set encryption is supported. See Planning for data set encryption.

Ensure that you have set up the key labels to use. See Managing Cryptographic Keys Using the Key Generator Utility Program in z/OS Cryptographic Services ICSF Administrator's Guide.

Ensure that the RACF tasks to provide authority to create encrypted data sets and to provide access to the key labels are complete, including the following tasks:

- Grant read access to `STGADMIN.SMS.ALLOW.DATASET.ENCRYPT CL(FACILITY)` to allow users to create encrypted data sets.
- Grant read access to the key labels. When a user attempts to access an encrypted data set, RACF checks that the user has authority to use the key label before any encryption occurs by checking access to the relevant profiles in the CSFKEYS and CSFSERV classes. For more information about these classes, see Data Set Encryption in z/OS DFSMS Using Data Sets .

Ensure that you have set up the key labels to use. See Managing Cryptographic Keys Using the Key Generator Utility Program in z/OS Cryptographic Services ICSF Administrator's Guide.

About this task

For user data sets defined to CICS, encryption support includes key-sequenced data sets (KSDS), entry-sequenced data sets (ESDS), relative record data sets (RRDS), and variable relative record data sets (VRRDS), accessed through base VSAM and VSAM Record Level Sharing (RLS). Encryption is also supported for the backing VSAM key-sequenced data sets that are used for shared data tables or coupling facility data tables.

Procedure

1. Either create an encryption key and associated key label, or use an existing key label.
2. Allocate a new instance of the data set, specifying extended format and the data set key label.
   a) Specify extended format in one of the following ways:
      - Through the SMS data class, using the **DSNTYPE=EXT** parameter and **R** (required) or **P** (preferred) subparameters. Specify **R** to ensure that the data set is allocated in extended format.
      - Use the **DSNTYPE** parameter on the JCL DD statement, with values of **EXTREQ** (required) or **EXTPREF** (preferred). Specify **EXTREQ** to ensure that the data set is allocated in extended format. The DSNTYPE specified on a JCL DD statement overrides any DSNTYPE set for the data class.

      When you specify extended format, ensure that you only specify the extended addressing attribute if you also require that option for your data set.

   b) Specify the data set key label in one of the following ways:
      - On a RACF data set profile by using the DATAKEY keyword.
      - Through JCL by using the DSKEYLBL keyword.
      - On dynamic allocation through the DALDKYL text unit.
      - Through TSO allocate by using DSKEYLBL.
      - On an IDCAMS DEFINE through the KEYLABEL parameter of DEFINE CLUSTER.
      - On the SMS data class by specifying the **Data Set Key Label** field on the DEFINE/ALTER panel.

      When a data set key label is specified through more than one interface, the order of precedence is as follows:

      i) The RACF data set profile

      ii) JCL, dynamic allocation, TSO allocate, or IDCAMS DEFINE

      iii) SMS data class

      Consider the granularity of different key labels that you want to use across the groups of data sets that you plan to encrypt, so that you balance convenience of managing the key labels against maximizing the protection of your data. This also helps to determine the appropriate mechanism to use to specify the key label.

3. Copy across data from the existing data set into the new, encrypted, data set.

   Use an appropriate DFSMS function such as REPRO, or the EXPORT and IMPORT functions. A typical sequence is as follows:

   a. Create the new data set.
   b. Copy the data across from the old data set.
   c. Delete the old data set.
   d. Rename the new data set back to the old name.

Identifying VSAM records

You can identify records in data sets by *key* , by *relative byte address (RBA)* or *extended relative byte address (XRBA)* , or by *relative record number (RRN)*.

The RIDFLD (record identification field) option on the CICS file control commands identifies a field containing the record identification appropriate to the access method and the type of file being accessed. For most things you can do to a record (read, add, delete, or start a browse), you identify the record by specifying the RIDFLD option, except when you have read the record for update first. However, there is no RIDFLD for ENDBR, REWRITE, and UNLOCK commands.

- "Key" on page 228
- "Relative record number (RRN), relative byte address (RBA) and extended relative byte address (XRBA)" on page 228

Key

If you use a key, you can specify either a complete key or a generic (partial) key. However, if you write a record to a KSDS or write a record by an alternate index path, you must specify the complete key in the RIDFLD option of the command.

When you use a generic key, you must specify its length in the KEYLENGTH option and you must specify the GENERIC option on the command. A generic key cannot have a key length equal to the full key length. You must define it to be shorter than the complete key.

You can also specify the GTEQ option on certain commands, for both complete and generic keys. The command then positions at, or applies to, the record with the next higher key if a matching key cannot be found. When accessing a data set by way of an alternate index path, the record identified is the one with the next higher alternate key when a matching record cannot be found.

Even when using generic keys, always use a storage area for the record identification field that is equal in length to the length of the complete key. During a browse operation, after retrieving a record, CICS copies into the record identification area the actual identifier of the record retrieved. CICS returns a complete key to your application, even when you specified a generic key on the command. For example, a generic browse through a KSDS returns the complete key to your application on each READNEXT and READPREV command.

Relative record number (RRN), relative byte address (RBA) and extended relative byte address (XRBA)

You can use the RRN, RBA, and XRBA options on most commands that access data sets. In effect, they define the format of the record identification field (RIDFLD).

Unless you specify the RRN, RBA, or XRBA, the RIDFLD option should hold a key to be used for accessing a KSDS (or a KSDS or ESDS by way of an alternate index).

RRN

RRN specifies that the record identification field contains the relative record number of the record to be accessed. The first record in the data set is number one. All file control commands that refer to an RRDS, and specify the RIDFLD option, must also specify the RRN option.

RBA and XRBA

RBA specifies that the record identification field contains the relative byte address of the record to be accessed. A relative byte address is used to access an ESDS, and it can also be used to access a KSDS that is not opened in RLS access mode. All file control commands that refer to an ESDS base, and specify the RIDFLD option, must also specify the RBA option.

Note: If a KSDS is accessed in this way, the RBA of the record can change during the transaction as a result of another transaction adding records to, or deleting records from, the same data set.

An RBA is an unsigned 32 bit number. The use of a 32 bit RBA means that a standard ESDS cannot contain more than 4 GB of data. However, there is a different ESDS that supports 64 bit extended relative byte addresses (XRBAs) and which is therefore not subject to the 4 GB limit. We call this type of ESDS an extended ESDS. From CICS TS for z/OS, Version 3.2 onwards, CICS supports 64 bit XRBAs and extended ESDS data sets.

Normally, to access an extended ESDS a program supplies a 64 bit XRBA. You can convert existing programs that use a 32 bit RBA to use a 64 bit XRBA, by replacing the RBA keyword on the relevant commands with the XRBA keyword and changing the lengths of the areas used for keys. Also in some circumstances it is possible to reuse older programs without change to access an extended ESDS, if they pass 32 bit RBAs but do not use them. "Upgrading to extended addressing for ESDS" on page 229 explains how to upgrade a standard ESDS to an extended addressing ESDS, and how to upgrade or reuse your existing programs with the new format.

Upgrading to extended addressing for ESDS

To use an extended entry-sequenced data set (ESDS), you must upgrade the data set and convert existing CICS application programs that use 32-bit relative byte addressing (RBA) to 64-bit extended relative byte addressing (XRBA).

Upgrading a standard ESDS to an extended addressing ESDS

Note: Before upgrading a standard ESDS to use extended addressing, if your data set is defined to use forward recovery it is essential to upgrade your forward recovery product to one that can read the new log records written for extended addressing ESDSs. If you use CICS VR, you must use CICS VSAM Recovery for z/OS V4.2 or higher.

To convert an existing standard ESDS to an extended addressing ESDS, you must re-create the data set as follows:

1. If you want to continue to use the contents of the existing data set, take a copy of its contents. You can use the AMS REPRO function to do this.
2. Delete the existing data set.
3. Create a new data set. You can base the AMS definition of the new data set on that of the old data set. The only mandatory change is that the **DATACLAS** parameter of the new data set definition must name an SMS data class that specifies both extended format and extended addressing.

   How to define SMS data classes is described in z/OS DFSMSdfp Storage Administration.
4. If required, restore the contents of the data set from the copy taken previously.

Upgrading a program from 32-bit RBA to 64-bit XRBA

To convert an existing program from using 32-bit RBA to 64-bit extended relative byte addressing (XRBA) you must:

1. Replace the RBA keyword with the XRBA keyword, on all the following commands:
   - EXEC CICS READ
   - EXEC CICS READNEXT
   - EXEC CICS READPREV
   - EXEC CICS RESETBR

- EXEC CICS STARTBR
- EXEC CICS WRITE

2. Replace all 4-byte areas used for keys with 8-byte areas. If you change "RBA" to "XRBA" but don't change the length of the key areas:
   a. On STARTBR and READ commands, CICS will treat your 4-byte RBAs as being the top half of 8-byte XRBAs. In most cases this produces a huge XRBA number. You can track down this error quite quickly because the program immediately receives a "no record at RBA" response.
   b. WRITE commands might produce more subtle errors. The command returns an 8-byte XRBA that overwrites the 4 bytes immediately following the key area.

Using RBA-insensitive programs to access extended ESDS data sets

It is possible to reuse existing 32-bit RBA programs, that do not make use of the RBAs, to access 64-bit extended ESDSs.

For example, there is a common type of application in which records are first written sequentially and later browsed sequentially from the beginning. Although RBAs are passed between CICS and the program, the program makes no use of them. The program reads or writes the next record only. These programs are called "RBA-insensitive" . Other programs, such as those that directly read or update records at named RBAs, are called "RBA-sensitive".

Existing 32-bit RBA-insensitive programs can access 64-bit extended ESDSs without change. Both RLS and non-RLS modes are supported.

Thirty-two-bit RBA-sensitive programs cannot access 64-bit extended ESDSs, even if the data set contains less than 4 gigabytes of data.

Connecting a back-level AOR to a CICS TS for z/OS, Version 5.6 FOR

In this scenario, old-style 32-bit RBA programs try to access files on a CICS TS for z/OS, Version 5.6 file-owning region (FOR). This access works in either of the following cases:

- The target file in the FOR has not been converted from conventional ESDS to extended addressing ESDS.
- The target file has been converted to extended addressing ESDS but the program is RBA-insensitive.

If the target file has been converted to extended addressing ESDS, a 32-bit RBA-sensitive program running in the AOR cannot access it. The program receives an ILLOGIC response.

Connecting a CICS TS for z/OS, Version 5.6 AOR to a back-level FOR

In this scenario, new-style 64-bit XRBA programs try to access files on a back-level file-owning region.

Because the target region supports only 32-bit RBAs, it does not understand a 64-bit XRBA. The program receives an ILLOGIC response.

Locking of VSAM records in recoverable files

Locks are acquired by VSAM if the file is accessed in record-level sharing (RLS) mode, and by CICS if not. The locks are held on behalf of the transaction doing the change until it issues a sync point request or terminates (at which time a sync point is automatically performed).

The prevention of transaction deadlocks in terms of the record locks acquired whenever records in a recoverable file are modified is explained in "Transaction deadlocks" on page 240 . For VSAM recoverable file processing, note the following:

- Whenever a VSAM record is obtained for modification or deletion, CICS file control (or VSAM in the case of RLS) locks the record with an ENQUEUE request using the primary record identifier as the enqueue argument.

  If a record is modified by way of a path, the enqueue uses the base key or the base RBA as an argument. So CICS permits only one transaction at a time to perform its request, the other transactions having to wait until the first has reached a sync point.

- For the READ UPDATE and REWRITE-related commands the record lock is acquired as soon as the READ UPDATE command has been issued.

  For a DELETE command that has not been preceded by a READ UPDATE command, or for a WRITE command, the record lock is acquired at the time the VSAM command is executed.

  For a WRITE MASSINSERT command (which consists of a series of WRITE commands), a separate record lock is acquired at the time each individual WRITE command is performed. Similarly, for a DELETE GENERIC command, each record deleted acquires a separate lock on behalf of the transaction issuing the request.

Update locks and delete locks (non-RLS mode only)

The record locks referred to previously are known as update locks, because they are acquired whenever a record is updated (modified). A further type of lock (a delete lock) can also be acquired by file control whenever a DELETE, WRITE, or WRITE MASSINSERT command is being performed for a recoverable KSDS or a recoverable path over a KSDS. A delete operation therefore can acquire two separate locks on the record being deleted.

The separate delete lock is needed because of the way file control does its write operations. Before executing a WRITE MASSINSERT command to a KSDS or RRDS, file control finds and locks the empty range into which the new record or records are to go. The empty range is locked by identifying the next existing record in the data set and acquiring its delete lock.

The empty range is locked to stop other requests simultaneously adding records into it. Moreover, the record defining the end of the empty range cannot be removed during the add operation. If another transaction issues a request to add records into the empty range or to delete the record at the end of the range, the delete lock makes the transaction wait until the WRITE or WRITE MASSINSERT command is complete. The record held with a delete lock can, however, be ***updated*** by another transaction during the write operation if it is in another CI.

Unlike an update lock, a delete lock is held only for the duration of a delete or write operation, or a sequence of WRITE MASSINSERT commands terminated by an UNLOCK command. A WRITE MASSINSERT command that adds records to the file into more than one empty range releases the previous delete lock as it moves into a new empty range.

The CICS enqueuing mechanism is only for updates and deletes and does not prevent a read request being satisfied before the next sync point. The integrity of a READ command in these circumstances is not guaranteed.

RLS Record level locking

Files opened in RLS mode can be accessed by many CICS regions simultaneously. This means that it is impractical for the individual CICS regions to attempt to control record locking, and therefore VSAM maintains a single central lock structure using the lock-assist mechanism of the MVS coupling facility.

This central lock structure provides sysplex-wide locking at a record level—control interval (CI) locking is not used. This is in contrast to the locks for files in non-RLS mode, the scope of which is limited to a single CICS region, and that are either CI locks or CICS ENQs.

Record locks within RLS are owned by a named UOW within a named CICS region. The lock owner name is the APPLID of the CICS region, plus the UOW ID. For example, when CICS makes a request that can create a lock, CICS passes to VSAM the UOW ID. This enables VSAM to build the lock name using the UOW ID, the record key, and the name of the CICS region.

CICS releases all locks on completion of a UOW using a VSAM interface.

When more than one request requires an exclusive lock against the same resource, VSAM queues the second and subsequent requests until the resource is freed and the lock can be granted. However, VSAM does not queue requests for resources locked by a retained lock (see “Active and retained states for locks” on page 233).

Note: For MASSINSERT operations on a file opened in RLS access mode, CICS acquires a separate update lock at the time each individual WRITE command is issued. Unlike the non-RLS mode operation (described under "Locking of VSAM records in recoverable files" on page 230) CICS does *not* acquire the separate delete lock in addition to the update lock. There is no equivalent to range locking for the MASSINSERT function for files opened in non-RLS mode.

Exclusive locks and shared locks

VSAM supports two types of lock for files accessed in RLS mode. The two types are *exclusive* and *shared* locks.

Exclusive locks can be active or retained; shared locks can only be active (see "Active and retained states for locks" on page 233). Note that there are no delete locks in RLS mode.

Exclusive locks

Exclusive locks protect updates to file resources, both recoverable and non-recoverable. They can be owned by only one transaction at a time. Any transaction that requires an exclusive lock must wait if another task currently owns an exclusive lock or a shared lock against the requested resource.

Shared locks

Shared locks support read integrity. They ensure that a record is not in the process of being updated during a read-only request. Shared locks can also be used to prevent updates of a record between the time that a record is read and the next syncpoint.

A shared lock on a resource can be owned by several tasks at the same time. However, although several tasks can own shared locks, there are some circumstances in which tasks can be forced to wait for a lock:

- A request for a shared lock must wait if another task currently owns an exclusive lock on the resource.
- A request for an exclusive lock must wait if other tasks currently own shared locks on this resource.
- A new request for a shared lock must wait if another task is waiting for an exclusive lock on a resource that already has a shared lock.

Lock duration

Shared locks for repeatable read requests, for recoverable and unrecoverable data sets, are held until the next sync point.

Exclusive locks against records in a unrecoverable data set remain held only for the duration of the request—that is, they are acquired at the start of a request and released on completion of it. For example, a lock acquired by a WRITE request is released when the WRITE request is completed, and a lock acquired by a READ UPDATE request is released as soon as the following REWRITE or DELETE request is completed. Exceptionally, locks acquired by sequential requests can persist beyond the completion of the immediate operation. Sequential requests are WRITE commands that specify the MASSINSERT option and browse for update requests. A lock acquired by a WRITE command with the MASSINSERT option is always released by the time the corresponding UNLOCKcommand completes, but can have been released by an earlier request in the WRITE MASSINSERT sequence. The exact request in the sequence that causes the lock to be released is not predictable. Similarly, a lock acquired by a READNEXT UPDATE command can still exist after the following DELETE or REWRITE command completes. Although this lock is guaranteed to be released by the time the subsequent ENDBR command completes, it can be released by some intermediate request in the browse sequence.

If a request is made to update a recoverable data set, the associated exclusive lock must remain held until the next sync point. This ensures that the resource remains protected until a decision is made to commit or back out the request. If CICS fails, VSAM continues to hold the lock until CICS is restarted.

```
 Task 1 Task 2
CICS: READ(filea) UPDATE KEY(99)
VSAM: grants exclusive lock - key 99
CICS: READ(filea) KEY(99)
with integrity
VSAM: Queues request for shared lock
CICS: REWRITE(filea) KEY(99)
VSAM: holds exclusive lock until syncpoint

CICS: task completes and takes syncpoint
VSAM: frees exclusive lock
VSAM grants shared lock to task 2
```

Figure 82. Illustration of lock contention between CICS tasks on a recoverable data set

Active and retained states for locks

VSAM RLS supports **active** and **retained** states for locks. The difference between these two types of lock is that whereas a request for a resource that has an active lock is queued until the resource becomes available, a request for a resource that has a retained lock fails immediately.

The active state is applicable to both exclusive and shared locks. However, only exclusive locks against recoverable resources can have their state changed from active to retained. The important characteristic of these states is that they determine whether a task must wait for a lock:

- A request for a lock is made to *wait* if there is already an active lock against the requested resource, except in two cases:
  1. A request for a shared lock does not have to wait if the current active lock is also a shared lock, and there are no exclusive lock requests waiting.
  2. An update request that specifies NOSUSPEND does not wait for a lock if an active lock exists. In this case, CICS returns an exception condition indicating that the "record is busy".
- A request for a lock is rejected immediately with the LOCKED condition if there is a retained lock against the requested resource.

When a lock is first acquired, it is an active lock. It is then either released, the duration of the lock depending on the type of lock, or if an event occurs which causes a UOW to fail temporarily and which would therefore cause the lock to be held for an abnormally long time, it is converted into a retained lock. The types of event that can cause a lock to become retained are:

- Failure of the CICS system, the VSAM server, or the whole MVS system
- A unit of work entering the backout failed state
- A distributed unit of work becoming indoubt owing to the failure of either the coordinating system or of links to the coordinating system

If a UOW fails, VSAM continues to hold the exclusive record locks that were owned by the failed UOW for recoverable data sets. To avoid new requests being made to wait for locks owned by the failed UOW, VSAM converts the active locks owned by the failed UOW into retained locks. Retaining locks ensures that data integrity for the locked records is maintained until the UOW is completed.

Exclusive recoverable locks are also converted into retained locks in the event of a CICS failure, to ensure that data integrity is maintained until CICS is restarted and performs recovery.

Exclusive recoverable locks are also converted into retained locks if the VSAM data-sharing server (SMSVSAM) fails (the conversion is carried out by the other servers in the Sysplex, or by the first server to restart if all servers have failed). This means that a UOW does not itself have to fail in order to hold retained RLS locks.

Any shared locks owned by a failed CICS region are discarded, and therefore an active shared lock can never become retained. Similarly, active exclusive unrecoverable locks are discarded. Only locks that are both exclusive and apply to recoverable resources are eligible to become retained.

BDAM data sets

CICS supports access to keyed and nonkeyed BDAM data sets. BDAM support uses the physical nature of a record on a DASD device.

BDAM data sets consist of unblocked records with the following format:

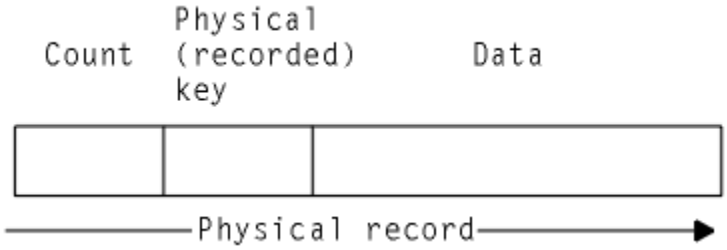


Figure 83. Format of unblocked records in a BDAM data set

Keyed BDAM files have a physical key identifying the BDAM record. The count area contains the physical key length, the physical data length, and the record's data location.

CICS can define a further structure on top of BDAM data sets, introducing the concept of blocked-data sets:

Figure 84. Blocked-data set

The data portion of the physical record is viewed as a block containing logical records. CICS supports the retrieval of logical records from the data part of the physical record. CICS also supports unblocked records (where the structure reverts to the original BDAM concept of one logical record per physical record).

To retrieve data from a physical record in a BDAM file under CICS, a record identification field (RIDFLD) has to be defined to specify how the physical record should be retrieved. This can be done using the physical key, by relative address, or by absolute address.

If the data set is defined to CICS as being blocked, individual records within the block can be retrieved (deblocked) in two addressing modes: by key or by relative record.

Deblocking by key uses the key of the logical record (that is, the key contained in the logical record) to identify which record is required from the block. Deblocking by relative record uses the record number in the block, relative to zero, of the record to be retrieved.

You specify the key or relative record number used for deblocking in a subfield of the RIDFLD option used when accessing CICS BDAM files. The addressing mode for CICS BDAM files is set in the FCT using the RELTYPE keyword.

For more details about record identification and BDAM record access, see "Understanding file control" on page 224.

Identifying BDAM records

You identify records in BDAM data sets by a block reference, a physical key (keyed data set), or a deblocking argument (blocked-data set).

The RIDFLD (record identification field) option on the CICS file control commands identifies a field containing the record identification appropriate to the access method and the type of file being accessed. For most things you can do to a record (read, add, delete, or start a browse), you identify the record by

specifying the RIDFLD option, except when you have read the record for update first. (However, there is no RIDFLD for ENDBR, REWRITE, and UNLOCK commands.)

For BDAM records, the record identification in the RIDFLD option has a subfield for the block reference, the physical key, and the deblocking argument. These subfields, when used, must be in the order given previously.

Note: When using EDF, only the first of the above three fields (the block reference subfield) is displayed.

Block reference subfield

This is one of the following:

- Relative block address: 3 byte binary, beginning at relative block zero (RELTYPE=BLK).
- Relative track and record (hexadecimal format): 2 byte TT, 1 byte R (RELTYPE=HEX).

  The 2 byte TT begins at relative track zero. The 1 byte R begins at relative record one.
- Relative track and record (zoned decimal format): 6 byte TTTTTT, 2 byte RR (RELTYPE=DEC).
- Actual (absolute) address: 8-byte MBBCCHHR (RELTYPE operand omitted).

The system programmer must specify the type of block reference you are using in the RELTYPE operand of the DFHFCT TYPE=FILE system macro that defines the data set.

Physical key subfield

You only need this if the data set has been defined to contain recorded keys. If used, it must immediately follow the block reference. Its length must match the length specified in the BLKKEYL operand of the DFHFCT TYPE=FILE system macro that defines the data set.

Deblocking argument subfield

You only need this if you want to retrieve specific records from a block. If used, it must immediately follow the physical key (if present) or the block reference. If you omit it, you retrieve an entire block.

The deblocking argument can be a key or a relative record number. If it is a key, specify the DEBKEY option on a READ or STARTBR command and make sure that its length matches that specified in the KEYLEN operand of the DFHFCT TYPE=FILE system macro. If it is a relative record number, specify the DEBREC option on a READ or STARTBR command. It is a 1 byte binary number (first record=zero).

Figure 85 on page 236 shows examples of the following possible forms of BDAM record identifiers. The examples assume a physical key of four bytes and a deblocking key of three bytes:

- Relative block number followed by relative record number for search by relative block and deblock by relative record
- Relative block number followed by a key for search by relative block and deblock by key
- TTR followed by physical key and deblocking key for search by relative track and record and key and deblock by key
- MBBCCHHR followed by relative record number for search by actual address and deblock by relative record
- TTTTTRR followed by physical key and deblocking key for search by zoned decimal relative track and record and key and deblock by key
- TR followed by physical key for search by relative track and record and deblock by key

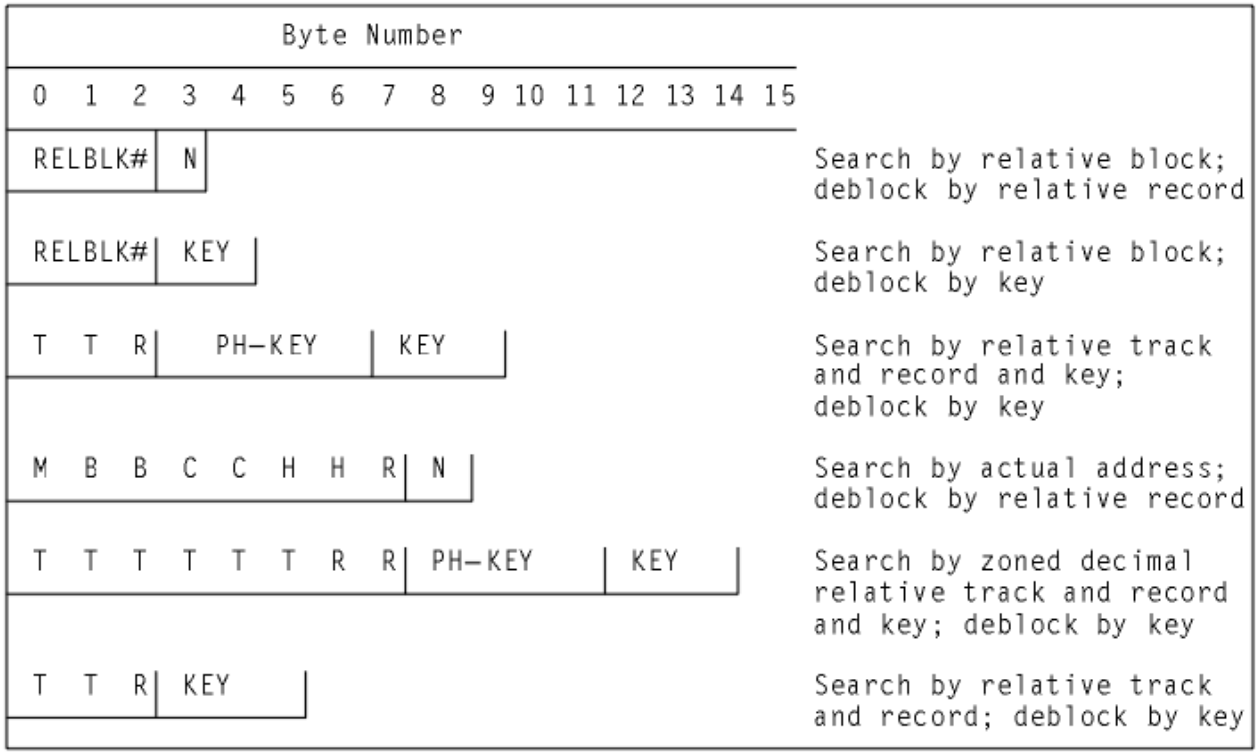


Figure 85. Examples of BDAM record identification

CICS shared data tables

CICS file control commands can access shared data tables. Shared data tables offer a method of constructing, maintaining, and gaining rapid access to data records contained in tables held in virtual storage, above the 16 MB line. Each shared data table is associated with a VSAM KSDS, known as its source data set.

For more information about shared data tables, see Shared data tables overview.

A table is defined using the FILE resource. When a table is opened, CICS builds it by extracting data from the table's corresponding source data set and loading it into virtual storage above the 16 MB line.

CICS supports two types of shared data table, as follows:

CICS-maintained tables (CMTs)
This type of data table is kept in synchronization with its source data set by CICS . All changes to the data table are reflected in the source data set. Similarly all changes to the source data set are reflected in the data table.

The source for a CICS-maintained data table cannot be a file opened in RLS access mode.

User-maintained tables (UMTs)
This type of data table is detached from its source data set after it has been loaded. Changes to the table are not automatically reflected in the source data set.

The full file control API appropriate to VSAM KSDS data sets is supported for CICS-maintained data tables. Requests that cannot be satisfied by reference to the data table result in calls to VSAM to access the source data set. Tables defined to be recoverable are supported with full integrity.

A subset of the file control API is supported for user-maintained tables. A table defined as recoverable participates in dynamic transaction backout but is not recovered at restart.

Coupling facility data tables

CICS file control commands can access coupling facility data tables (CFDTs). Coupling facility data tables provide a method of file data sharing, without the need for a file-owning region, and without the need for VSAM RLS support.

CICS coupling facility data table support is designed to provide rapid sharing of working data across a sysplex, with update integrity. The data is held in a coupling facility, in a table that is similar in many ways to a shared user-maintained data table. A coupling facility data table is different from a UMT in one important respect in that initial loading from a VSAM source data set is optional. You can specify LOAD(NO) and load the table by writing data directly from a user application program. The API used to store and retrieve the data is based on the file control API used for user-maintained data tables. Read access and write access to CFDTs have similar performance, making this form of table useful for informal shared data. Informal shared data is characterized as:

- Data that is relatively short term in nature (it is either created as the application is running, or is initially loaded from an external source)
- Data volumes that are not typically large
- Data that needs to be accessed fast
- Data of which the occasional loss can be tolerated by user applications
- Data that commonly requires update integrity.

Typical uses might include sharing scratchpad data between CICS regions across a sysplex, or sharing of files for which changes do not have to be permanently saved. There are many different ways in which applications use informal shared data, and most of these could be implemented using coupling facility data tables. Coupling facility data tables are useful for grouping data into different tables, where the items can be identified and retrieved by their keys. For example, you could use a record in a coupling facility data table to maintain the next free order number for use by an order processing application. Other examples are:

- Look-up tables of telephone numbers or the numbers of stolen credit cards
- Working data consisting of a few items, such as a subset of customers from a customer list
- Information that is specific to the user of the application, or that relates to the terminal from which the application is being run
- Data extracted from a larger file or database for further processing.

Coupling facility data tables allow various types of access to your informal data: read-only, single updater, multiple updaters, sequential access, random access, random insertion, and deletion.

For many purposes, because it is global in scope, coupling facility data tables can offer significant advantages over resources such as the CICS common work area (CWA).

To an application program, a CFDT appears much like a sysplex-wide user-maintained data table: a CFDT is accessed using the same subset of the API as a UMT (that is, the full file control API except for the MASSINSERT and RBA options). However, a CFDT is restricted to a maximum key-length of 16 bytes.

Note the following comparisons with user-maintained data tables:

- Updates to a CFDT, like updates to a UMT, are not reflected in the base VSAM data set (if the table was initially loaded from one). Updates are made to the CFDT only.
- A CFDT is loaded once only, when the table is first created in the coupling facility data table, and remains in existence in the coupling facility, even when the last file referring to the CFDT is closed (whereas a UMT is deleted each time the owning region terminates). You can force a reload of a CFDT from the original source data set only by first deleting the table from the CFDT pool, using a CFDT server DELETE TABLE command. The first file opened against the CFDT after the delete operation causes the server to reload the table.

  Note: A coupling facility data table pool is defined as a coupling facility list structure, and can hold more than one data table. For information about creating a list structure for coupling facility data tables, see Coupling facility data table server parameters).

- The access rules for a UMT that is in the course of loading allow any direct read request to be satisfied either from the table (if the record has already been loaded) or from the source data set, but reject any update request, or imprecise read or browse request, with the LOADING condition. For a CFDT, any request is allowed during loading, but requests succeed only for records that are within the key range already loaded.

Coupling facility data table models

There are two models of coupling facility data table:

Contention model
This gives optimal performance, but requires programs that are written to handle the situation where the data has been changed since it issued a read-for-update request. The new CHANGED response can occur on a REWRITE or DELETE command. There is also a new use for the existing NOTFND response, which can be returned to indicate to the application program that the record has been deleted since the program issued the read-for-update request.

Note: It might be possible to use existing programs with the contention model if you are sure they cannot receive the CHANGED or NOTFND exceptions on a REWRITE or DELETE. An example of this could be where an application program operates only on records that relate to the user of the program, and therefore no other user could be updating the same records.

Locking model
This model is API-compatible with existing programs that conform to the UMT subset of the file control API. The locking model can be:

Unrecoverable
For updates to unrecoverable CFDTs, locks do not last until sync point (they are released on completion of the file control request) and updates are not backed out if a unit of work fails

Recoverable
CFDTs are recoverable in the event of a unit of work failure, and in the event of a CICS region failure, a CFDT server failure, and an MVS failure (updates made by units of work that were in-flight at the time of the failure are backed out).

The recoverable locking model supports indoubt and backout failures: if a unit of work fails when backing out an update to the CFDT, or if it fails indoubt during sync point processing, the locks are converted to retained locks and the unit of work is shunted.

CFDTs cannot be forward recoverable. A CFDT does not survive the loss of the CF structure in which it resides.

You specify the update model you want for each table on its file resource definition, enabling different tables to use different models.

Techniques for sharing data

The CICS techniques that can be used for sharing data are compared, in tabular form, to show you when to consider using a coupling facility data table.

Table 26. Techniques for sharing scratchpad data

Constraints and factors	Single Region	Single MVS	Sysplex
Technique no longer recommended (too restrictive)	TWA	—	—
Recommended method for single area for each transaction	COMMAREA or channel	COMMAREA or channel	COMMAREA or channel
Existing application programs use temporary storage (TS) queues	Local TS queue	Remote TS queue	Shared TS queue

Table 26. Techniques for sharing scratchpad data (continued)			
Constraints and factors	**Single Region**	**Single MVS**	**Sysplex**
Existing programs use UMT Random insert and delete required Multiple types of data stored	UMT	Remote UMT	CFDT (contention model)

In Table 26 on page 238 , different techniques are considered for passing scratchpad data between phases of a transaction, where only one task is accessing the data at a time, but the data can be passed from a task in one region to a task in another. 'Remote UMT' means a shared user-maintained data table that is accessed from AORs either by function shipping where necessary (that is, for update accesses) or by SDT cross-memory sharing for non-update accesses. The table shows that, within a Parallel Sysplex® , a coupling facility data table is the best solution for random insertion and deletion of data, and where multiple types of data need to be stored. Without these constraints, shared TS queues are a more appropriate choice if the application programs are already using temporary storage.

Table 27. Techniques for sharing queues of data			
Constraints and factors	**Single Region**	**Single MVS**	**Sysplex**
Read-only at head, write-only at tail Triggering required	Local transient data (TD)	Remote TD	Remote TD
Process batches of items	TS queue or UMT	Remote TS or remote UMT	Shared TS or CFDT
Delete each item after processing. Random insert and delete required.	UMT	Remote UMT	CFDT

In Table 27 on page 239 , different techniques for sharing queues of data are shown, where information is stored in a specific sequence, to be processed by another application program or task in the same sequence. The CICS transient data and temporary storage queue facilities are recommended in most cases, with a few instances where data tables provide a more appropriate solution for handling sequenced data.

Table 28. Techniques for sharing control records			
Constraints and factors	**Single Region**	**Single MVS**	**Sysplex**
Technique no longer recommended	CWA	MVS CSA	—
Single updating region, single record	TS queue or UMT	Remote TS queue or UMT	Shared TS queue or CFDT (contention model)
Multiple updating regions or multiple records	UMT	Remote UMT	CFDT

In Table 28 on page 239 , different techniques for managing control records are shown. This illustrates where a central control record is used to make information available to all transactions. For example, this can contain the next unused order number, or customer number, to make it easier for programs to create

new records in a keyed file or database. (For this type of application consider the named counter function, which is also a sysplex-wide facility. See “Named counter servers” on page 330 for details.)

The table shows that within an MVS image, if there is a single region that makes all the updates to a single record, you can use a UMT without any function shipping overheads.

Where there are multiple regions updating the control record, or there is more than one control record to update, then a coupling facility data table is the only solution within a Parallel Sysplex environment, and it could also be more effective than function shipping the updates to a UMT within a single MVS.

Table 29. Techniques for sharing keyed data

Constraints and factors	Single Region	Single MVS	Sysplex
Read-only or rarely updated	UMT	UMT	Replicated UMT
Single updating region	UMT	UMT	Replicated UMT or CFDT
Multiple updating regions Recoverable (backout only)	UMT	Remote UMT or CFDT	CFDT

In Table 29 on page 240 , different techniques for sharing keyed data are shown. This covers applications that use data similar in structure to a conventional keyed file, but where the information does not need to be stored permanently, and the performance benefits are sufficient to justify the use of main storage or coupling facility resources to store the relevant data.

This data is most appropriately accessed using the file control API, which means that within a Parallel Sysplex, the solution is to use:

- A replicated user-maintained data table where the highest performance is required, and where access is either read-only, or updates are rare and you can arrange to make these from a single region and refresh the replicated UMT in other regions
- A coupling facility data table.

Note that recovery support for UMTs is limited to transaction backout after a failure. For coupling facility data tables, recovery is also provided for CICS and CFDT server failures, and also for indoubt failures,

Transaction deadlocks

Design your applications to avoid transaction deadlocks. A transaction needs exclusive control of resources while executing file control commands.

A deadlock occurs if each of two transactions (for example, A and B) needs exclusive use of some resource (for example, a particular record in a data set) that the other already holds. Transaction A waits for the resource to become available. However, if transaction B is not in a position to release it because it, in turn, is waiting on some resource held by A, both are deadlocked and the only way of breaking the deadlock is to cancel one of the transactions, thus releasing its resources.

For both VSAM and BDAM data sets, any record that is being modified is held in exclusive control by CICS from the time when the modification begins (for example, when a READ UPDATE command is issued to obtain control of the record), to the time when it ends (for example, when a REWRITE command has finished making a change to the record).

With VSAM files accessed in RLS mode, only the individual record is ever locked during this process. With VSAM files accessed in non-RLS mode, when VSAM receives a command that requires control of the record, it locks the complete control interval containing the record. CICS then obtains an enqueue on the record that it requires, and releases the control interval, leaving only the record locked. The control interval lock is released after each command, and only the individual record is locked for the whole of the modification process. (For more information about how the control interval lock is released, see Locks.)

As well as CICS having exclusive control of a record during the modification process, there is an extra locking period when a transaction modifies a record in a **recoverable** file. In this situation, CICS (or VSAM if the file is accessed in RLS mode) locks that record to the transaction even after the request

that performed the change has completed. The transaction can continue to access and modify the same record; other transactions must wait until the transaction releases the lock, either by terminating or by issuing a sync point request. For more information, see “Syncpointing” on page 188.

Whether a deadlock occurs depends on the relative timing of the acquisition and release of the resources by different concurrent transactions. Application programs can continue to be used for some time before meeting circumstances that cause a deadlock; it is important to recognize and allow for the possibility of deadlock early in the application program design stages.

Here are examples of different types of deadlock found in recoverable data sets:

- Two transactions running concurrently are modifying records within a single recoverable file, as follows:

```
 Transaction 1
READ UPDATE record 1
UNLOCK record 1

WRITE record 2

Transaction 2
DELETE record 2

READ UPDATE record 1
REWRITE record 1
```

  Transaction 1 has acquired the record lock for record 1 (even though it has completed the READ UPDATE command with an UNLOCK command). Transaction 2 has similarly acquired the record lock for record 2. The transactions are then deadlocked because each wants to acquire the CICS lock held by the other. The CICS lock is not released until sync point.

- Two transactions running concurrently are modifying two recoverable files as follows:

```
 Transaction 1 Transaction 2
READ UPDATE file 1, record 1 READ UPDATE file 2, record 2
REWRITE file 1, record 1 REWRITE file 2, record 2

READ UPDATE file 2, record 2 READ UPDATE file 1, record 1
REWRITE file 2, record 2 REWRITE file 1, record 1
```

  Here, the record locks have been acquired on different files as well as different records; however, the deadlock is like the first example.

For VSAM files accessed in non-RLS mode, CICS detects deadlock situations, and a transaction about to enter a deadlock is abended with the abend code AFCF if it is deadlocked by another transaction, or with abend code AFCG if it has deadlocked itself.

VSAM-detected deadlocks (RLS only)

With files accessed in RLS mode, deadlocks can arise between two different CICS regions, possibly running under different MVS images. In these cases, deadlock detection and resolution cannot be performed by CICS, and therefore it is performed by VSAM.

If VSAM detects an RLS deadlock condition it returns a deadlock exception condition to CICS, causing CICS file control to abend the transaction with an AFCW abend code. CICS also writes messages and trace entries that identify the members of the deadlock chain.

However, VSAM cannot detect a cross-resource deadlock (for example, a deadlock arising from use of RLS and Db2 resources) where another resource manager is involved. A cross-resource deadlock is resolved by VSAM when the timeout period expires, and the waiting request is timed out. In this situation, VSAM cannot determine whether the timeout is caused by a cross-resource deadlock, or a timeout caused by another transaction acquiring an RLS lock and not releasing it. In the event of a timeout, CICS writes trace entries and messages to identify the holder of the lock for which a timed-out transaction is waiting.

All file control requests issued in RLS mode have an associated timeout value. This timeout value is that defined by DTIMEOUT if DTIMEOUT is active for the transaction, or FTIMEOUT from the system initialization table if DTIMEOUT is not active.

Rules for avoiding deadlocks

Use the following rules to avoid deadlocks.

- All applications that update (modify) multiple resources should do so in the same order. For instance, if a transaction is updating more than one record in a data set, it can do so in ascending key order. A transaction that is accessing more than one file should always do so in the same predefined sequence of files.

  If a data set has an alternate index, beware of mixing transactions that perform several updates by the base key with transactions that perform several updates by the alternate key. Assume that the transactions that perform updates always access records in ascending key sequence. Then transactions that perform all updates by the base key will not deadlock with other transactions that perform all updates by the base key. Likewise, transactions that perform all updates by the alternate key do not deadlock with other transactions that perform all updates by the alternate key. But transactions that perform all updates by the base key can deadlock with transactions that perform all updates by the alternate key. This is because the key that is locked is always the base key. Consequently, a transaction performing updates by the alternate key can be acquiring locks in a different order to a transaction performing updates by the base key.
- An application that issues a READ UPDATE command should follow it with a REWRITE, DELETE without RIDFLD, or UNLOCK command to release the position before doing anything else to the file, or should include the TOKEN option with both parts of each update request.
- A sequence of WRITE MASSINSERT commands must terminate with the UNLOCK command to release the position. No other operation on the file should be performed before the UNLOCK command has been issued.
- An application must end all browses on a file with ENDBR commands (releasing the VSAM lock) before issuing a READ UPDATE, WRITE, or DELETE with RIDFLD command, to the file.

File control operations

CICS file control has an option for you to read, update, add, and browse data in VSAM and BDAM data sets and delete data from VSAM data sets. You can also access CICS shared data tables and coupling facility data tables by using file control.

CICS processing after waiting for VSAM resources

CICS file control does not organize which tasks that wait on resources (such as VSAM exclusive control of a control interval) are redispatched when the resource becomes available. When the VSAM releases exclusive control of a control interval, the CICS dispatcher chooses the next task to be dispatched based on factors such as task priority and priority aging. It is not assured that tasks will be redispatched in the order in which they suspended on the resource to become available.

Using CICS commands to read records

There are three ways for application programs to read records: direct reading using the READ command, sequential reading, and skip-sequential reading. All these methods for reading records can be used on both VSAM and BDAM data sets.

About this task

A file can be defined in the file definition as containing either fixed-length or variable-length records. Define fixed-length records only if:

- The definition of the VSAM data set (using access method services) specifies an average record size that is equal to the maximum record size
- ***and*** All the records in the data set are of that length.

For direct reading and browsing, if the file contains fixed-length records, and if the application program provides an area into which the record is to be read, that area must be of the defined length. If the file

contains variable-length records, the command must also specify the length of the area provided to hold them (which is typically the maximum length of records in the file).

For fixed-length records and for records retrieved into storage provided by CICS (when you use the SET option), you need not specify the LENGTH argument. However, although the LENGTH argument is optional, you are recommended to specify it when using the INTO option, because it causes CICS to check that the record being read is not too long for the available data area. If you specify LENGTH, CICS uses the LENGTH field to return the actual length of the record retrieved.

Direct reading (using READ command)
Records in a file can be read using the READ command with the RIDFLD (record identification field) option to specify a record. Further options are available which describe the content and length of the record identification field.

About this task

When you use the READ command:

Procedure

- Use the RIDFLD (record identification field) option to specify the record you want, and add further options to describe the content of the record identification field.

  The exact method of identifying the record depends on the type of data set:

  a) For a KSDS, you can identify the record you want uniquely by specifying its full key, or you can specify a generic key and retrieve the first (lowest key) record whose key meets those requirements.

  – The GENERIC option indicates that you require a match on only a part of the key. When you specify the GENERIC option, you must also specify the KEYLENGTH option, to say how many positions of the key, starting from the left, must match. For the READ command, CICS uses only the first KEYLENGTH option characters.

  – The GTEQ (greater than or equal to) option indicates that you want the first record whose key is "greater than or equal to" the key you have specified. You can use GTEQ with either a full or a generic key.

  – The opposite of the GTEQ option is the EQUAL option (the default), which means that you want only a record whose key matches exactly that portion (full or generic) of the key that you have specified.

  If a KSDS has an alternate index and an alternate index path, you can retrieve a record in the file by specifying the alternate key that you set up in the alternate index (see step “3” on page 243).

  b) For a standard (non-extended) ESDS, you can identify the record by specifying its relative byte address (RBA).

  Add the RBA option to inform CICS that an RBA is being used. Because the RBA of a record in an ESDS cannot change, your application program can keep track of the values of the RBAs corresponding to the records it wants to access. An RBA must always point to the beginning of a record. There is no equivalent to the GENERIC or GTEQ options that you can use for a KSDS.

  c) For an extended ESDS, you can identify the record by specifying its extended relative byte address (XRBA).

  Add the XRBA option to inform CICS that an XRBA is being used. Because the XRBA of a record in an ESDS cannot change, your application program can keep track of the values of the XRBAs corresponding to the records it wants to access. An XRBA must always point to the beginning of a record. There is no equivalent to the GENERIC or GTEQ options that you can use for a KSDS.

  d) For a KSDS or a standard ESDS with an alternate index (an extended ESDS cannot have an alternate index) , you can retrieve a record in the file by specifying the alternate key that you set up in the alternate index.

  When you use an alternate key:

- The GENERIC option and the GTEQ option can be used for both KSDS and ESDS records, in the same way as for a read from a KSDS using the primary key.
- If the alternate key is not unique, the first record in the file with that key is read and you get the DUPKEY condition. To retrieve other records with the same alternate key, you have to start a browse operation at this point.
- If no matching record is found, the record identified is the one with the next higher alternate key.

e) For an RRDS, identify the record by specifying its relative record number (RRN). Add the RRN option to inform CICS that an RRN is being used.

The application program must know the RRN values of the records it wants. There is no equivalent to the GENERIC or GTEQ options that you can use for a KSDS, and you cannot use an alternate key.

- Specify the KEYLENGTH option if required, to state the length of the key which you specified in the RIDFLD option.

a) If you specify an RBA or RRN for the record identification field, and specify the RBA or RRN option, the KEYLENGTH option **is not** required.

b) If you specify the GENERIC option, the KEYLENGTH option **is** required. Specify the length of the generic key, which must be less than the key length specified in the VSAM definition.

c) If you are reading a remote file and specify the SYSID option, the key length can be identified by any of the following means:

- Specified in the file definition.
- Specified by the application program using the KEYLENGTH option.
- Defaulted to 4, if the key is longer than 4 characters, and the key length is not specified in the file definition or by the application program.

This is different from the situation for the WRITE command, where the KEYLENGTH option is required for remote files, unless you have used an RBA or RRN.

d) In other situations, the KEYLENGTH option may be specified or omitted.

If you specify a KEYLENGTH which is different from the length defined for the data set and the operation is not generic, the INVREQ condition occurs.

- Use the INTO or SET option to state whether the record is to be read into an area of main storage provided by your application program (READ INTO), or into an area of main storage CICS SET storage acquired by file control (READ SET).

  If the latter, the address of the data in the CICS SET storage is returned to your program.

  CICS SET storage normally remains valid until the next syncpoint, or the end of the task, or until next READ against the same file, whichever comes first.
- Make sure that the LENGTH parameter is set if required:

a) If you have used the SET option, you do not have to specify the LENGTH option. However, you may specify it if you want to check whether the record length agrees with that originally defined to VSAM.

The data area specified is set to the actual record length after the record has been retrieved.

b) For fixed-length records, you do not have to specify the LENGTH option, but it is advisable to do so.

If you specify the option, CICS checks that the record is not too long for the available data area. The length you specify should exactly match the record length specified at the time the file was created.

c) If you have used the INTO option for reading variable-length records, a length parameter must be provided.

However, the LENGTH option does not always need to be specified explicitly on the command:

- For a file on a remote system, the LENGTH parameter need not be set here, but it must be set in the file resource definition.
- If you are using assembler language or PL/I, instead of specifying LENGTH explicitly, you can allow it to be defaulted using the length attribute reference in assembler language, or STG and CSTG in PL/I. LENGTH must be specified explicitly in C.

If you do specify the LENGTH option explicitly when reading a variable-length file, always specify the longest record your application program accepts, which should correspond to the value defined as the maximum record size when the data set was created.

If the record exceeds the length specified, the LENGERR condition occurs, and the record is then truncated to that length.

After the record has been retrieved, the data area specified in it is set to the actual record length (before any truncation occurs).

- If the file is open in RLS mode, use the UNCOMMITTED, CONSISTENT, REPEATABLE and NOSUSPEND options to control read integrity.

  a) If you specify the UNCOMMITTED option, there is no read integrity and shared locks are not used for read requests.

  "RLS Record level locking" on page 231 explains shared and exclusive locks.

  This is the default and is the way in which file control works for files that are opened in non-RLS mode.

  b) If you specify the CONSISTENT option, a request to read a record is queued if the record is being updated by another task.

  The read completes only when the update is complete, and the updating unit of work (UOW) relinquishes its exclusive lock. "Syncpointing" on page 188 explains UOWs and syncpoints.

  c) If you specify the REPEATABLE option, processing of the read request is the same as for consistent read requests, except that the reader holds on to its shared lock until syncpoint.

  This applies to both recoverable and non-recoverable files. This ensures that a record read in a UOW cannot be modified while the UOW makes further read requests. It is particularly useful when you issue a series of related read requests and you want to ensure that none of the records is modified before the last record is read.

  d) If you specify either CONSISTENT or REPEATABLE, you can also specify the NOSUSPEND option to ensure that the request does not wait if the record is locked by VSAM with an active lock.

  "Active and retained states for locks" on page 233 explains active locks.

  If you do not specify any of the options, the value from the file resource definition is used.

  Note: Specify read integrity only when an application cannot tolerate 'stale' data. This is because RLS uses locks to support read integrity, and as a result your applications could encounter deadlocks that do not occur in releases of CICS that do not support read integrity. This is particularly important if you define read integrity on file resource definitions. The application programs that reference these files may have been written for releases of CICS that do not support read integrity, and are not designed to deal with deadlock conditions on read-only file accesses.

Sequential reading (browsing)
Use the **STARTBR** , **READNEXT** , **READPREV** , and **RESETBR** commands to browse records in a file. You can browse forwards or backwards, and change the position or characteristics of the browse while the browse is in progress. Use the **ENDBR** , **SYNCPOINT** and **SYNCPOINT ROLLBACK** commands to end the browse.

About this task

When browsing records, you generally identify and read the records in the same way as for a direct read using the **READ** command (see "Direct reading (using READ command)" on page 243). You specify the record identification field (RIDFLD option), and select a destination for the read records. If the file is open in RLS mode, you can use the UNCOMMITTED , CONSISTENT , REPEATABLE and NOSUSPEND options to control read integrity. Some additional special cases for browsing are noted in this topic.

The same types of key are used for browsing as for a direct read:

- For a KSDS, a full key, a generic key, or an alternate key can be used.
- For a standard (non-extended) ESDS, an RBA or an alternate key can be used.
- For an extended ESDS, an extended RBA (XRBA) can be used.

Note: In some circumstances, a 32-bit RBA can be used to access an extended (64-bit) ESDS: see "Relative record number (RRN), relative byte address (RBA) and extended relative byte address (XRBA)" on page 228.

- For an RRDS, an RRN is used.

If you use an alternate key for browsing, the records are retrieved in alternate key order. If the alternate key is not unique, the DUPKEY condition is raised for each retrieval operation except the last occurrence of the duplicate key. For example, if there are three records with the same alternate key, DUPKEY is raised on the first two, but not the third. The order in which records with duplicate alternate keys are returned is the order in which they were written to the data set. This is true whether you are using a **READNEXT** or a **READPREV** command. For this reason, you cannot retrieve records with the same alternate key in reverse order.

CICS allows a transaction to perform more than one browse on the same file at the same time. You distinguish between browse operations by including the REQID option on each browse command.

The instructions in this topic cover general principles for browsing, and specific information for the different types of VSAM data set. For specific information about browsing BDAM data sets, see "Browsing records from BDAM data sets" on page 248.

Procedure

- To start a browse, use the **STARTBR** command.

  The **STARTBR** command only identifies the starting position for the browse; it does not retrieve a record.

  Identify a particular record in the same way as for a direct read, and use the RIDFLD option to specify the record identification. See "Direct reading (using READ command)" on page 243 , and also note the following considerations:

  a) To position the browse at the start of the file, for a KSDS or ESDS, specify a RIDFLD of hexadecimal zeros.

     For a standard ESDS, also specify the RBA option. For an extended ESDS, specify the XRBA option.

  b) To position the browse at the start of the file, for an RRDS, specify an RRN of 1 using the RIDFLD option, and also the RRN option.

  c) For a KSDS only, as an alternative method to position the browse at the start of the file, you may specify the options GENERIC , GTEQ , and KEYLENGTH(0).

     You need the RIDFLD keyword although its value is not used, and, after the command completes, CICS is using a generic key length of one.

     A browsing command with the option KEYLENGTH(0) is always treated as if KEYLENGTH(1) and a partial key of one byte of binary zeros have been specified.

  d) To position the browse at the last record in the file, ready for a backward browse, specify a RIDFLD of X'FF' characters.

     This applies to starting a browse of a KSDS, ESDS, or RRDS. For a standard ESDS, specify the RBA option. For an extended ESDS, specify the XRBA option. For an RRDS, specify the RRN option.

  e) For starting a browse of a KSDS, you may specify a generic key using the RIDFLD option.

     However, if you use a generic key, you can only browse forwards through the file, not backwards.

  f) For starting a browse of a KSDS, you may use the options EQUAL (key equal to) and GTEQ (key greater than or equal to), and they have the same meaning as on the **READ** command.

     However, option GTEQ is the default for the STARTBR command, whereas EQUAL is the default for the **READ** command. The **STARTBR** command assumes that you want to position at the key specified or the first key greater if that key does not exist.

  g) For starting a browse of an RRDS, the GTEQ (key greater than or equal to) option is the default on a STARTBR command.

     When this option is in effect, if the specified RRN does not exist, a pointer is set to the first record having a greater key.

Note that the GTEQ option has no effect on a direct **READ** command for a RRDS, and if a direct **READ** command specifies an RRN that does not exist, the NOTFND condition is returned.

- To browse through the records, use the **READNEXT** command.

  The **READNEXT** command reads records sequentially from the starting point specified by the **STARTBR** command. It operates in a similar way to a direct read. See "Direct reading (using READ command)" on page 243 , and also note the following considerations:

  a) Include the RIDFLD option to give CICS a way to return the identifier of each record retrieved, but do not set the field, unless you want to reposition the browse.

     On completion of each **READNEXT** command, CICS returns the full key of the record it retrieved. Be sure to provide a field as long as the full key, even if you use a **STARTBR** command with a shorter generic key.

  b) You only need the KEYLENGTH option if you are providing a key using the RIDFLD option, to reposition the browse.

     Otherwise, the current key length is used, as set on the **STARTBR** command or during the last repositioning of the browse.

  c) For remote files for which the SYSID option has been specified, the KEYLENGTH option must be specified if the RIDFLD option is passing a key to file control. If the remote file is being browsed, the KEYLENGTH option is not required for the **READNEXT** or **READPREV** commands.

  d) As for a direct read, you can read the record into an area supplied by the application program (when you use the INTO option), or into storage provided by CICS (when you use the SET option).

     In the latter case, the CICS storage addressed by the SET pointer remains valid until the next operation in the browse, or until the browse ends, syncpoint, or end of task, whichever occurs first.

- To browse backwards in the file, use **READPREV** commands instead of **READNEXT** commands.

  The **READPREV** command is like the **READNEXT** command, except that the records are read sequentially **backward** from the current position. You can switch from one direction to the other at any time.

  Note the following considerations:

  a) As you switch from one direction to the other, you retrieve the same record twice, unless you reposition.

  b) If you issue a **READPREV** command immediately following a **STARTBR** command, your **STARTBR** command RIDFLD must specify the key of a record that exists on the data set; otherwise the NOTFND condition occurs.

     To avoid this response, you could issue a **READNEXT** command following the **STARTBR** command, to return a record that exists on the data set. Then issue the **READPREV** command, which retrieves the same record that was retrieved by the **READNEXT** command.

  c) For a KSDS, if you used a generic key on the **STARTBR** command, you cannot use the **READPREV** command. An INVREQ condition is returned if you try to use the command in this case.

- To change the current browse position when the browse has started (reposition the browse), use either the **RESETBR** command, or the **READNEXT** command, or the **READPREV** command.

  a) For VSAM, you can reposition the browse by setting a value in RIDFLD when you issue the next **READNEXT** or **READPREV** command.

     – When you set RIDFLD to reposition a browse, the record identifier must be in the same form as on the previous **STARTBR** or **RESETBR** command (key, RBA, XRBA , or RRN). You can use a key, or use hexadecimal zeroes to indicate the beginning of the file, or for a KSDS, specify the options GENERIC , GTEQ , and KEYLENGTH(0) to indicate the beginning of the file. If you use the KEYLENGTH(0) option, note that you need the RIDFLD keyword although its value is not used, and, after the command completes, CICS is using a generic key length of one. (X'FF' characters cannot be used for repositioning with the **READNEXT** or **READPREV** command.)

     – You can change the length of a generic key by specifying a KEYLENGTH in your **READPREV** command, which is different from the current generic key length and not equal to the full length.

If you change the length of a generic key in this way, you reposition to the generic key specified by the RIDFLD option.

If you also want to change characteristics of the browse, use the **RESETBR** command instead.

b) The **RESETBR** command can specify a new browse position in the same ways as the **STARTBR** command.

You can:

- Identify a specific record.
- Use a key of hexadecimal zeros to indicate the beginning of the file.
- Use a key of X'FF' characters to indicate the end of the file.
- For a KSDS, use the options GENERIC , GTEQ , and KEYLENGTH(0) to indicate the beginning of the file. If you use the KEYLENGTH(0) option, note that you need the RIDFLD keyword although its value is not used, and, after the command completes, CICS is using a generic key length of one.

- To change the characteristics of the browse (for example, to search with a generic key instead of an exact match), use the **RESETBR** command.

  You can use the command to:

  - Change the form of the key from key to RBA.
  - Switch between generic and full keys, or between "equal to" and "greater than or equal to" searches.

  Under certain conditions, CICS uses VSAM skip-sequential processing when you change the key, as explained in “Skip-sequential processing” on page 249.

- To end the browse, use the **ENDBR** command. There is no RIDFLD for this command.

  Trying to browse past the last record in a file raises the ENDFILE condition. You must issue the **ENDBR** command before performing an update operation on the same file (a **READ UPDATE** , **DELETE** with RIDFLD , or **WRITE** command). If you do not, you get unpredictable results, possibly including deadlock within your own transaction. You can also end the browse by using the **SYNCPOINT** and **SYNCPOINT ROLLBACK** commands. If an implicit sync point occurs at the end of a task, the browse also ends.

Browsing records from BDAM data sets

The record identification field must contain a block reference (for example, TTR or MBBCCHHR) that conforms to the addressing method defined for the data set. Processing begins with the specified block and continues with each subsequent block until you end the browse.

If the data set contains blocked records, processing begins at the first record of the first block and continues with each subsequent record, regardless of the contents of the record identification field.

In other words, CICS uses only the information held in the TTR or MBBCCHHR subfield of the RIDFLD to identify the record. It ignores all other information, such as physical key and relative record, or logical key.

After the READNEXT command, CICS updates the RIDFLD with the complete identification of the record retrieved. For example, assume that a browse is to be started with the first record of a blocked, keyed data set, and deblocking by logical key is to be performed. Before issuing the STARTBR command, put the TTR (assuming that is the addressing method) of the first block in the record identification field. After the first READNEXT command, the record identification field might contain X'0000010504' , where X'000001' represents the TTR value, X'05' represents the block key, (of length 1), and X'04' represents the logical record key.

Now assume that a blocked, nonkeyed data set is being browsed using relative record deblocking and the second record from the second physical block on the third relative track is read by a command, put the TTR (assuming that is the addressing method) of the first block in the record identification field. After the first READNEXT command. Upon return to the application program, the record identification field contains X'00020201' , where X'0002' represents the track, X'02' represents the block, and X'01' represents the number of the record in the block relative to zero.

Note: Specify the options DEBREC and DEBKEY on the STARTBR command when browsing blocked-data sets. This enables CICS to return the correct contents in the RIDFLD. Specifying DEBREC on the STARTBR command causes the relative record number to be returned. Specifying DEBKEY on the STARTBR command causes the logical record key to be returned.

Do not omit DEBREC or DEBKEY when browsing a blocked file. If you do, the logical record is retrieved from the block, the length parameter is set equal to the logical record length, ***but*** the RIDFLD is not updated with the full identification of the record. ***Do not use this method*** .

Compare this with what happens if you omit the DEBREC or DEBKEY option when reading from a blocked BDAM data set. In this case, you retrieve the ***whole*** block, and the length parameter is set equal to the length of the block.

For a remote BDAM file, where the DEBKEY or DEBREC options have been specified, KEYLENGTH (when specified explicitly) should be the total length of the key (that is, all specified subfields).

Skip-sequential processing
When possible, CICS uses VSAM "skip-sequential" processing to speed browsing. Skip-sequential processing applies only to forward browsing of a file when RIDFLD is specified in key form.

CICS uses it when you increase the key value in RIDFLD on your READNEXT command and make no other key-related specification, such as KEYLENGTH. In this situation, VSAM locates the next wanted record by reading forward through the index, rather than repositioning from scratch. This method is faster if the records you are retrieving are relatively close to each other but not necessarily adjacent; it can have the opposite effect if the records are far apart in a large file. If you know that the key you are repositioning to is much higher in the file, and that you can incur a long index scan, you might want to consider using a RESETBR command which forces a reposition from scratch.

Using CICS commands to update records

To update a record, you must first retrieve it using one of the file control read commands that specifies the UPDATE option. The record is identified in the same way as for a direct read.

About this task

In a KSDS or ESDS, the record can (as with a direct read) be accessed by way of a file definition that refers either to the base, or to a path defined to it. For files opened in RLS mode, you can specify the NOSUSPEND option as well as the UPDATE option on an EXEC CICS command to ensure that the request does not wait if the record is already locked by VSAM with an active lock.

After modification by the application program, the record is written back to the data set using the REWRITE command, which does not identify the record being rewritten. Within a unit of work, each REWRITE command should be associated with a previous READ UPDATE by a common keyword (TOKEN). You can have one READ UPDATE without a TOKEN outstanding at any one time. Attempts to perform multiple concurrent update requests within a unit of work, upon the same data set without the use of TOKENS, are prevented by CICS. If you want to release the string held by a READ UPDATE without rewriting or deleting the record, use the UNLOCK command. The UNLOCK command releases any CICS storage acquired for the READ command, and releases VSAM resources held by the READ command. If TOKEN is specified with the UNLOCK command, CICS attempts to match this with an outstanding READ UPDATE whose TOKEN has the same value. (For more explanation about the TOKEN option, see “The TOKEN option” on page 250 .)

For both update and non-update commands, you must identify the record to be retrieved by the record identification field specified in the RIDFLD option. Immediately on completion of a READ UPDATE command, the RIDFLD data area is available for reuse by the application program. The REWRITE and UNLOCK commands do not use the RIDFLD option.

A record retrieved as part of a browse operation can only be updated during the browse if the file is opened in RLS mode (see “Updating and deleting records in a browse (VSAM RLS only)” on page 252). For files opened in non-RLS mode, the application program must end the browse, read the wanted record with a READ UPDATE command, and perform the update. Failure to end the browse before issuing the READ UPDATE command can cause a deadlock.

The record to be updated can (as in the case of a direct read) be read into an area of storage supplied by the application program or into storage set by CICS. For a READ UPDATE command, CICS SET storage remains valid until the next REWRITE, UNLOCK, DELETE without RIDFLD, or SYNCPOINT command, whichever is encountered first.

For a KSDS, the base key in the record must not be altered when the record is modified. Similarly, if the update is being made by way of a path, the alternate key used to identify the record must not be altered either, although other alternate keys can be altered. If the file definition allows variable-length records, the length of the record can be changed.

Specify the record to be written with the FROM option. The FROM option specifies the main storage area that contains the record to be written. In general, this area is part of the storage owned by your application program. With the REWRITE command, the FROM area is typically (but not necessarily) the same as the corresponding INTO area on the READ UPDATE command.

The length of records in an ESDS, a fixed-length RRDS, or a fixed-length KSDS must not be changed on update. However, the length of the record can be changed when rewriting to a KSDS with variable-length records.

For a file defined to CICS as containing fixed-length records, the length of record being rewritten ***must equal the original length*** . However, when writing to a file with fixed-length records, you need not specify the LENGTH option. If you do, its value is checked against the defined value and you get a LENGERR condition if the values do not match.

For variable-length records, you must specify the LENGTH option with both the READ UPDATE and the REWRITE commands. The length must not be greater than the maximum defined to VSAM.

The TOKEN option
The TOKEN option is a unique value within a task that is supplied by CICS on any valid read for update command, and you return this to CICS with an associated REWRITE, DELETE, or UNLOCK command. For each file that is being updated by a task, at any one time you can have only one outstanding read request with the UPDATE option that does not specify the TOKEN option.

You can perform multiple concurrent updates on the same data set using the same task by including the TOKEN option with a read for update command, and specifying the token on the associated REWRITE, DELETE, or the UNLOCK command. Note that, for files accessed in non-RLS mode, a set of concurrent updates fails if more than one record is being updated in the same CI, irrespective of the TOKEN associated with the request. Also, only one token remains valid for a given REQID on a browse, and that is the one returned on the last READNEXT or READPREV command (see "Updating and deleting records in a browse (VSAM RLS only)" on page 252).

You can function ship a read for update request containing the TOKEN option. However, if you function ship a request specifying TOKEN to a member of the CICS family of products that does not recognize this keyword, the request fails.

Conditional VSAM file update requests
On file control update requests against files opened in RLS mode, you can avoid waiting for a lock by making your request conditional upon being given a lock immediately. You do this by specifying the NOSUSPEND option on the request. If another task already holds an active lock, CICS returns the RECORDBUSY condition instead of queuing your request.

You can specify NOSUSPEND on READ, READNEXT, READPREV, WRITE, REWRITE, and DELETE commands.

It is important to distinguish between the LOCKED and RECORDBUSY responses:

- A LOCKED response occurs when a request attempts to access a record that is locked by a *retained* lock.
- A RECORDBUSY response occurs when a request attempts to access a record that is locked by an active lock. Remember that this could be caused by a DEADLOCK, in which case retries might not work. It might be necessary to issue a SYNCPOINT with or without rollback to resolve the condition.

  Note: Requests that specify NOSUSPEND wait for at least 1 second before CICS returns the RECORDBUSY response.

If you do not specify NOSUSPEND on your request, CICS causes it to wait for a lock if the record is already locked by an active lock. If you specify NOSUSPEND, your request receives a RECORDBUSY response if the record is locked by an active lock.

If you issue a request (with or without the NOSUSPEND option) against a record locked by a retained lock, CICS returns a LOCKED response.

There is a slight difference in the way that NOSUSPEND works on file control commands compared with the way that NOSUSPEND works on other CICS commands. If you issue HANDLE CONDITION(RECORDBUSY) it does not cause NOSUSPEND to be assumed on subsequent file control requests. Specifying HANDLE CONDITION(QBUSY) causes NOSUSPEND to be assumed on subsequent transient data commands even when it is not explicitly specified.

Updating records from BDAM data sets
You cannot change the record length of a variable blocked or unblocked BDAM record on a REWRITE command which specifies deblocking. You cannot change the record length of an undefined format BDAM record on a REWRITE command either.

Note: When a blocked BDAM record is read for update, CICS maintains exclusive control of the containing block. An attempt to read a second record from the block before the first is updated (by a REWRITE command), or before exclusive control is released (by an UNLOCK command), causes a deadlock.

Using CICS commands to delete records

Use the DELETE command to delete a record or group of records from a file. You may retrieve the record for update first. For a file opened in RLS mode, you may delete a record during a browse.

About this task

If a full key is provided with the DELETE command, a single record with that key is deleted. So, if the data set is being accessed by way of an alternate index path that allows non-unique alternate keys, only the first record with that key is deleted. After the deletion, you know whether further records exist with the same alternate key, because you get the DUPKEY condition if they do.

The NOSUSPEND option discussed in “Direct reading (using READ command)” on page 243 , applies to the CICS browse commands when you are using them to update a file.

Procedure

- Records can never be deleted from an ESDS.
- To delete a single record in a KSDS or RRDS, use one of these three methods:
  a) Retrieve it for update with a READ UPDATE command, and then issue a DELETE command without specifying the RIDFLD option.
  b) Issue a DELETE command specifying the RIDFLD option.
  c) For a file opened in RLS mode, retrieve the record with a READNEXT or READPREV command with the UPDATE option, and then issue a DELETE command. This method is described in “Updating and deleting records in a browse (VSAM RLS only)” on page 252.
- To delete groups of records in a KSDS or RRDS, use a generic key with the DELETE command.

  Instead of deleting a single record, all the records in the file whose keys match the generic key are deleted with the single command. If access is by way of an alternate index path, the records deleted are all those whose alternate keys match the generic key. However, this method cannot be used if the KEYLENGTH value is equal to the length of the whole key (even if duplicate keys are allowed). The number of records deleted is returned to the application program if the NUMREC option is included with the command.
- The **DELETE** command with GENERIC option should be used with caution. If the generic key specified will match a very large number of records, then the CICS unit of work will be holding a very large number of record locks until syncpoint. This could cause storage issues within the CICS region, or if the file is an RLS file, it could cause excessive use of coupling facility resources and affect other systems.

Instead, consider using a series of generic deletes (by using generic keys that will match less records) and syncpoint after each range of records has been deleted.

Updating and deleting records in a browse (VSAM RLS only)

For files accessed in RLS mode, you can specify the UPDATE option on a READNEXT or READPREV command and then update or delete the record by issuing a DELETE or REWRITE command.

About this task

If the browse command returns a TOKEN, the TOKEN remains valid only until the next browse request. The TOKEN is invalidated by REWRITE, DELETE, or UNLOCK commands, that specify the same value for TOKEN or by the commands READNEXT, READPREV, or ENDBR that specify the same REQID. If you issue many READNEXT commands with the UPDATE and TOKEN options, the TOKENS invalidate each other and only the last one is usable. (For more explanation about the TOKEN option, see "The TOKEN option" on page 250 .)

Use of the UPDATE option in a browse is subject to the following rules:

- You can specify UPDATE within a browse only if the file is accessed in RLS mode. If you specify UPDATE for a file accessed in non-RLS mode, CICS returns an INVREQ condition.
- You can specify UPDATE only on the READNEXT and READPREV commands, not on the STARTBR or RESETBR commands.
- CICS supports only one TOKEN in a browse sequence, and the TOKEN value on each READNEXT or READPREV command overwrites the previous value.
- You can mix update and non-update requests within the same browse.
- You must specify on the READNEXT, DELETE, or UNLOCK command the TOKEN to be returned by the corresponding READNEXT or READPREV command.

Locks for UPDATE

Specifying UPDATE on a READNEXT or READPREV command acquires an exclusive lock. The duration of these exclusive locks within a browse depends on the action your application program takes and on whether the file is recoverable or not.

- If the file is recoverable and you decide to DELETE or REWRITE the last record acquired by a read for update in a browse (using the associated token), the VSAM exclusive lock remains active until completion of the UOW. That is, until successful sync point or rollback.
- If the file is not recoverable and you decide to DELETE or REWRITE the last record acquired, the lock is released either when you next issue an ENDBR command or when you issue a subsequent READNEXT or READPREV command. This is explained more fully in "RLS Record level locking" on page 231.
- If you decide *not* to update the last record read, CICS frees the exclusive lock either when your program issues another READNEXT or READPREV command in the browse, or ends the browse.

Note: An UNLOCK command does *not* free an RLS exclusive lock held by VSAM against a record acquired during a browse operation. An UNLOCK within a browse invalidates the TOKEN returned by the last request. Another READNEXT or READPREV in the browse also invalidates the TOKEN for the record read by the previous READNEXT or READPREV UPDATE command. Therefore, it is not necessary to use UNLOCK in an application program that decides not to update a particular record.

Using CICS commands to add records

Add new records to a file in a VSAM data set using the WRITE command. New records must always be written from an area provided by the application program.

About this task

These instructions cover general principles for writing records, and specific information for the different types of VSAM data set. For specific information about adding records to BDAM data sets, see "Adding records to BDAM data sets" on page 254.

When you use the WRITE command:

Procedure

- To add a record to a KSDS, use the RIDFLD (record identification field) option to specify the base key of the record.

  The base key of the record identifies the position in the data set where the record is to be inserted. Although the key is part of the record, CICS also requires the application program to specify the key separately.
- To add a record to an ESDS base data set, the record must be added to the end of the file.

  You cannot insert a record in an ESDS between existing records. After the operation is completed, the relative byte address in the file where the record was placed is returned to the application program.
- To add a record to a KSDS or standard ESDS by way of an alternate index path, use the RIDFLD option to specify the alternate key.

  For a KSDS, the record is inserted into the data set in the position determined by the base key. For an ESDS, the record is placed at the end of the data set.

  Note: Extended ESDS data sets do not support alternate indexes.
- To add a record to an RRDS, use the RIDFLD option to specify the relative record number.

  The record is stored in the file in the position corresponding to the RRN.
- Specify the KEYLENGTH option if required, to state the length of the key which you specified in the RIDFLD option.

  a) If you specify an RBA, XRBA, or RRN for the record identification field, and specify the RBA, XRBA, or RRN option, the KEYLENGTH option **is not** required.

  b) In all other situations, the KEYLENGTH option **is** required.

  If you specify a KEYLENGTH which is different from the length defined for the data set, the INVREQ condition occurs.
- Use the FROM option to specify the main storage area that contains the record to be written.

  In general, this area is part of the storage owned by your application program.
- Specify the LENGTH option if required.

  a) When writing to a file with fixed-length records, the LENGTH option is not required.

  CICS uses the length specified in the cluster definition as the length of the record to be written. If you do specify the LENGTH option, its value is checked against the defined value, and you get a LENGERR condition if the values do not match.

  b) When writing to a file with variable-length records, always include the LENGTH option.

  If the value specified exceeds the maximum allowed in the cluster definition, LENGERR is raised when the command is executed. LENGERR is also raised if the LENGTH option is omitted when accessing a file with variable-length records.
- If the file is open in RLS mode, and there is a possibility that the record might already be locked, you can specify the NOSUSPEND option.

  The NOSUSPEND option ensures that the request does not wait if the record is locked by VSAM with an active lock. “Active and retained states for locks” on page 233 explains active locks.
- If you have more than one record to add to a KSDS, ESDS, or path, and the keys in successive requests are in ascending order, using the MASSINSERT option improves performance.

  (The performance improvement is only obtained if the keys are in ascending order.)

  a) Specify the MASSINSERT option on each WRITE command in the sequence.

  b) When the MASSINSERT operation is complete and all records have been written, issue an UNLOCK command to ensure that all the records are written to the file and the position is released.

  Always issue an UNLOCK command before performing an update operation on the same data set (read update, delete with RIDFLD, or write). If you do not, you might get unpredictable results, possibly including a deadlock.

If you do not issue the UNLOCK command, the MASSINSERT operation is completed when a sync point is issued, or at task termination.

Note: A READ command does not necessarily retrieve a record that has been added by an incomplete MASSINSERT operation.

See VSAM data sets in “Efficient data set operations” on page 255 for more information about MASSINSERT.

CICS locking for writing to ESDS
CICS write operations to ESDS are single threaded, for both RLS and non-RLS mode access. However, the lock held for serialization can be held for slightly longer for RLS-mode access compared with non-RLS mode. You can compensate for the possible increase in overhead by increasing the task priority of those transactions that add new records to ESDS files.

It is possible that when you switch an ESDS RLS mode from non-RLS mode that you might see an increase in timeouts for those transactions that add new records.

Using RLS mode access for an ESDS can also cause availability problems. If a CICS region fails while writing to an ESDS, the data set might be locked until the CICS region is restarted. It is recommended that you do not use RLS mode access for ESDS.

Adding records to BDAM data sets
Records added to a BDAM data set can be undefined or variable-length (keyed or nonkeyed), keyed fixed-length, nonkeyed fixed-length, or variable-length blocked.

About this task

When adding records to a BDAM data set, bear in mind the following:

- When adding undefined or variable-length records (keyed or nonkeyed), you must specify the track on which each new record is to be added. If space is available on the track, the record is written following the last previously written record, and the record number is put in the "R" portion of the record identification field of the record. The track specification can be in any format except relative block. If you use zoned-decimal relative format, the record number is returned as a 2 byte zoned decimal number in the seventh and eighth positions of the record identification field.

  The extended search option allows the record to be added to another track if no space is available on the specified track. The location at which the record is added is returned to the application program in the record identification field being used.

  When adding records of undefined length, use the LENGTH option to specify the length of the record. When an undefined record is retrieved, the application program must find out its length.
- When adding keyed fixed-length records, you must first format the data set with dummy records or "slots" into which the records can be added. You signify a dummy record by a key of X'FF' s. The first byte of data contains the record number.
- When adding nonkeyed fixed-length records, give the block reference in the record identification field. The new records are written in the location specified, destroying the previous contents of that location.
- When adding keyed fixed-length records, track information only is used to search for a dummy key and record, which, when found, is replaced by the new key and record. The location of the new record is returned to the application program in the block reference subfield of the record identification field.

  For example, for a record with the following identification field:

  ```
  0 3 0 ALPHA
  T T R KEY
  ```

  the search starts at relative track three. When control is returned to the application program, the record identification field is:

  ```
  0 4 6 ALPHA
  ```

showing that the record is now record six on relative track four.

- When adding variable-length blocked records you must include a 4 byte record description field (RDF) in each record. The first 2 bytes specify the length of the record (including the 4 byte RDF); the other 2 bytes consist of zeros.

Efficient data set operations

The efficiency of database and data set operations is an important factor in the performance of any CICS system. Use the following information to maximize response times using VSAM and BDAM data sets.

In VSAM, the main impact on efficiency, and thus on response time, comes from contention for serial-use resources (record keys, control intervals, and strings), and for storage use and processor overhead. As is typical in these situations, any improvements you make in one area can be at the expense of other areas.

VSAM data sets

To minimize contention delays using VSAM data sets:

- Minimize the time that VSAM resources are reserved for exclusive use. The exclusive use enqueue is the way CICS and VSAM prevent concurrent updates.

  If you use VSAM record-level sharing, described in “Accessing files in RLS mode” on page 226 , VSAM locks a record that has been requested for update, so that no other transaction can attempt to update the record at the same time. If the file is recoverable, VSAM releases the lock at the next sync point. If the file is not recoverable, VSAM releases the lock when the request is complete. The recoverability of a file is defined in the integrated catalog facility (ICF) catalog.

  If you do not use VSAM record-level sharing, CICS serializes update requests by base cluster record key. The complete VSAM control interval (CI) containing the requested record is held for exclusive use while an individual command (for example, a READ command with the UPDATE option) is being executed on the record. Once each command is complete, the control interval is released, and only the requested record remains locked. For unrecoverable data sets, both the VSAM exclusive use and the CICS exclusive use of the record end when the update request is complete in VSAM terms; for example, when the REWRITE command has completed. For recoverable data sets, however, the CICS exclusive use does not end until the task ends or issues a SYNCPOINT command. Recoverability is specified in the data set resource definition. See FILE attributes for more information about the FILE resource definitions.
- Hold position in a VSAM data set for as short a time as possible. Table 1 shows which commands hold position and when the hold is released.

Table 30. Commands that hold position and when hold is released

Command	**Released by VSAM at**
READ.. UPDATE	REWRITE/DELETE/UNLOCK
WRITE.. MASSINSERT	UNLOCK
STARTBR	ENDBR

  Each request in progress against a VSAM data set requires at least one string. Requests that hold position tie up a string until a command is issued to release the hold position. Requests that do not hold position release the string as soon as that request is complete.

To minimize processor overhead when using VSAM data sets:

- Use the MASSINSERT option if you are adding many records in sequence. This improves performance by minimizing processor overheads and therefore improves the response times. For ESDSs and KSDSs, adding records with MASSINSERT causes CICS to use sequential VSAM processing. This changes the way VSAM places records within control intervals when a split is required, resulting in fewer splits and less unused space within the affected CIs.

- Use skip sequential processing if you are reading many records in sequence whose keys are relatively close together but not necessarily adjacent. (Skip sequential processing begins with a start browse (STARTBR command).) Each record is retrieved with an READNEXT command, but the key feedback area pointed to by RIDFLD is supplied with the key of the next requested record before the READNEXT command is issued.
- Use the GENERIC option on the DELETE command when deleting a group of records whose keys start with a common character string. CICS internally optimizes a generic DELETE.

BDAM data sets

BDAM data sets are less efficient than VSAM because CICS has to do some single-thread processing and issue some operating system waits to handle BDAM data set requests. Therefore, if possible, you should use a relative record VSAM data set or an entry-sequenced data set addressed by relative byte address (RBA) in place of a BDAM data set.

If you are using BDAM data sets in update mode, you should be aware that performance is affected dramatically by the means of data set integrity you choose.

If you specify exclusive control in file control table SERVREQ operands for a BDAM data set, CICS requests the operating system to prevent concurrent updates. However, this involves significant overhead.

Efficient browsing (in non-RLS mode)
A data set browse is often the source of the output in transactions that produce many output screens, which can monopolize system resources. You can allow other tasks to get control by issuing DELAY or SUSPEND commands.

A long browse can put a severe load on the system, locking out other transactions and increasing overall response time, in addition to the overhead needed for BMS, task control, and terminals.

This is because CICS control philosophy is based on the assumption that the terminal operator initiates a transaction that accesses a few data records, processes the information, and returns the results to the operator. This process involves numerous waits that enable CICS to do multitasking. However, CICS is not an interrupt-driven multitasking system, so tasks that involve small amounts of I/O relative to processing can monopolize the system regardless of priority. A browse of a data set with many records in a control interval is just such a transaction.

You can prevent this by issuing DELAY or SUSPEND commands periodically, so that other tasks can get control. If the browse produces paged output, you should consider breaking the transaction up in one of the ways suggested in “Page-building and routing operations” on page 440.

Terminal control

The CICS application programming interface contains two sets of commands for communicating with terminals, Terminal control commands and Basic Mapping Support (BMS).

Terminal control interface

Terminal control is the more basic of the two. It gives you flexibility and function, at the cost of more programming. In particular, if you code at the terminal control level, you need to build the device data stream in your application.

Terminal control commands apply to various devices, reducing the sensitivity of programs to the terminals they support and to the access methods controlling the terminals. In addition to the commands themselves, CICS provides the data translation, synchronization of input and output operations, and session control needed to read from or write to a terminal or logical unit. This helps insulate you from the APIs of the individual communications access methods, which are complex and different from one another.

BMS

BMS lets you communicate with a terminal at a much higher language level. It formats your data, and you do not need to know the details of the data stream. It is thus easier to code initially and easier to maintain, especially if your application has to support new types of terminal. However, BMS

pathlengths are longer (BMS itself uses terminal control), and BMS does not support all the terminal types that terminal control does. BMS is described in Basic mapping support.

BMS insulates you even more from the characteristics of particular devices and the mechanics of communication than does terminal control, but at the cost of some flexibility and function.

This section describes:

- "Terminal access method support" on page 257
- "Terminal control commands" on page 257
- "Using data transmission commands" on page 260
- "Device control commands" on page 261
- "Terminal device support" on page 262
- "Finding out about your terminal" on page 265
- "Using SNA" on page 268
- "Using sequential terminal support" on page 270
- "Using batch data interchange" on page 272

Terminal access method support

CICS Transaction Server for z/OS, Version 5 Release 6 supports terminals directly through interfaces to the following access methods, z/OS Communications Server, Basic Graphics Access Method (BGAM) for graphics terminals using GDDM and Sequential Access Method (SAM) for terminals simulated by sequential devices.

CICS supports operating system consoles as terminals too, but through operating system services rather than through an access method. The terminal control interface to a console is the same as to other terminals (though certain consoles might have certain restrictions), but BMS is not available. You can find a full list of the terminals supported by CICS in DFHTCT: CICS terminals list.

Terminal control commands

Terminal control commands are used for basic data transmission, for control functions, and for retrieving information about your terminal. Terminal control also provides special device group commands.

The commands described in this section apply only to the principal facility of the task issuing them, where that facility is one of the following:

- A device connected through SAM
- An LU Type 0, 1, 2, 3, or 4 connected through z/OS Communications Server.

Note: This section does not cover program-to-program communication, whether directed to the alternate or principal facility. This is covered in Summary of commands for APPC basic conversations.

Terminal control commands fall into four groups:

- Basic data transmission commands: RECEIVE, SEND, and CONVERSE
- Commands that send device controls, synchronize transmission, end a session, or perform similar control functions
- Commands to tell you about your terminal: ASSIGN and INQUIRE.
- Special device group commands: the batch data interchange (BDI) commands

Send/receive mode

The terminals and logical units all operate in "half-duplex, flip-flop" mode. This means, essentially, that at any given moment, one partner in a conversation is in send mode (allowed to send data or control commands) and the other is in receive mode (restricted to receiving).

This protocol is formally defined and enforced by the z/OS Communications Server for SNA. CICS observes the same conventions for terminals attached under other access methods, but both the hardware and the access methods work differently, so that not all operations are identical.

When a terminal is the principal facility of a task, its conversation partner is the task. When it is not associated with a task, its conversation partner is the terminal control component of CICS. Between tasks, under the Communications Server, the conversation is left in a neutral state where either partner can send first. Ordinarily the terminal goes first, sending the unsolicited input that initiates a task (see "How tasks are started" on page 73).

This transmission also reverses the send/receive roles; thereafter the terminal is in receive mode and CICS , represented by the task that was attached, is in send mode. The task starts and remains in send mode, no matter how many SENDs it executes, until it explicitly changes the direction of the conversation. One way in which you can put the task in receive mode is by specifying the INVITE option on a SEND command. After SEND with INVITE, the task is in receive mode and must issue a RECEIVE before sending to the terminal again. You can also put the task in receive mode by issuing a RECEIVE, without a preceding INVITE; INVITE optimizes transmissions.

Note that the first RECEIVE command in a task initiated by unsolicited input does not count in terms of send/receive mode, because the input message involved has long since transmitted (it started the task). This RECEIVE just makes the message accessible to the task, and sets the related EIB fields.

ATI tasks—those initiated automatically by CICS—also start out in send mode, just like tasks started by unsolicited input.

Note that if a task is executing normally and performing non-terminal operations when a Communications Server or network error occurs, the task is unaware of the error and continues processing until it attempts the next terminal control request. It is at this point that the task receives the TERMERR. If the task does not issue any further terminal control request, it will not receive the TERMERR or ABEND.

Contention for the terminal

CICS satisfies requests for automatic task initiation (ATI) as soon as the terminal required as principal facility is available.

But when a task ends at a terminal, and CICS has an ATI request for that terminal, there may be contention between CICS, which wants to initiate the ATI task, and the terminal user, who wants to initiate a certain task by unsolicited input. In this situation, CICS always sets itself up as contention *loser* . That is, if the terminal sends unsolicited input quickly enough after the end of the previous transaction, CICS creates a task to process it and delay fulfilling the ATI request. This is intentional—it gives the user priority in contention situations.

RETURN IMMEDIATE

S ometimes need to execute a sequence of particular tasks in succession at a terminal without allowing the user to intervene. CICS provides a way for you to do this, with the IMMEDIATE option on the RETURN command that ends the task.

With RETURN IMMEDIATE, CICS initiates a task to execute the transaction named in the TRANSID option immediately, before honoring any other waiting requests for tasks at that terminal and without accepting input from the terminal. The old task can even pass data to the new one. The new task accesses this data with a RECEIVE, as if the user had initiated the task with unsolicited input, but no input/output occurs. This RECEIVE, like the first one in a task initiated by unsolicited input, has no effect on send/receive status; it just makes the passed data available to the new task. If the terminal is using bracket protocol (explained in "Preventing interruptions with bracket protocol" on page 270), CICS does not end the bracket at the end of the first task, as it ordinarily does, but instead continues the bracket to include the following task. Consequently, the automatic opening of the keyboard at the end of bracket between tasks does not occur.

Speaking out of turn

It is typically clear to users when they are supposed to “talk” (key and transmit), and when they are supposed to “listen” (wait for output), because the application makes this clear.

On 3270 displays and many other terminals, the keyboard locks after the user has transmitted to reinforce this convention. It remains locked until the task unlocks it, which it typically does on a SEND before a RECEIVE, or on the last SEND in the task. This means that the user has to do something particular (press the keyboard reset key) in order to break protocol.

What happens if the user does this? For terminals connected under the z/OS Communications Server, violating this protocol causes the task to abend (code ATCV) unless read-ahead queuing is in force. Read-ahead queuing allows the logical unit and the task to send and receive at any time; CICS saves input messages in temporary storage until the task needs them. Inputs not read by task end are discarded. Read-ahead queuing is applied at the transaction level (it is specified in the RAQ option of the PROFILE under which the transaction runs). Read-ahead queuing applies only to LU type 4 devices, and was originally provided for compatibility reasons, to allow a transaction to support both BTAM-connected and z/OS Communications Server-connected terminals in the same way. As BTAM is no longer supported, read-ahead queuing should no longer be used.

Sequential terminals differ from others in send/receive rules. Because the input is a pre-prepared file, CICS provides input messages whenever the task requests them, and it is impossible to break protocol. If the input is improperly prepared, or is not what the task is programmed to handle, it is possible for the task to get out of synchronization with its inputs, to exhaust them prematurely, or to fail to read some of them.

Interrupting

The z/OS Communications Server provides a mechanism for a terminal in receive mode to tell its partner that it would like to send. This is the *signal* data flow in Communications Server, which is detected on the next SEND, RECEIVE or ISSUE DISCONNECT command from the task.

When a signal flow occurs, CICS raises the SIGNAL condition and sets EIBSIG in the EIB. CICS default action for the SIGNAL condition is to ignore it. For the signal to have any effect, the task must first detect the signal and then honor it by changing the direction of the conversation.

On a 3270 display terminal and some others, the ATTENTION key is the one that generates the interrupt. Not all terminals have this feature, however, and in z/OS Communications Server, the bind image must indicate support for it as well, or the z/OS Communications Server ignores the interrupts.

Terminal waits

When you use the WAIT option on a SEND command, CICS does not return control to your task until the output operation is complete.

When a task issues a SEND command without specifying WAIT, CICS can defer transmission of the output to optimize either its overall terminal handling or the transmissions for your task. When it does this, CICS saves the output message and makes your task dispatchable, so that it can continue executing. The ISSUE COPY and ISSUE ERASE commands, which also transmit output, work similarly without WAIT.

If you use the WAIT option, CICS does not return control to your task until the output operation is complete. This wait lengthens the elapsed time of your task, with attendant effects on response time and memory occupancy, but it ensures that your task knows whether there has been an error on the SEND before continuing. You can avoid some of this wait and still check the completion of the operation if you have processing to do after your SEND. You issue the SEND without WAIT, continue processing, and then issue a WAIT TERMINAL command at the point where you need to know the results of your SEND.

When you issue a RECEIVE command that requires transmission of input, your task always waits, because the transmission must occur before the RECEIVE can be completed. However, there are cases where a RECEIVE does not correspond to terminal input/output. The first RECEIVE in a task initiated by unsolicited terminal input is the most frequent example of this, but there are others, as explained in the next section.

Also, when you issue any command involving your terminal, CICS ensures that the previous command is complete (this includes any deferred transmissions), before processing the new one.

Using data transmission commands

Three commands transmit data to and from the terminal or logical unit that is the principal facility of your task.

The commands are:

RECEIVE
reads data from the terminal.

SEND
writes data to the terminal.

CONVERSE
writes data to the terminal, waits for input, and reads the input.

CONVERSE is essentially a combination of SEND and RECEIVE and is usually the equivalent of SEND followed by RECEIVE. In certain cases you must use CONVERSE instead of SEND and RECEIVE, for example, sending structured-field data to certain 3270 devices. In other cases you must use SEND and RECEIVE, because CONVERSE is not provided; these are noted in Table 33 on page 264.

The SEND, RECEIVE, and CONVERSE commands are fully described in CICS command summary . They are broken down by device group, because the options for different devices and access methods vary considerably. “Terminal device support” on page 262 tells you which device group to use for your particular device.

What you get on a RECEIVE

We use the terms *input message* and *transmission* to mean both what the terminal sent and what the application received. For the most common types of terminals, the application receives the data that was sent by the terminal. A 3270 display, for example, sends whatever was changed in its buffer as a single entity, and the task associated with the terminal normally gets the entire message in response to a single RECEIVE command.

However, input messages and physical transmissions are not always equivalent, and there are several factors that can affect the one-to-one relationship of either to RECEIVE commands.

Input chaining

Some SNA devices break up long input messages into multiple physical transmissions, a process called *chaining*.

CICS assembles the component transmissions into a single input message or present them individually, depending on how the terminal associated with the task has been defined. This affects how many RECEIVEs you need to read a chained input message. Details on inbound chaining are explained in “Chaining input data” on page 268.

Logical messages

Just as some devices break long inputs into multiple transmissions, others block short inputs and send them in a single transmission.

Here again, CICS provides an option about who deblocks, CICS or the receiving program. This choice also affects how much data you get on a single RECEIVE. (See “Handling logical records” on page 269 for more on this subject.)

NOTRUCATE option

An exception to the one-input-message-per-RECEIVE rule occurs when the length of the input data is greater than the program expects.

If this occurs and the RECEIVE command specifies NOTRUNCATE, CICS saves the excess data and uses it to satisfy subsequent RECEIVE commands from the program with no corresponding read to the terminal.

If you are using NOTRUNCATE, issue RECEIVE commnads until the field EIBCOMPL in the EIB is set on (that is, set to X'FF'). CICS turns on EIBCOMPL when no more of the input message is available.

Without NOTRUNCATE, CICS discards the excess data, turns on EIBCOMPL, and raises the LENGERR condition. It reports the true length of the data, before truncation, in the data area named in the LENGTH option, if you provide one.

Print key

If your CICS system has a PA key defined as a “print” key, another exception to the normal send/receive sequence can occur.

If the task issues a RECEIVE, and the user presses the “print” key in response, CICS intercepts this input, does the necessary processing to fulfill the request, and puts the terminal in receive mode again. The user must send another input to satisfy the original RECEIVE. (See CICS print key in “Printing display screens” on page 357 for more information about the “print” key.)

Device control commands

In addition to data transmission commands, the CICS API for terminals includes a series of commands that send instructions or control information, rather than data, to the terminal or to the access method controlling it.

These commands are listed in the following table, along with a brief description of their function. Not all these commands apply to all terminals, and for some, different forms apply to different terminals. See “Terminal device support” on page 262.

The terminal in the following table is always the principal facility of the task issuing the command, except where explicitly stated otherwise. It can be a logical unit of a type not ordinarily considered a terminal.

Table 31. Control commands for terminals and logical units

Command	Action
FREE	Releases the terminal from the task, so that the terminal can be used in another task before the current one ends.
ISSUE COPY	Copies the buffer contents of the terminal named in the TERMID option to the buffer of the terminal owned by the task. Both terminals must be 3270s.
ISSUE DISCONNECT	Schedules termination of the session between CICS and the terminal at the end of the task.
ISSUE EODS	Sends an end-of-data-set function management header (for 3650 interpreter logical units only).
ISSUE ERASEAUP	Erases all the unprotected fields of the terminal (for 3270 devices only).
ISSUE LOAD	Instructs the terminal to load the program named in the PROGRAM option (for 3650 interpreter logical units only).
ISSUE PASS	Schedules disconnection of the terminal from CICS and its transfer to the z/OS Communications Server application named in the LUNAME option, at the end of the issuing task.
ISSUE PRINT	Copies the terminal buffer to the first printer eligible for a print request (for 3270 displays only).
WAIT SIGNAL	Suspends the issuing task until its terminal sends a SIGNAL data flow command.
WAIT TERMINAL	Suspends the issuing task until the previous terminal operation has completed.

Terminal device support

Hardware and access method sensitivity is one of the major distinctions between using BMS and using terminal control commands to communicate with a terminal. BMS shields an application from hardware dependencies at the expense of some loss of function, whereas terminal control provides all the function.

The result of providing full function is that not all terminal control commands apply to all devices. Some commands require that you know the type of terminal, to determine the options that apply and the exception conditions that can occur. For some commands, you also need to know what access method is in use. The following tables show which commands apply to which terminal and access method combinations. If you need to support several types of terminals, you can find out which type your task has as its principal facility using the commands described in "Finding out about your terminal" on page 265.

To use the tables, look up the terminal type that your program must support in the first column of Table 32 on page 262. Use the value in the second column to find the corresponding command group in the first column of Table 33 on page 264. The second column of this table shows the access method, and the third shows the commands you can use. The commands themselves are described in full in CICS command summary . Where there is more than one version of a command, the table shows which one to use. This information appears in parentheses after the command.

Table 32. Devices supported by CICS

Device	**Use commands for**
2260, 2265	2260
3101 (supported as TWX 33/35)	3767
3230	3767
3270 displays, 3270 printers	LU type 2/3
3270 displays, 3270 printers (non-SNA)	3270 logical
3270 displays, 3270 printers	3270 display
SCS printers	SCS
3600 Pipeline mode	3600 pipeline
3601	3600-3601
3614	3600-3614
3630, attached as 3600 (3631, 3632, 3633, 3643, 3604)	Use 3600 entry
3641, 3644, 3646, 3647 (attached as 3767)	3767
3643 (attached as LU type 2)	LU type 2/3
3642, 3645 (attached as SCS printer)	SCS
3650 interpreter LU	3650 interpreter
3650 host conversational LU (3270)	3650-3270
3650 host conversational LU (3653)	3650-3653
3650 host command LU (3680, 3684)	3650-3680
3650 interpreter LU	3650 interpreter
3650 host conversational LU (3270)	3650-3270
3650 host conversational LU (3653)	3650-3653
3650 host command LU (3680, 3684)	3650-3680
3730	3790 full function or inquiry

Table 32. Devices supported by CICS (continued)

Device	Use commands for
3767 interactive LU	3767
3770 Interactive LU	3767
3770 Full function LU	3790 full function or inquiry
3770 Batch LU (3771, 3773, 3774)	3770
3790 Full function or inquiry	3790 full function or inquiry
3790 3270 display LU	3790 3270-display
3790 SCS printer	3790 SCS
3790 3270 printer	3790 3270-printer
4700 (supported as 3600)	Use 3600 entry
5280 attached as 3270	Use 3270 entry
5520, supported as 3790 full-function LU	3790 full function or inquiry
5550 (supported as 3270)	Use 3270 entry
5937 (supported as 3270)	Use 3270 entry
6670	LU type 4
8130, 8140 under DPCX (supported as 3790)	3790 full function or inquiry
8100 DPPX/BASE using Host Presentation Services or Host Transaction Facility (attached as 3790)	3790 full function or inquiry
8100 DPPX/DSC, DPCX/DSC, including 8775 attach (supported as 3270)	LU type 2/3
8775	LU type 2/3
8815	APPC
Displaywriter supported as 3270	Use 3270 entry
Displaywriter supported as APPC	APPC
INTLU (interactive LU)	3767
PC, PS/2 , attached as 3270	Use 3270 entry
Scanmaster	APPC
Series/1 supported as 3650 pipeline	3600 pipeline
Series/1 supported as 3790 full-function LU	3790 full function or inquiry
System/32 (5320), supported as 3770	Use 3770 entry
System/34 (5340), supported as 3770	Use 3770 entry
System/34 (5340)	System/3
System/36 (supported as System/34)	Use System/34 entry
System/38 (5381), attached as 3770	Use 3770 entry
System/38 (5381), attached as APPC	APPC
TWX 33/35 NTO	3767

Table 32. Devices supported by CICS (continued)	
Device	**Use commands for**
WTTY NTO	3767

Table 33. Terminal control commands by device type

Device group name	Access methods	Commands applicable
2260	non- z/OS Communications Server	RECEIVE (2260), SEND (2260), CONVERSE (2260), ISSUE DISCONNECT (default), ISSUE RESET
3270 display	non- z/OS Communications Server	RECEIVE (3270 display), SEND (3270 display), CONVERSE (3270 display), ISSUE COPY (3270 display), ISSUE DISCONNECT (default), ISSUE ERASEAUP, ISSUE PRINT, ISSUE RESET
LU type 2/3 (3270 SNA)	z/OS Communications Server	RECEIVE (LU type 2/3), SEND (LU type 2/3), CONVERSE (LU type 2/3), ISSUE COPY (3270 logical), ISSUE DISCONNECT (default), ISSUE ERASEAUP, ISSUE PASS, ISSUE PRINT
3270 logical (3270 non-SNA)	z/OS Communications Server	RECEIVE (3270 logical), SEND (3270 logical), CONVERSE (3270 logical), ISSUE COPY (3270 logical), ISSUE DISCONNECT (default), ISSUE ERASEAUP, ISSUE PASS, ISSUE PRINT
SCS	z/OS Communications Server	SEND (SCS), CONVERSE (SCS), ISSUE DISCONNECT (default), ISSUE PASS
3600 pipeline	z/OS Communications Server	RECEIVE (3600 pipeline), SEND (3600 pipeline), ISSUE DISCONNECT (default), ISSUE PASS
3600-3601	z/OS Communications Server	RECEIVE (3600-3601), SEND (3600-3601), CONVERSE (3600-3601), ISSUE DISCONNECT (default), ISSUE PASS, WAIT SIGNAL
3600-3614	z/OS Communications Server	RECEIVE (3600-3614), SEND (3600-3614), CONVERSE (3600-3614), ISSUE DISCONNECT (default), ISSUE PASS
3650 interpreter	z/OS Communications Server	RECEIVE (3650), SEND (3650 interpreter), CONVERSE (3650 interpreter), ISSUE DISCONNECT (default), ISSUE EODS, ISSUE LOAD, ISSUE PASS
3650-3270	z/OS Communications Server	RECEIVE (3650), SEND (3650-3270), CONVERSE (3650-3270), ISSUE DISCONNECT (default), ISSUE ERASEAUP, ISSUE PASS, ISSUE PRINT
3650-3653	z/OS Communications Server	RECEIVE (3650), SEND (3650-3653), CONVERSE (3650-3653), ISSUE DISCONNECT (default), ISSUE PASS
3650-3680	z/OS Communications Server	RECEIVE (3650), RECEIVE (3790 full function or inquiry), SEND (3650-3680), SEND (3790 full function or inquiry), CONVERSE(3650-3680), ISSUE DISCONNECT (default), ISSUE PASS

Table 33. Terminal control commands by device type (continued)		
Device group name	**Access methods**	**Commands applicable**
3767	z/OS Communications Server	RECEIVE (3767), SEND (3767), CONVERSE (3767), ISSUE DISCONNECT (default), ISSUE PASS, WAIT SIGNAL
3770	z/OS Communications Server	RECEIVE (3770), SEND (3770), CONVERSE (3770), ISSUE DISCONNECT (default), ISSUE PASS, WAIT SIGNAL
3790 full function or inquiry	z/OS Communications Server	RECEIVE (3790 full function or inquiry), SEND (3790 full function or inquiry), CONVERSE (3790 full function or inquiry), ISSUE DISCONNECT (default), ISSUE PASS, WAIT SIGNAL
3790 3270-display	z/OS Communications Server	RECEIVE (3790 3270-display), SEND (3790 3270-display), CONVERSE (3790 3270-display), ISSUE DISCONNECT (default), ISSUE ERASEAUP, ISSUE PASS, ISSUE PRINT
3790 3270-printer	z/OS Communications Server	SEND (3790 3270-printer), ISSUE DISCONNECT (default), ISSUE ERASEAUP, ISSUE PASS
3790 SCS	z/OS Communications Server	SEND (3790 SCS), ISSUE DISCONNECT (default), ISSUE PASS
LU type 4	z/OS Communications Server	RECEIVE (LU type 4), SEND (LU type 4), CONVERSE (LU type 4), ISSUE DISCONNECT (default), ISSUE PASS, WAIT SIGNAL
Outboard controllers (batch data interchange)	z/OS Communications Server	ISSUE ABORT, ISSUE ADD, ISSUE END, ISSUE ERASE, ISSUE NOTE, ISSUE QUERY, ISSUE RECEIVE, ISSUE REPLACE, ISSUE SEND, ISSUE WAIT
All others	z/OS Communications Server	RECEIVE (Communications Server for SNA default), SEND (Communications Server for SNA default), CONVERSE (Communications Server for SNA default), ISSUE PASS
All others	non- z/OS Communications Server	RECEIVE (non- z/OS Communications Server default), SEND (non- z/OS Communications Server default), CONVERSE (non- z/OS Communications Server default)

Finding out about your terminal

Some applications must support more than one type of terminal, and sometimes the types are sufficiently different that they require separate code. If you are writing such a program, and you need to determine what sort of terminal it is currently communicating with, you can use the ASSIGN command to find out.

ASSIGN returns a variety of information about the executing task, including a number of fields that describe its principal facility. Table 34 on page 266 lists the ones that relate directly to terminal control operations. There are other ASSIGN options that relate to BMS and to other aspects of the task. You can find details on all ASSIGN options in CICS command summary . The "terminal" cited in column 2 of the table is always the principal facility of the task.

Table 34. ASSIGN command options for terminals	
ASSIGN option	**Information returned**
ALTSCRNHT ALTSCRNWD	The alternate height and width of the terminal screen (from its terminal definition); see also SCRNHT and SCRNWD
APLKYBD	Whether terminal has an APL keyboard
APLTEXT	Whether terminal has the APL text feature
BTRANS	Whether terminal has background transparency capability
COLOR	Whether terminal has extended color capability
DEFSCRNHT DEFSCRNWD	The default height and width of the terminal screen (from its terminal definition); see also SCRNHT and SCRNWD
DELIMITER	The data-link control character for the terminal (for 3600 terminals only)
DESTID DESTIDLENGTH	The identifier of the outboard destination and its length (for BDI operations only)
DSSCS	Whether the terminal is an SCS data stream device
DS3270	Whether the terminal is a 3270 data stream device
EXTDS	Whether the terminal supports “query structured field” orders
EWASUPP	Whether the terminal supports “erase write alternate” orders (i.e. has alternate screen size capability)
FACILITY	The 4-character identifier of the terminal
FCI	The type of principal facility associated with the task (terminal, queue, and so on)
GCHARS GCODES	The graphic character set global identifier and the code page global identifier associated with the terminal
HILIGHT	Whether the terminal has extended highlight capability
KATAKANA	Whether the terminal supports Katakana
LANGINUSE	The 3-character mnemonic
MSRCONTROL	Whether the terminal supports magnetic slot reader control
NATLANGINUSE	The national language in use for the current task
NETNAME	The 8-character identifier of the terminal in the SNA network
NUMTAB	Number of tabs required to position the print element in the correct passbook area (for 2980s only)
OPID OPCLASS	Operator identifier code and operator class of user signed on at terminal
OUTLINE	Whether the terminal has field outlining capability
PARTNS	Whether the terminal supports screen partitions
PS	Whether the terminal has programmed symbols capability

Table 34. ASSIGN command options for terminals (continued)	
ASSIGN option	**Information returned**
SCRNHT SCRNWD	Height and width of the terminal screen for the current task
SIGDATA	SIGNAL data received from the terminal
SOSI	Whether the terminal has mixed EBCDIC/double-byte character set capability
STATIONID TELLERID	Station and teller identifier of the terminal (for 2980s only)
TERMCODE	Type and model number of the terminal
TERMPRIORITY	Terminal priority value
TEXTKYBD	Whether the terminal has the TEXTKYBD feature
TEXTPRINT	Whether the terminal has the TEXTPRINT feature
UNATTEND	Whether the terminal is unattended
USERID USERNAME USERPRIORITY	The 8-character identifier, 20-character name and priority of the user signed on at the terminal
VALIDATION	Whether the terminal has validation capability

You can also use the **INQUIRE TERMINAL** command to find out about your own terminal or any other terminal. **INQUIRE TERMINAL** returns information from the terminal definition, whereas ASSIGN describes the use of that terminal in the current task. For many options, however, particularly the hardware characteristics, the information returned is the same.

EIB feedback on terminal control operations

CICS reports the results of processing terminal control commands, including those generated by BMS, in the EIB.

Because of the complexity of terminal operations, many EIB fields are specific to terminal commands. Those that apply to the principal facility are listed in Table 35 on page 268 . (Other fields relate only to LU type 6.1, APPC and MRO operations; see CICS command summary for programming information about these.)

EIB fields are posted when CICS returns control to your task, and they always describe the most recent command to which they apply. This means that if you are conducting program-to-program communication over an alternate facility and using your principal facility, you need to check the EIB before results from one overlay those of the other.

It also means that when a task is initiated by unsolicited input from the terminal, or by a RETURN IMMEDIATE in the previous task at the same terminal, the EIB fields that describe the input are *not* set at task start. You must issue a RECEIVE to gain access to the input and post the EIB fields.

Note: If you are interested only in the EIB values and not the data itself, omit both the INTO and SET options from your RECEIVE.

Here are the fields that apply to the principal facility:

Table 35. EIB fields that apply to terminal control commands	
Field	**Contents**
EIBAID	The attention identifier (AID) from the last input operation (3270s only, see the AID in "Input from a 3270 terminal" on page 284).
EIBATT	Whether the input contains attach header data (an attach FMH)
EIBCOMPL	Whether the RECEIVE command just issued used all the input data, or more RECEIVEs are required (see "Chaining output data" on page 268).
EIBCPOSN	Cursor position at time of last input operation (3270s only)
EIBEOC	Whether an end-of-chain indicator appeared in the input from the last RECEIVE
EIBFMH	Whether user data just received or retrieved contains an FMH
EIBFREE	Whether the facility just used has been freed
EIBRCODE EIBRESP EIBRESP2	CICS response code values from the previously issued command **Note:** For output commands in which transmission can be deferred, these values reflect only the initial CICS processing of the command, not the eventual transmission (see "Terminal waits" on page 259).
EIBSIG	Whether the terminal has sent a SIGNAL
EIBTRMID	(CICS) identifier of the terminal

Using SNA

Communication with logical units is governed by the conventions (protocols) which vary with the type of logical unit. This section describes the options provided by CICS to enable applications to conform to and make best use of these protocols.

Chaining input data

Some SNA devices segment long input messages for transmission. Each individual segment is called a *request unit* (RU), and the entire logical message is called a *chain* . CICS provides an option in the terminal definition, BUILDCHAIN, that governs who assembles the chain.

If the BUILDCHAIN value for the terminal is YES, CICS assembles the chain and presents the entire message to the program in response to a single RECEIVE command. This choice ensures that the whole chain is complete and available before it is presented to the application.

If BUILDCHAIN=NO, the application assembles the chain. CICS provides one RU for each RECEIVE. The application can tell when it has received the last RU in the chain, because CICS raises the EOC (end-of-chain) condition at that time. CICS raises this condition even when there is only one RU in the chain, or when it assembles the chain, or when the input is from a terminal that does not support inbound chaining, like a 3270 display. An EOC condition is not considered an error; the CICS default action when it occurs is to ignore the condition.

EOC may occur simultaneously with either the EODS (end-of-data-set) or INBFMH (inbound-FMH) conditions, or both. Either condition takes precedence over EOC in determining where control goes if both it and EOC are the subject of active HANDLE CONDITION commands.

Chaining output data

z/OS Communications Server for SNA supports the chaining of outbound and inbound terminal data. If the length of an output message exceeds the outbound RU size, and the terminal supports outbound chaining, CICS breaks the message into RU-size segments, and transmits them separately.

Your application can take advantage of the fact that chaining is permitted by passing a single output message to CICS bit by bit across several SEND commands. To do this, you specify the CNOTCOMPL ("chain not complete") option on each SEND except the one that completes the message. (Your

message segments do not need be any particular length; CICS assembles and transmits as many RUs as are required.) The PROFILE definition under which your transaction is running must specify CHAINCONTROL=YES in order for you to do this.

Note: Options that apply to a complete logical message (that is, the whole chain) must appear only on the first SEND command for a chain. These include FMH, LAST, and, for the 3601, LDC.

Handling logical records

Some devices block input messages and send multiple inputs in a single transmission. CICS allows you to specify whether CICS or the application should deblock the input.

The choice is expressed in the LOGREC option of the PROFILE under which the current transaction is executing.

With LOGREC (NO), CICS provides the entire input message in response to a RECEIVE (assuming the input is not chained or BUILDCHAIN=YES). The user is responsible for deblocking the input. If BUILDCHAIN=NO, a RECEIVE retrieves one RU of the chain at a time. In general, logical records do not span RUs, so that a single RU contains one or more complete logical records. The exception is LU type 4 devices, where a logical record may start in one RU and continue in another; for this reason, BUILDCHAIN=YES is recommended if you do your own deblocking for these devices.

If the PROFILE specifies LOGREC (YES), CICS provides one logical record in response to each RECEIVE command (whether or not CICS is assembling input chains).

If an RU contains more than one logical record, the records are separated by new line (NL) characters, X'15' , interrecord separators (IRS characters), X'1E' , or transparent (TRN) characters, X'35' . If NL characters are used, they are not removed when the data is passed to the program and appear at the end of the logical record. If IRS characters are used, however, they are removed. If the delimiter is a transparent character, the logical record can contain any characters, including NL and IRS, which are considered normal data in transparent mode. The terminating TRN is removed, however. CICS limits logical records separated by TRNs to 256 characters.

Response protocol

Under z/OS Communications Server for SNA, CICS allows the use of either definite response or exception response protocol for outbound data.

Under exception response, a terminal acknowledges a SEND only if an error occurred. If your task is using exception response, CICS does not wait for the last SEND in the task (which can be the only SEND) to complete before terminating your task. Consequently, if an error does occur, it can not be possible to report it to your task. When this happens, the error is reported to a CICS-supplied task created for the purpose.

Definite response requires that the terminal acknowledge every SEND, and CICS does not terminate your task until it gets a response on the last SEND. Using definite response protocol has some performance disadvantages, but it can be necessary in some applications.

The MSGINTEG option of the PROFILE under which a task is running determines which response mode is used. However, if you select MSGINTEG (NO) (exception response), you can still ask for definite response on any particular SEND by using the DEFRESP option. In this way, you can use definite response selectively, paying the performance penalty only when necessary. For transactions that must verify the delivery of data before continuing, the DEFRESP option must be used on the last SEND.

Using function management headers

SNA architecture defines a particular type of header field that accompanies some messages, called a function management header (FMH).

The header conveys information about the message and how it should be handled. For some logical units, use of an FMH is mandatory, for others it is optional, and in some cases FMHs cannot be used at all. In particular, FMHs do not apply to LU type 2 and LU type 3 terminals, which are the most common 3270 devices.

Inbound FMH
When an FMH is present in an input message, CICS examines the PROFILE definition under which the transaction is executing to decide whether to remove the FMH or pass it on to the application program that issued the RECEIVE.

The PROFILE can specify that no FMHs are to be passed, that only the FMH indicating the end of the data set should be passed, or that all FMHs are to be passed. There is also an option that causes the FMH to be passed to the batch data interchange program.

If an FMH is present, it occupies the initial bytes of the input message; its length varies by device type. CICS sets the EIBFMH field in the EIB on (X'FF') to tell you that one is present, and it also raises the INBFMH condition, which you can detect through a HANDLE CONDITION command or by testing the RESP value.

Outbound FMH
On output, the FMH can be built by the application program or by CICS.

If your program supplies the FMH, you place it at the front of your output data and specify the FMH option on your SEND command. If CICS is to build the FMH, you reserve the first three bytes of the message for CICS to complete and omit the FMH option. CICS builds an FMH only for devices that require one; you must supply it for devices for which it is optional.

Preventing interruptions with bracket protocol
Brackets are an SNA protocol for ensuring that a conversation between two LUs is not interrupted by a request from a third LU. CICS uses bracket protocol to prevent interruption of the conversation between a CICS task and its principal facility for the duration of the task.

If the task has an alternate facility, bracket protocol is used there also, for the same reason. The logical unit begins the bracket if it sends unsolicited input to initiate the task, and CICS begins the bracket if it initiates the task automatically. CICS ends the bracket at task end, unless the IMMEDIATE option appears on the final RETURN command. RETURN IMMEDIATE lets you initiate another task at your principal facility without allowing it to enter input. CICS does this by *not* ending the bracket between the ending task and its successor when brackets are in use.

CICS requires the use of brackets for many devices under the z/OS Communications Server for SNA. For others, the use of brackets is determined by the value of the BRACKET option in the terminal definition. Because bracket protocol is a feature of SNA, if you specify BRACKET(YES) for non-SNA devices, CICS will neither follow, nor enforce, strict bracket protocol.

In general, bracket protocol is transparent to an application program, but it is still possible to optimize flows related to bracket protocol using the LAST option on the SEND command. If you know that a particular SEND is the last command for the terminal in a task, you can improve performance by adding the LAST option. LAST allows Communications Server to send the “end-of-bracket” indicator with the data and saves a separate transmission to send it at task end. If you are sending the last output in a program-built chain (using CNOTCOMPL), LAST must be specified on the first SEND for the chain in order to be effective.

If your task has significant work to do or may experience a significant delay after its last SEND, you may want to issue a FREE command. FREE releases the terminal for use in another task.

Using sequential terminal support

One of the many types of terminal that CICS supports is not really a terminal at all, but a pair of sequential devices or files simulating a terminal.

One of the pair represents the input side of the terminal, and might be a card reader, a spool file or a SAM file on tape or DASD. The other represents the output, and might be a printer, a punch, spool, or SAM file. Many device-type combinations are allowed, and either of the pair can be missing; that is, you can have an input-only or output-only sequential terminal.

You read from and write to the devices or files that constitute a sequential terminal with terminal control commands, specifically RECEIVE, SEND, and CONVERSE. (BMS supports sequential terminals too; see “Special options for non-3270 terminals” on page 382 .)

The original purpose of sequential terminal support was to permit application developers to test online code before they had access to real terminals. This requirement rarely occurs any more, but sequential terminals are still useful for:

Printing
See “Programming for non-CICS printers” on page 356 . Sequential terminals are useful for output that is sometimes directed to a low-speed CICS printer, for which BMS or terminal control commands are required, and sometimes directed to a high-speed system printer (spool or transient data commands). If you define the high-speed printer as a sequential terminal, you can use terminal control or BMS commands, and you can use the same code for both types of printers. (If there are differences in the device data streams, you need to use BMS for complete transparency.)

Regression testing
Tests run from sequential terminals leave a permanent record of both input and output. This encourages systematic and verifiable initial testing. Also, it allows you to repeat tests after modifications, to ensure that a given set of inputs produces the same set of outputs after the change as before.

Initialization
Some installations use a sequential terminal to execute one or more initialization transactions, in preference to program list table programs. Transactions initiated from a sequential terminal begin execution as soon as the terminal is in service, and they continue as quickly as CICS can process them until the input is exhausted. Hence the inputs from a sequential terminal can be processed immediately after startup, if the sequential terminal is initially in service, at some later time (when it is put in service) or even as part of a controlled shutdown.

Coding considerations for sequential terminals

The input data submitted from a sequential terminal must be in the form in which it would come from a telecommunication device.

For example, the first record usually starts with a transaction code, to tell CICS what transaction to execute. The transaction code must start in the first position of the input, just as it must on a real terminal. Note that this limits the ability to test applications that require input in complex formats. For example, there is no provision for expressing a formatted 3270 input stream as a sequential file, because of all the complex control sequences. However, you can use an unformatted 3270 data stream (or any other similar stream) for input, and you can still use BMS to format your output.

When you build the input file, you place an end-of-data indicator (EODI) character after each of your input records. The EODI character is defined in the system initialization table; the default value is a backslash ('\', X'E0'), but your installation may have defined some other value.

When processing the input stream, CICS observes EODI characters only. CICS does not analyze the record structure of the input file or device, which means that each input can span records in the input file. However, you must start each input on a new physical record to ensure each input is correctly processed.

The length of an input record (the number of characters between EODIs) should not exceed the size of the input buffer (the INAREAL value in the LINE component of the sequential terminal definition). If it does, the transaction that attempts to RECEIVE the long record abends, and CICS positions the input file after the next EODI before resuming input processing.

An end-of-file marker in the input also acts as an EODI indicator. Any RECEIVE command issued after end-of-file is detected also causes an abend.

Print formatting
If the definition of a sequential terminal indicates that the output half is a line printer, you can write multiple lines of output with a single SEND.

For this type of device, CICS breaks your output message into lines after each new line character (X'15') or after the number of characters defined as the line length, whichever occurs first. Line length is defined by the LPLEN value in the terminal definition. Each SEND begins a new line.

GOODNIGHT convention
CICS continues to initiate transactions from a sequential terminal until it (or the transactions themselves) have exhausted all the input or until the terminal goes out of service. To prevent CICS from attempting to read beyond the end of the input file (which causes a transaction abend), the last input can be a CESF GOODNIGHT transaction, which signs the terminal off and puts it out of service.

Alternatively, the last transaction executed can put the terminal out of service after its final output. You cannot normally enter further input from a sequential terminal once CICS has processed its original input, without putting it out of service.

Using batch data interchange

The CICS batch data interchange program provides for communication between an application program and a named data set (or destination) that is part of a batch data interchange logical unit in an outboard controller, or with a selected medium on a batch logical unit or an LU type 4 logical unit. This medium indicates the required device such as a printer or console.

The term *outboard controller* is a generalized reference to a programmable subsystem, such as the IBM 3770 Data Communication System, the IBM 3790 Data Communication System, or the IBM 8100 System running DPCX, which uses SNA protocols. Details of SNA protocols and the data sets that can be used are given in IBM 3767/3770/6670 Guide. Figure 86 on page 272 gives an overview of batch data interchange.

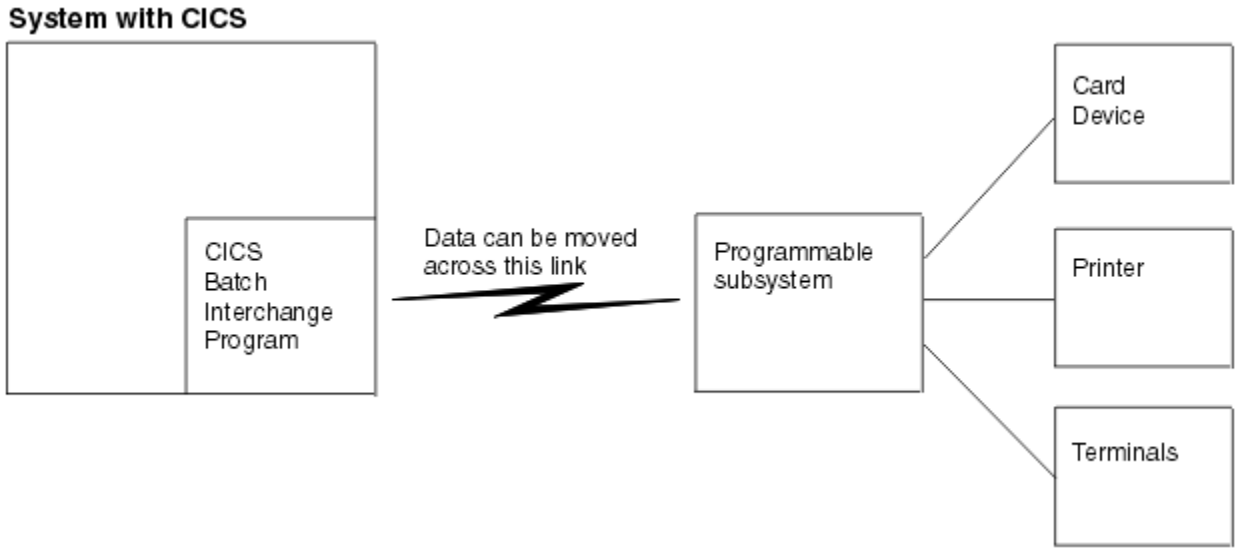


Figure 86. CICS batch data interchange

The following batch data interchange commands are provided:

ISSUE QUERY
Initiate transfer of a data set to the CICS application program.

ISSUE RECEIVE
Read a record from a data set or read data from an input medium.

ISSUE SEND
Transmit data to a named data set or to a selected medium.

ISSUE ADD
Add a record to a data set.

ISSUE REPLACE
Update (replace) a record in a data set.

ISSUE ERASE
Delete a record from a data set.

ISSUE END
Terminate processing of a data set.

ISSUE ABORT
Terminate processing of a data set abnormally.

ISSUE NOTE
Request the next record number in a data set.

ISSUE WAIT
Wait for an operation to be completed.

Where the controller is an LU type 4 logical unit, only the ISSUE ABORT, ISSUE END, ISSUE RECEIVE, ISSUE SEND, and ISSUE WAIT commands can be used.

Where the data set is a DPCX/DXAM data set, only the ISSUE ADD, ISSUE ERASE, and ISSUE REPLACE commands can be used.

Refer to "Dealing with exception conditions" on page 189 for information about how to deal with any exception conditions that occur during execution of a batch data interchange command.

Destination selection and identification

All batch data interchange commands except ISSUE RECEIVE include options that specify the destination. This is either a named data set in a batch data interchange logical unit, or a selected medium in a batch logical unit or LU type 4 logical unit.

When you select a destination by named data set, the DESTID and DESTIDLENG options must always be specified, to supply the data set name and its length (up to a maximum of eight characters). For destinations having diskettes, the VOLUME and VOLUMELENG options may be specified, to supply a volume name and its length (up to a maximum of six characters); the volume name identifies the diskette that contains the data set to be used in the operation. If the VOLUME option is not specified for a multidiskette destination, all diskettes are searched until the required data set is found.

As an alternative to naming a data set as the destination, various media can be specified by means of the CONSOLE, PRINT, CARD, or WPMEDIA1–4 options. These media can be specified only in an ISSUE ABORT, ISSUE END, ISSUE SEND, or ISSUE WAIT command.

Definite response (DEFRESP option)

CICS uses terminal control commands to carry out the functions specified in batch data interchange commands. For those commands that cause terminal control output requests to be made, the DEFRESP option can be specified. This option has the same effect as the DEFRESP option of the SEND terminal control command; that is, to request a definite response from the outboard controller, irrespective of the specification of message integrity for the CICS task (by the system programmer). The DEFRESP option can be specified for the ISSUE ADD, ISSUE ERASE, ISSUE REPLACE, and ISSUE SEND commands.

Waiting for function completion (NOWAIT option)

For those batch data interchange commands that cause terminal control output requests to be made, the NOWAIT option can be specified. This option has the effect of allowing CICS task processing to continue; unless the NOWAIT option is specified, task activity is suspended until the batch data interchange command is completed. The NOWAIT option can be specified only on the ISSUE ADD, ISSUE ERASE, ISSUE REPLACE, and ISSUE SEND commands.

After a batch data interchange command with the NOWAIT option has been issued, task activity can be suspended, by the ISSUE WAIT command, at a suitable point in the program to wait for the command to be completed.

Terminal control: design for performance

This list shows you what you need to consider for terminal control when designing for performance.

- **Minimize the length of the data stream sent to the terminal.**

  Good screen design and effective use of 3270 hardware features can significantly affect the number of bytes transmitted on a teleprocessing link. It is important to keep the number of bytes as small as possible because, in most cases, this is the slowest part of the path a transaction takes. The efficiency of the data stream therefore affects both response time and line usage.

- **Use only one physical SEND command per screen.**

  It is typically more efficient to create a screen with a single call to BMS, than to build the screen with a series of SEND MAP ACCUM commands. It is important to send the screen in a single physical output to the terminal. It is ***very*** inefficient to build a screen in parts and send each part with a separate command, because of the additional processor overhead of using several commands and the additional line and access method overhead.

- **Use the CONVERSE command.**

  Use the CONVERSE command rather than the SEND and RECEIVE commands (or a SEND, WAIT, RECEIVE command sequence if your program is conversational). They are functionally equivalent, but the CONVERSE command crosses the CICS services interface only once, which saves processor time.

- **Limit the use of message integrity options.**

  Like specifying the WAIT option on the final SEND command of a transaction, the MSGINTEG option of CEDA requires CICS to keep the transaction running until the last message has been delivered successfully. The PROTECT option of the PROFILE definition implies message integrity and causes the system to log all input and output messages, which adds to I/O and processor overhead.

- **Avoid using the DEFRESP option on SEND commands.**

  Avoid using the DEFRESP option on SEND commands, unless the transaction must verify successful delivery of the output message. It delays termination of the transaction in the same way as MSGINTEG.

- **Avoid using unnecessary transactions.**

  Avoid situations that can cause users to enter an incorrect transaction or to use the CLEAR key unnecessarily, thus adding to terminal input, task control processing, terminal output, and overhead. Good screen design and standardized PF and PA key assignments should minimize this.

- **Send unformatted data without maps.**

  If your output to a terminal is entirely or even mostly unformatted, you can send it using terminal control commands rather than BMS commands (that is, using a BMS SEND command without the MAP or TEXT options).

The 3270 family of terminals

The 3270 is a family of display and printer terminals, with supporting control units, that share common characteristics and use the same encoded data format to communicate between terminal and host processor. This data format is known as the *3270 data stream*.

The 3270 is a complex device with many features and capabilities. Only basic operations are covered here and the emphasis is on the way CICS supports the 3270. For a comprehensive discussion of 3270 facilities, programming and data stream format, see IBM 3270 Data Stream Programmers Reference. IBM 3270 Data Stream Programmers Reference also contains important information. It is primarily intended for programmers using terminal control, but contains information that might be helpful for BMS programmers as well. BMS support for a few special features is discussed in “Support for special hardware” on page 433.

Although this discussion is focused on display terminals, most of the material applies equally to 3270 printers. A 3270 printer accepts the same data stream as a 3270 display and delivers the screen image in hardcopy form. Most of the differences relate to input, which is (mostly) lacking on printers.

However, additional formatting facilities are available for use with printers, and there are special considerations in getting your printed output to the intended printer. For more information see "Printing and spool files" on page 345.

The 3270 buffer

Communication with a 3270 device occurs through its **character buffer** , which is a hardware storage mechanism similar to the memory in a processor. Output to the 3270 is sent to the buffer. The buffer, in turn, drives the display of a display terminal and the print mechanism of a printer terminal.

Conversely, keyboard input reaches the host through the buffer, as explained in "Input from a 3270 terminal" on page 284.

Each position on the screen corresponds to one in the buffer, and the contents of that buffer position determine what is displayed on the screen. When the screen is formatted in fields, the first position of each field is used to store certain display characteristics of the field and is not available to display data (it appears blank). In the original models of the 3270, this byte was sufficient to store all of the display characteristics. In later models, which have more types of display characteristics, the additional information is kept in an area of buffer storage not associated with a fixed position on the screen. See "Display characteristics" on page 276 for more about display characteristics.

The output datastream

To create a 3270 display, you send a stream of data that consists of a write command (1 byte), a write control character or WCC (1 byte) and display data (variable number of bytes).

The WCC and display data are not always present; the write command determines whether a WCC follows and whether data can or must be present.

When you use BMS, CICS builds the entire data stream for you. The WCC is assembled from options in the SEND command, and the write command is selected from other SEND options and information in the PROFILE of the running transaction. Display data is built from map or text data that you provide, which BMS translates into 3270 format for you.

When you use terminal control commands, such as SEND, CICS still supplies the write command, built from the same information. However, you provide the WCC and you must express the display data in 3270 format.

3270 write commands

There are five 3270 commands that send data or instructions to a terminal:

- Write
- Erase/write
- Erase/write alternate
- Erase all unprotected fields
- Write structured fields

The 3270 **write** command sends the data that follows it to the 3270 buffer, from which the screen (or printer) is driven. **Erase/write** and **erase/write alternate** also send the data, but they erase the buffer first (that is, they set it entirely to null values). They also determine the buffer size (the number of rows and columns on the screen), if the terminal has a feature called *alternate screen size*.

Terminals with this feature have two sizes, default size and alternate size. The **erase/write** command causes the default size to be used in subsequent operations (until the next **erase/write** or **erase/write alternate** command). The **erase/write alternate** command selects the alternate size.

CICS uses the **write** command to send data unless you include the ERASE option on your **SEND** command:

- If you specify ERASE , CICS uses the PROFILE definition associated with the running transaction to determine whether to use **erase/write** or **erase/write alternate**.
- If you specify ERASE DEFAULT , CICS uses the **erase/write** command and sets the screen to default size.
- If you specify ERASE ALTERNATE , CICS uses the **erase/write alternate** and sets the screen to alternate size).

The **erase unprotected to address** command causes a scan of the buffer for unprotected fields; these fields are defined more precisely in "3270 field attributes" on page 277. Any such fields that are found are set to nulls. This selective erasing is useful in data entry operations, as explained in the **SEND CONTROL** command in "Merging the symbolic and physical maps" on page 390. No WCC or data follows this command; you send only the command.

The **write structured fields** command causes the data that follows to be interpreted as 3270 structured fields. Structured fields are required for some of the advanced function features of the 3270. They are not covered here, but you can write them with terminal control **SEND** commands containing the STRFIELD option. For more information, see IBM 3270 Data Stream Programmers Reference.

Write control character

The byte that follows a 3270 **write** , **erase/write** , or **erase/write alternate** command is the write control character or WCC. The WCC tells the 3270 whether to:

- Sound the audible alarm
- Unlock the keyboard
- Turn off the modified data tags
- Begin printing (if terminal is a printer)
- Reset structured fields
- Reset inbound reply mode

In BMS, CICS creates the WCC from the ALARM , FREEKB , FRSET , and PRINT options on your **SEND MAP** command. If you use terminal control commands, you can specify your WCC explicitly, using the CTLCHAR option. If you do not, CICS generates one that unlocks the keyboard and turns off the modified data tags (these tags are explained in "3270 field attributes" on page 277).

3270 display fields

Display data consists of a combination of characters to be displayed and instructions to the device on how and where to display them.

Under ordinary circumstances, this data consists of a series of field definitions, although it is possible to write the screen without defining fields, as explained in "Unformatted mode" on page 287.

After a write command that erases, you need to define every field on the screen. Thereafter, you can use a plain write command and send only the fields you want to change.

To define a field, you need to tell the 3270:

- How to display it
- What its contents are
- Where it goes on the screen (that is, its starting position in the buffer)

Display characteristics

Each field on the screen has a set of display characteristics, called attributes. Attributes tell the 3270 how to display a field, and you need to understand what the possibilities are whether you are using BMS or terminal control commands.

Attributes fall into two categories:

Field attributes
These include:

- Protection (whether the operator can modify the field or not)
- Modification (whether the operator *did* modify the field)
- Display intensity

All 3270s support field attributes; "3270 field attributes" on page 277 explains your choices for them.

Field attributes are stored in the first character position of a field. This byte takes up a position on the screen and not only stores the field attributes, but marks the beginning of the field. The field continues up to the next attributes byte (that is, to the beginning of the next field). If the next field does not start on the same line, the current one wraps from the end of the current line to the beginning of the next line until another field is encountered. A field that has not ended by the last line returns to the first.

Extended field attributes
(Typically shortened to **extended attributes**). These are not present on all models of the 3270. Consequently, you need to be aware of which ones are available when you design your graphical user interface. Extended attributes include special forms of highlighting and outlining, the ability to use multiple symbol sets and provision for double-byte character sets. Table 36 on page 278 lists the seven extended attributes and the values they can take.

3270 field attributes

The field attributes byte holds the protection, modification, and display intensity attributes of a field. Your choices for each of these attributes are described here using the terms that BMS uses in defining formats. If you use terminal control commands, you need to set the corresponding bits in the attributes byte to reflect the value you choose.

See IBM 3270 Data Stream Programmers Reference for the bit assignments. See also "Attribute value definitions: DFHBMSCA" on page 389 for help from CICS in this area.

Protection

There are four choices for the protection attribute, using up 2 bit positions in the attributes byte. They are:

Unprotected
The operator can enter any data character into an unprotected field.

Numeric-only
The effect of this designation depends on the keyboard type of the terminal. On a data entry keyboard, a numeric shift occurs, so that the operator can key numbers without shifting. On keyboards equipped with the "numeric lock" special feature, the keyboard locks if the operator uses any key except one of the digits 0 through 9, a period (decimal point), a dash (minus sign) or the DUP key. This prevents the operator from keying alphabetic data into the field, although the receiving program must still inspect the entry to ensure that it is a number of the form it expects. Without the numeric lock feature, numeric-only allows any data into the field.

Protected
The operator cannot key into a protected field. Attempting to do so locks the keyboard.

Autoskip
The operator cannot key into an autoskip field either, but the cursor behaves differently. (The cursor indicates the location of the next keystroke; for more information about this, see "Input from a 3270 terminal" on page 284 .) Whenever the cursor is being advanced to a new field (either because the previous field filled or because a field advance key was used), the cursor skips over any autoskip fields in its path and goes to the first field that is either unprotected or numeric-only.

Modification

The second item of information in the field attributes byte occupies only a single bit, called the **modified data tag** or **MDT** . The MDT indicates whether the field has been modified or not. The hardware turns on this bit automatically whenever the operator changes the field contents. The MDT bit is important because, for the read command that CICS normally uses, it determines whether the field is included in the

inbound data or not. If the bit is on (that is, the field was changed), the 3270 sends the field; if not, the field is not sent.

You can also turn on the MDT by program, when you send a field to the screen. Using this feature ensures that a field is returned on a read, even if the operator cannot or does not change it. The FRSET option on BMS SEND commands allows you to turn off the tags for all the fields on the screen by program; you cannot turn off individual tags by program. If you are using terminal control commands, you turn on a bit in the WCC to turn off an individual tag.

Intensity

The third characteristic stored in the attributes byte is the display intensity of the field. There are three mutually exclusive choices:

Normal intensity
The field is displayed at normal brightness for the device.

Bright
The field is displayed at higher than normal intensity, so that it appears highlighted.

Nondisplay
The field is not displayed at all. The field can contain data in the buffer, and the operator can key into it (provided it is not protected or autoskip), but the data is not visible on the screen.

2 bits are used for display intensity, which allows one more value to be expressed than the three listed previously. For terminals that have either of the associated special hardware features, these same 2 bits are used to determine whether a field is light-pen detectable or cursor selectable. Because there are only 2 bits, not all combinations of intensity and selectability are possible. The compromise is that bright fields are always detectable, nondisplay fields are never detectable, and normal intensity fields can be either. “Cursor and pen-detectable fields” on page 436 contains more information about these features.

Base color

Some terminals support **base color** without, or in addition to, the **extended colors** included in the extended attributes. There is a mode switch on the front of such a terminal, allowing the operator to select base or default color. Default color shows characters in green unless field attributes specify bright intensity, in which case they are white. In base color mode, the protection and intensity bits are used in combination to select among four colors: normally white, red, blue, and green; the protection bits retain their protection functions as well as determining color. (If you use extended color, rather than base color, for 3270 terminals, note that you cannot specify "white" as a color. You need to specify "neutral", which is displayed as white on a terminal.)

Extended attributes
In addition to the field attributes previously described, some 3270 terminals have extended attributes as well. The following table lists the types of extended attributes in the first column and the possible values for each type in the second column.

Table 36. 3270 extended attributes

Attribute type	**Values**
Extended color	Blue, red, pink, green, turquoise, yellow, neutral
Extended highlighting	Flashing, reverse video, underscoring
Field outlining	Lines over, under, left and right, in any combination
Background transparency	Background transparent, background opaque
Field validation	Field must be entered; field must be filled; field triggers input

Table 36. 3270 extended attributes (continued)	
Attribute type	**Values**
Programmed symbol sets	Number identifying the symbol set **Note:** The control unit associated with a terminal contains a default symbol set and can store up to five additional ones. To use one of these others, you need to load the symbol set into the controller before use. You can use a terminal control SEND command to do this.
SO/SI creation	Shift characters indicating double-byte characters can be present; shift characters are not present

IBM 3270 Data Stream Programmers Reference contains details about extended attributes and explains how default values are determined. You can use ASSIGN and INQUIRE commands to determine which extended attributes your particular terminal has. These commands are described in "Finding out about your terminal" on page 265.

Some models of the 3270 also allow you to assign extended attribute values to individual characters within a field that are different from the value for the field as a whole. Generally, you need to use terminal control commands to do this, because BMS does not make explicit provision for character attributes. However, you can insert the control sequences for character attributes in text output under BMS, as explained in "Text lines" on page 416. "The set attribute order" on page 281 describes the format of such a sequence.

Orders in the data stream

Writing to a 3270 using terminal control commands you need to format the outbound data to express the attributes, position, and contents of a field.

About this task

If you are using BMS, all this is done for you, and you can move on to "Input from a 3270 terminal" on page 284.

When you define a field in the 3270 data stream, you begin with a **start field (SF)** or a **start field extended (SFE)** order. **Orders** are instructions to the 3270. They tell it how to load its buffer. They are 1 byte long and typically are followed by data in a format specific to the order.

The start field order

The SF order is supported on all models and lets you specify the field attributes and the display contents of a field, but not extended attributes.

To define a field with SF, you insert a sequence in the data stream as in Figure 87 on page 279.

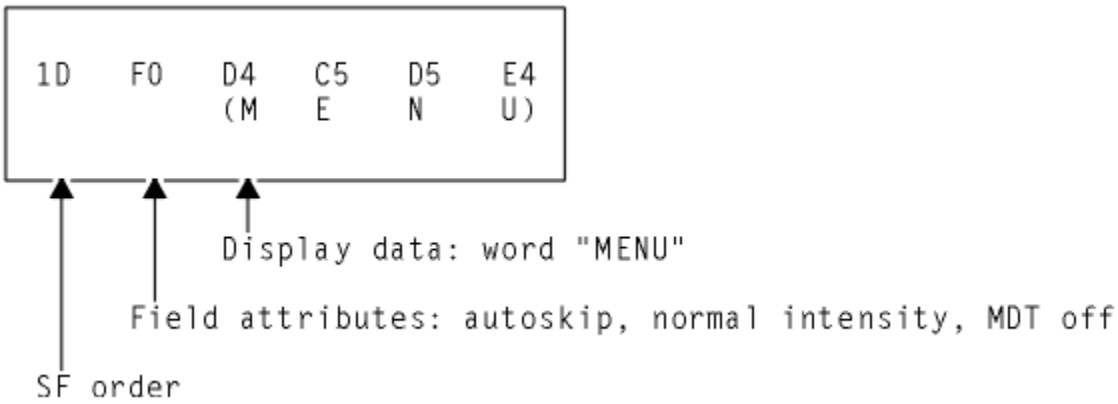


Figure 87. Field definition using SF order

If you need to specify extended attributes, and your terminal supports them, you use the start field extended order instead. SFE requires a different format, because of the more complex attribute information. Extended attributes are expressed as byte pairs. The first byte is a code indicating which type of attribute is being defined, and the second byte is the value for that attribute. The field attributes are treated collectively as an additional attribute type and also expressed as a byte pair. Immediately after the SFE order, you give a 1 byte count of the attribute pairs, then the attribute pairs, and finally the display data. The whole sequence is shown in Figure 88 on page 280.

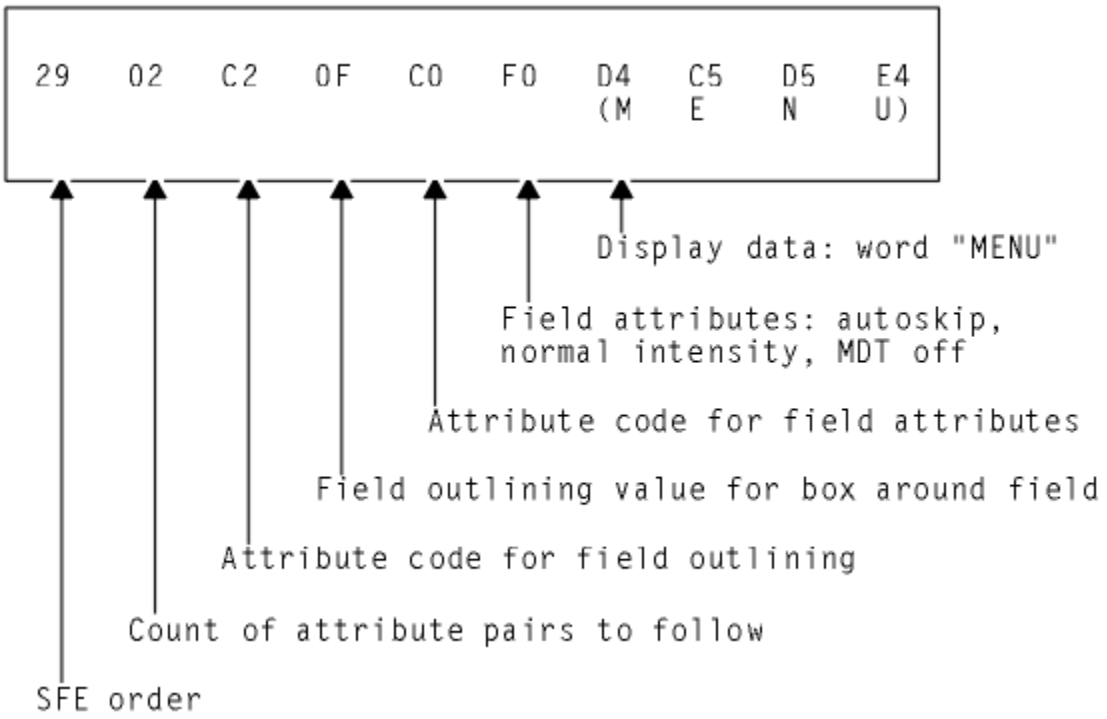


Figure 88. Field definition using SFE order

The modify field order
When a field is on the screen, you can change it with a command almost identical in format to SFE, called modify field (MF).

The only differences from SFE are:

- The field must exist.
- The command code is X'2C' instead of X'29'.
- You send only the attributes you want to change from their current values, and you send display data only if you want to change it.
- A null value sets an attribute back to its default for your particular terminal (you accomplish the same thing in an SFE order by omitting the attribute).

For example, to change the “menu” field of earlier examples back to the default color for the terminal and underscore it, you would need the sequence in Figure 89 on page 281.

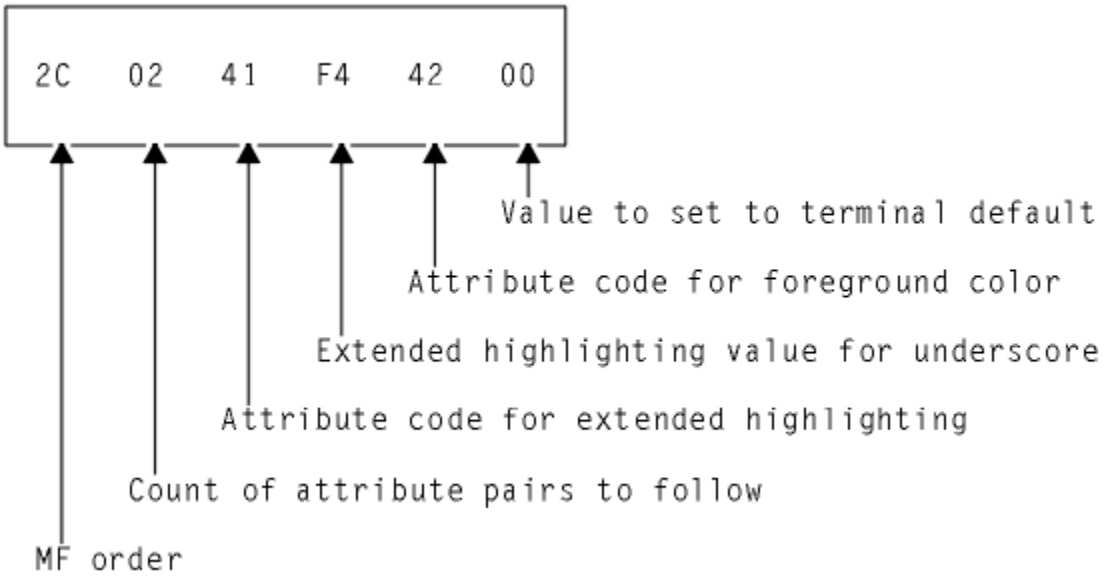


Figure 89. Changing field attributes within an MF order

The set buffer address order
The SF and SFE orders place the field they define at the current position in the buffer, and MF modifies the field at this position. Unless the field follows the last character sent (that is, begins in the current buffer position), you need to precede these orders with a set buffer address (SBA) order to indicate where you want to place or change a field.

To do this, you send an SBA order followed by a 2 byte address, as in Figure 90 on page 281.

Figure 90. SBA sequence

The address in the figure is a “12 bit” address for position 112 (X'70'), which is row 2, column 33 on an 80-column screen. Counting starts in the first row and column (the zero position) and proceeds along the rows. There are two other addressing schemes used: “14 bit” and “ 16 bit”. Buffer positions are numbered sequentially in all of them, but in 12 bit and 14 bit addressing, not all the bits in the address are used, so that they do not appear sequential. (The X'70' (B'1110000') in the figure appears as B'110000' in the low-order 6 bits of the rightmost byte of the address and B'000001' in the low-order 6 bits of the left byte.) IBM 3270 Data Stream Programmers Reference explains how to form addresses.

After an SF, SFE, or MF order, the current buffer address points to the first position in the buffer you did not fill—right after your data, if any, or after the field attributes byte if none.

The set attribute order
To set the attributes of a single character position, you use a set attribute (SA) order for each attribute you want to specify.

For example, to make a character flash, you need the sequence in Figure 91 on page 282.

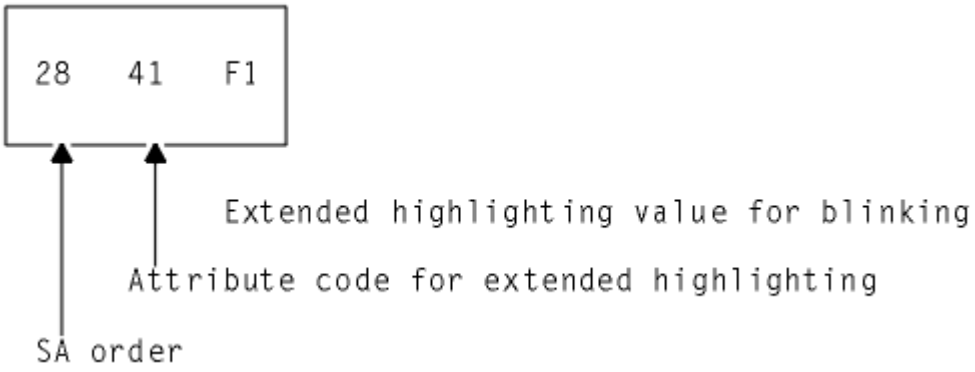


Figure 91. SA sequence to make a character flash

The attributes you specify with SA orders are assigned to the current buffer position, in the same way that field definitions are placed at the current buffer position, so you generally need to precede your SAs with SBA sequences.

Outbound data stream example

The outbound data stream example shows the data stream required to paint a particular 3270 screen, to reinforce the explanation of how the data stream is built.

Figure 92 on page 282 shows an example screen that is part of an application that tracks cars used by the employees at a work site, and is used to record a new car. The only inputs are the employee identification number, the license plate (tag) number, and, if the car is from out-of-state, the licensing state.

```
Car Record
                      Employee No: ______ Tag No: ________ State: __
```

Figure 92. Example of a data-entry screen

Note: This is an unrealistically simple screen, designed to keep the explanation manageably short. It does not conform to accepted standards of screen design, and you should not use it as a model.

There are eight fields on this screen:

1. Screen title, “Car Record”, on line 1, column 26
2. Label field, “Employee No:” (line 3, column 1), indicating what the operator is to enter into the next field
3. An input field for the employee number (line 3, column 14), 6 positions long
4. Label field, “Tag. No:”, at line 3, column 21
5. An input field (tag number) at line 3, column 31, 8 positions long
6. Label field, “State:”, at line 3, column 40
7. An input field (state), at line 3, column 49, 2 positions long
8. A field to mark the end of the previous (state) input field, at line 3, column 52

Table 37 on page 282 shows the outbound data stream:

Table 37. 3270 output data stream

Bytes	Contents	Notes
1	X'F5'	The 3270 command that starts the data stream, in this case erase/write.
2	X'C2'	WCC; this value unlocks the keyboard, but does not sound the alarm or reset the MDTs.

Table 37. 3270 output data stream (continued)		
Bytes	**Contents**	**Notes**
3	X'11'	SBA order to position first field at ...
4-5	X'40D6'	Address of line 1, column 23 on 24 by 80 screen, using 12 bit addressing.
6	X'1D'	SF order to begin first field definition.
7	X'F8'	Field attributes byte; this combination indicates a field which is autoskip and bright, with the MDT initially off.
8-17	'Car record'	Display contents of the field.
18-20	X'11C260'	SBA sequence to reset the current buffer position to line 3, column 1 for second field.
21	X'1D'	SF order for second field.
22	X'F0'	Field attributes byte: autoskip, normal intensity, MDT off.
23-34	'Employee No:'	Display contents of field.
35	X'29'	SFE order to start fourth field. SFE is required, instead of SF, because you need to specify extended attributes. This field starts immediately after the previous one ended, so you do not have to precede it with an SBA sequence.
36	X'02'	Count of attribute *types* that are specified (two here: field outlining and field attributes).
37	X'41'	Code indicating attribute type of extended highlighting.
38	X'F4'	Extended highlighting value indicating underscoring.
39	X'C0'	Code indicating attribute type of field attributes.
40	X'50'	Field attributes value for numeric-only, normal intensity, MDT off. Any initial data for this field would appear next, but there is none.
41	X'13'	Insert cursor (IC) order, which tells the 3270 to place the cursor at the current buffer position. We want it at the start of the first field which the operator has to enter data, which is the current buffer position.
42-44	X'11C2F4'	SBA sequence to position to line 3, column 21, to leave the six positions required for an employee number. The beginning of the “Tag No” label field marks the end of the employee number input field, so that the user is aware immediately if they try to key too long a number.
45	X'1D'	SF order to start field.
46	X'F0'	Field attributes byte: autoskip, normal intensity, MDT off.
47-55	' Tag No:'	Display data. We attach two leading blanks to the label for more space between the fields. (We could have used a separate field, but this is easier for only a few characters.)
56	X'29'	SFE (the next field is another input field, where we want field outlining, so we use SFE again).
57	X'02'	Count of attribute types.

Table 37. 3270 output data stream (continued)		
Bytes	**Contents**	**Notes**
58-59	X'41F4'	Code for extended highlighting with value of underscoring.
60-61	X'C040'	Code for field attributes and attributes of unprotected, normal intensity, MDT off.
62-64	X'11C3C7'	SBA sequence to reposition to line 3, column 40, leaving eight positions for the tag.
65	X'1D'	SF to start field.
66	X'F0'	Field attributes byte: autoskip, normal intensity, MDT off.
67-74	' State:'	Field data (two leading blanks again for spacing).
75-80	X'290241F4C040'	SFE order and attribute specifications for state input field (attributes are identical to those for tag input field).
81-82	X'0000'	The (initial) contents of the state field. We could have omitted this value as we did for other input fields, but we would need an SBA sequence to move the current buffer position to the end of the field, and this is shorter.
83	X'1D'	SF. The last field indicates the end of the previous one, so that the user does not attempt to key more than two characters for the state code. It has no initial data, just an attributes byte. This field is sometimes called a “stopper” field.
84	X'F0'	Field attributes byte: autoskip, normal intensity, MDT off.

Note: If you use terminal control commands and build your own data stream, the data you provide in the FROM parameter of your SEND command starts at byte 3 in the table; CICS supplies the write command and the WCC from options on your SEND command.

Input from a 3270 terminal

Keyboard input reaches the host through the buffer. There are many different keyboard arrangements available for 3270 terminals, but in any arrangement, a key falls into one of three categories, data key, keyboard control key, or attention key.

Data keys

The data keys include all the familiar letters, numbers, punctuation marks, and special characters. Pressing a data key changes the content of the buffer (and therefore the screen) at the point indicated by the cursor. The cursor is a visible pointer to the position on the screen (that is, in the buffer) where the next data keystroke is to be stored. As the operator keys data, the cursor advances to the next position on the screen, skipping over fields defined with the autoskip attribute on the screens that have been formatted.

Keyboard control keys

Keyboard control keys move the cursor to a new position, erase fields or individual buffer positions, cause characters to be inserted, or otherwise change where or how the keyboard modifies the buffer.

Attention keys

The keys in the previous groups, Data, and Keyboard control keys, cause no interaction with the host; they are handled entirely by the device and its control unit. An attention key, signals that the buffer is ready for transmission to the host. If the host has issued a read to the terminal, the typical situation in CICS , transmission occurs at this time.

There are five types of attention key:

- ENTER
- F (function) key
- CLEAR
- PA (program attention) key
- CNCL (cancel key, present only on some keyboard models)

In addition to pressing an attention key, there are other operator actions that cause transmission:

- Using an identification card reader
- Using a magnetic slot reader or hand scanner
- Selecting an attention field with a light pen or the cursor select key
- Moving the cursor out of a trigger field

Trigger field capability is provided with extended attributes on some terminal models, but all the other actions listed previously require special hardware, and in most cases the screen (buffer) must be set up appropriately beforehand. We talk about these features in "Support for special hardware" on page 433 . For this section, we concentrate on standard features.

The AID

The 3270 identifies the key that causes transmission by an encoded value in the first byte of the inbound data stream. This value is called the **attention identifier** or **AID**.

Ordinarily, the key that the terminal operator chooses to transmit data is dictated by the application designer. The designer assigns specific meanings to the various attention keys, and the user must know these meanings in order to use the application. (Often, there are only a few such keys in use: ENTER for normal inputs, one function key to exit from control of the application, another to cancel a partially completed transaction sequence, for example. Where there are a number of choices, you can list the key definitions on the screen, so that the user does not have to memorize them.)

There is an important distinction between two groups of attention keys, which the application designer must keep in mind. The ENTER and function keys transmit data from the buffer when the host issues a "read modified" command, the command normally used by CICS. CLEAR, CNCL and the PA keys do not, although you do get the AID (that is, the identity of the key that was used). These are called the **short read** keys. They are useful for conveying simple requests, such as "cancel", but not for those that require accompanying data. In practice, many designers use function keys even for the nondata requests, and discard any accompanying data.

Note: The CLEAR key has the additional effect of setting the entire buffer to nulls, so that there is literally no data to send. CLEAR also sets the screen size to the default value, if the terminal has the alternate screen size feature, and it puts the screen into unformatted mode, as explained in "Unformatted mode" on page 287.

Reading from a 3270 terminal

There are two basic read commands for the 3270, read buffer and read modified.

For either command, the inbound data stream starts with a 3 byte **read header** consisting of:

- Attention identifier (AID), 1 byte
- Cursor address, 2 bytes

As noted in the previous section, the AID indicates which action or attention key causes transmission. The cursor address indicates where the cursor was at the time of transmission. CICS stores this information in the EIB, at EIBAID and EIBCPOSN, on the completion of any RECEIVE command.

The read buffer command brings in the entire buffer following the read header, and the receiving program is responsible for extracting the information it wants based on position. It is intended primarily for diagnostic and other special purposes, and CICS uses it in executing a RECEIVE command only if the

BUFFER option is specified. CICS never uses read buffer to read unsolicited terminal input, so the BUFFER option cannot be used on the first RECEIVE of a transaction initiated in this way.

With read modified, the command that CICS normally uses, much less data is transmitted. For the short read keys (CLEAR, CNCL and PAs), only the read header comes in. For other attention keys (ENTER and PFs), the fields on the screen that were changed (those with the MDT on, to be precise) follow the read header. We describe the format in the next section. When transmission occurs because of a trigger field, light pen detect or cursor select, the amount and format of the information is slightly different; these special formats are described in “Support for special hardware” on page 433 . Input from a **program attention** key on an SCS printer is also an exception; see “SCS input” on page 351 for a description of that data stream.

Inbound field format

If you are using terminal control commands, you must be aware of the format in which the 3270 transmits data.

If you are using BMS, you can skip to “Unformatted mode” on page 287 , because BMS translates the input for you.

Each modified field comes in as follows:

- SBA order
- 2 byte address of the first *data* position of field
- SF order
- Field contents

Only the non-null characters in the field are transmitted; nulls are skipped, wherever they appear. Thus if an entry does not fill the field, and the field was initially nulls, only the characters keyed are transmitted, reducing the length of the inbound data. Nulls (X'00') are not the same as blanks (X'40'), even though they are indistinguishable on the screen. Blanks get transmitted, and hence you normally initialize fields to nulls rather than to blanks, to minimize transmission.

A 3270 read command can specify that the terminal must return the attribute values along with the field contents, but CICS does not use this option. Consequently, the buffer address is the location of the first byte of field data, not the preceding attributes byte (as it is in the corresponding outbound data stream).

Note: Special features of the 3270 for input, such as the cursor select key, trigger fields, magnetic slot readers, and so on, produce different input formats. See “Field selection features” on page 435 for details.

Input data stream example

This example shows the input data stream created using the data entry screen from the previous outbound data stream example.

To illustrate an inbound data stream, we assume that an operator using the screen shown in Figure 92 on page 282 did the following:

- Put “123456” in the employee identifier field
- Put “ABC987” in the tag number
- Pressed ENTER, omittinglast the state field

Here is the resulting inbound data stream:

Table 38. 3270 input data stream

Bytes	Contents	Notes
1	X'7D'	AID, in this case the ENTER key.
2-3	X'C3C5'	Cursor address: line 3, column 38, where the operator previously used it after the last data keystroke.

Table 38. 3270 input data stream (continued)		
Bytes	**Contents**	**Notes**
4	X'11'	SBA, indicating that a buffer address follows.
5-6	X'C26E'	Address of line 3, column 15, which is the starting position of the field to follow.
7-12	'123456'	Input, the employee number entered by the operator.
13-15	X'11C3D1'	SBA sequence indicating a buffer address of line 3, column 32.
16	X'1D'	SF, indicating another input field follows.
17-22	'ABC987'	Input field: plate number. Notice that only six characters came in from a field that was eight long, because the remaining null positions were not completed by an operator.

The third input field (the state code) does not appear in the input data stream. This is because its MDT did not get turned on; it was set off initially, and the operator did not turn it on by keying into the field. Note also that no SF is required at byte 7 because CICS normally issues a Read Modified All.

Unformatted mode

Even though the high function of the 3270 revolves around its field structure, it is possible to use the 3270 without fields, in what is called **unformatted mode** . In this mode, there are no fields defined, and the entire screen (buffer) behaves as a single string of data, inbound and outbound, much like earlier, simpler terminals.

When you write in unformatted mode, you define no fields in your data, although you can include SBA orders to direct the data to a particular positions on the screen. Data that precedes any SBA order is written starting at the current position of the cursor. (If you use an erase or write command, the cursor is automatically set to zero, at the upper left corner of the screen.)

When you read an unformatted screen, the first 3 bytes are the read header (the AID and the cursor address), just as when you read a formatted screen. The remaining bytes are the contents of the entire buffer, starting at position zero. There are no SBA or SF orders present, because there are no fields. If the read command was read modified, the nulls are suppressed, and therefore it is not always possible to determine exactly where on the screen the input data was located.

You cannot use a BMS RECEIVE MAP command to read an unformatted screen. BMS raises the MAPFAIL condition on detecting unformatted input, as explained in “MAPFAIL and other exception conditions” on page 405 . You can read unformatted data only with a terminal control RECEIVE command in CICS.

Note: The CLEAR key puts the screen into unformatted mode, because its sets the buffer to nulls, erasing all the attributes bytes that demarcate fields.

Interval control

The CICS interval control services provide functions that are related to time.

Java and C++

The application programming interface described here is the EXEC CICS API, which is not used in Java programs. For information about Java programs using the JCICS classes to access Interval Control services, see Java development using JCICS and the JCICS Javadoc documentation. For information about C++ programs using the CICS C++ classes, see Using the CICS foundation classes.

Using interval control commands, you can:

- Start a task at a specified time or after a specified interval, and pass data to it (START command).
- Retrieve data passed on a START command (RETRIEVE command).
- Delay the processing of a task (DELAY command).

- Request notification when a specified time has expired (POST command).
- Wait for an event to occur (WAIT EVENT command).
- Cancel the effect of previous interval control commands (CANCEL command).
- Request the current date and time of day (ASKTIME command).
- Select the format of date and time (FORMATTIME command). Options are available that help you to handle dates in the 21st century.

Note: Do not use EXEC CICS START TRANSID() TERMID(EIBTRMID) to start a remote transaction. Use EXEC CICS RETURN TRANSID() IMMEDIATE instead. START, used in this way, ties up communications resources unnecessarily and can lead to performance degradation across the connected regions.

If you use WAIT EVENT, START, RETRIEVE with the WAIT option, CANCEL, DELAY, or POST commands, you could create inter-transaction affinities that adversely affect your ability to perform dynamic transaction routing.

Storage for the timer-event control area on WAIT EVENT must reside in shared storage if you have specified ISOLATE(YES).

If CICS is executing with or without transaction isolation, CICS checks that the timer-event control area is not in read-only storage.

To help you identify potential problems with programs that issue these commands, you can use the CICS Interdependency Analyzer. See CICS Interdependency Analyzer for z/OS Overview for more information about this utility and “Affinity” on page 156 for more information about transaction affinity.

Request identifiers

As a means of identifying the request and any data associated with it, a unique request identifier is assigned by CICS to each DELAY , POST, and START command. You can specify your own request identifier with the REQID option. If you do not, CICS assigns (for POST and START commands only) a unique request identifier and places it in field EIBREQID in the EXEC interface block (EIB). You should specify a request identifier if you want the request to be canceled at some later time by a CANCEL command.

Expiration times

The time at which a time-controlled function is to be started is known as the **expiration time** . You can specify expiration times absolutely, as a time of day (using the TIME option), or as an interval that is to elapse before the function is to be performed (using the INTERVAL option).

For DELAY commands, you can use the FOR and UNTIL options; and for POST and START commands, you can use the AFTER and AT options.

Note: The C and C++ languages do not provide the support for the packed decimal types used by the TIME and INTERVAL options.

You use an **interval** to tell CICS when to start a transaction in a specified number of hours, minutes, and seconds from the current time. A nonzero INTERVAL value always indicates a time in the future—the current time plus the interval you specify. The hours can be 0–99, but the minutes and seconds must not be greater than 59. For example, to start a task in 40 hours and 10 minutes, you would code:

```
EXEC CICS START INTERVAL(401000)
```

You can use an **absolute time** to tell CICS to start a transaction at a specific time, again using hhmmss. For example, to start a transaction at 3:30 in the afternoon, you would code:

```
EXEC CICS START TIME(153000)
```

An absolute time is always relative to the midnight before the current time and can therefore be earlier than the current time. TIME can be in the future or the past relative to the time at which the command is executed. CICS uses the following rules:

- If you specify a task to start at any time within the previous six hours, it starts immediately. This happens regardless of whether the previous six hours includes a midnight. For example:

  ```
  EXEC CICS START TIME(123000)
  ```

  This command, issued at 05:00 or 07:00 on Monday, expires at 12:30 on the same day.

  ```
  EXEC CICS START TIME(020000)
  ```

  This command, issued at 05:00 or 07:00 on Monday expires immediately because the specified time is within the preceding six hours.

  ```
  EXEC CICS START TIME(003000)
  ```

  This command, issued at 05:00 on Monday, expires immediately because the specified time is within the preceding six hours. However, if it is issued at 07:00 on Monday, it expires at 00:30 on Tuesday, because the specified time is not within the preceding six hours.

  ```
  EXEC CICS START TIME(230000)
  ```

  This command, issued at 02:00 on Monday, expires immediately because the specified time is within the preceding six hours.
- If you specify a time with an hours component that is greater than 23, you are specifying a time on a day following the current one. For example, a time of 250000 means 1 a.m. on the day following the current one, and 490000 means 1 a.m. on the day after that.

If you do not specify an expiration time or interval option on DELAY, POST, or START commands, CICS responds using the default of INTERVAL(0), which means immediately.

Because each end of an intersystem link can be in a different time zone, you should use the INTERVAL form of expiration time when the transaction to be started is in a remote system.

If the system fails, the times associated with unexpired START commands are remembered across the restart.

Note:

1. On a lightly used system, the interval time specified can be exceeded by as much as a quarter of a second.
2. If your expiration time falls within a possible CICS shutdown, you should consider whether your task should test the status of CICS before attempting to run. You can do this using the CICSSTATUS option of the **INQUIRE SYSTEM** . During a normal shutdown, your task could run at the same time as the PLT programs with consequences known only to you.

Task control

The CICS task control facility provides functions that synchronize task activity, or that control the use of resources.

Java and C++

The application programming interface described here is the EXEC CICS API, which is not used in Java programs. For information about Java programs using the JCICS classes to access Task Control services CICS, see Java development using JCICS and the JCICS Javadoc documentation. For information about C++ programs using the CICS C++ classes, see Using the CICS foundation classes.

CICS assigns priorities based on the value set by the CICS system programmer. Control of the processor is given to the highest-priority task that is ready to be processed, and is returned to the operating system when no further work can be done by CICS or by your application programs.

You can:

- Suspend a task (SUSPEND command) to enable tasks of higher priority to proceed. This can prevent processor-intensive tasks from monopolizing the processor. When other eligible tasks have proceeded and terminated or suspended processing, control is returned to the issuing task; that is, the task remains dispatchable.
- Schedule the use of a resource by a task (ENQ and DEQ commands). This is sometimes useful in protecting a resource from concurrent use by more than one task; that is, by making that resource serially reusable. Each task that is to use the resource issues an enqueue command (ENQ). The first task to do so has the use of the resource immediately but, if a HANDLE CONDITION ENQBUSY command has not been issued, subsequent ENQ commands for the resource, issued by other tasks, result in those tasks being suspended until the resource is available.

  If the NOSUSPEND option is coded on an ENQ command, control is always returned to the next instruction in the program. By inspecting the contents of the EIBRESP field, you can see whether the ENQ command was successful or not.

  Each task using a resource should issue a dequeue command (DEQ) when it has finished with it. However, when using the enqueue/dequeue mechanism, there is no way to guarantee that two or more tasks issuing ENQ and DEQ commands issue these commands in a given sequence relative to each other. For a way to control the sequence of access, see Controlling sequence of access to resources.
- Change the priority assigned to a task (CHANGE TASK PRIORITY command).
- Wait for events that post MVS format ECBs when they complete.

  Two commands are available, WAITCICS and WAIT EXTERNAL. These commands cause the issuing task to be suspended until one of the ECBs has been posted; that is, until one of the events has occurred. The task can wait on one or more ECBs. If it waits on more than one, it is dispatchable as soon as one of the ECBs is posted. You must ensure that each ECB is cleared (set to binary zeros) no later than the earliest time it could be posted. CICS cannot do this for you. If you wait on an ECB that has been previously posted and is not subsequently cleared, your task is not suspended and continues to run as though WAITCICS or WAIT EXTERNAL had not been issued.

  WAIT EXTERNAL typically has less overhead, but the associated ECBs must always be posted using the MVS POST facility or by an optimized post (using the compare and swap (CS) instruction). They must never be posted by any other method. If you are in any doubt about the method of posting, use a WAITCICS command. When dealing with ECBs passed on a WAIT EXTERNAL command, CICS extends the ECBs and uses the MVS POST exit facility. A given ECB must not be waited on by more than one task at once (or appear twice in one task's ECBLIST). Failure to follow this rule leads to an INVREQ response.

  WAITCICS must be used if ECBs are to be posted by any method other than the MVS POST facility or by an optimized post. For example, if your application posts the ECB by moving a value into it, WAITCICS must be used. (The WAITCICS command can also be used for ECBs that are posted using the MVS POST facility or optimized post.) Whenever CICS goes into an MVS WAIT, it passes a list to MVS of all the ECBs being waited on by tasks that have issued a WAITCICS command. The ECBLIST passed by CICS on the MVS WAIT contains duplicate addresses, and MVS abends CICS.

  If you use MVS POST, WAIT EXTERNAL, WAITCICS, ENQ, or DEQ commands, you could create inter-transaction affinities that adversely affect your ability to perform dynamic transaction routing.

  To help you identify potential problems with programs that issue this command, you can use the CICS Interdependency Analyzer. See CICS Interdependency Analyzer for z/OS Overview for more information about this utility and Affinity for more information about transaction affinity.

Controlling sequence of access to resources

If you want a resource to be accessed by two or more tasks in a specific order, instead of ENQ and DEQ commands, use one or more WAITCICS commands with one or more hand-posted ECBs.

To hand-post an ECB, a CICS task sets a 4 byte field to either the cleared state of binary zeros, or the posted state of X'40008000'. The task can use a START command to start another task and pass the address of the ECB. The started task receives the address through a RETRIEVE command.

Either task can set the ECB or wait on it. Use the ECB to control the sequence in which the tasks access resources. Two tasks can share more than one ECB if necessary. You can extend this technique to control as many tasks as you want.

Note: Only one task can wait on a given ECB at any one time.

The example in Figure 93 on page 291 shows how two tasks can sequentially access a temporary storage queue by using hand-posted ECBs and a WAITCICS command.

The example uses two ECBs, (ECB1 and ECB2), addressed by the pointers illustrated in Figure 94 on page 291.

In theory, these tasks could exchange data through the temporary storage queue for ever. In practice, some code would be included to close down the process in an orderly way.

```
Task A Task B
Delete temporary storage queue
Clear ECB1 (set to X'00000000')
Clear ECB2
EXEC CICS START task B ( pass addresses EXEC CICS RETRIEVE
of PTR_ECB1_ADDR_LIST and (addresses passed)
PTR_ECB2_ADDR_LIST

LOOP: LOOP:
EXEC CICS WAITCICS Write to TS queue
ECBLIST(PTR_ECB1_ADDR_LIST) Post ECB1 (set to X'40008000)
NUMEVENTS(1) EXEC CICS WAITCICS
Clear ECB1 ECBLIST(PTR_ECB2_ADDR_LIST)
Read TS queue NUMEVENTS(1)
Process data
Delete TS queue
Write to TS queue
Post ECB2 ClearECB2
Go to start of loop Read TS queue
Process data
Delete TS queue
Go to start of loop
```

Figure 93. Two tasks using WAITCICS to control access to a shared resource

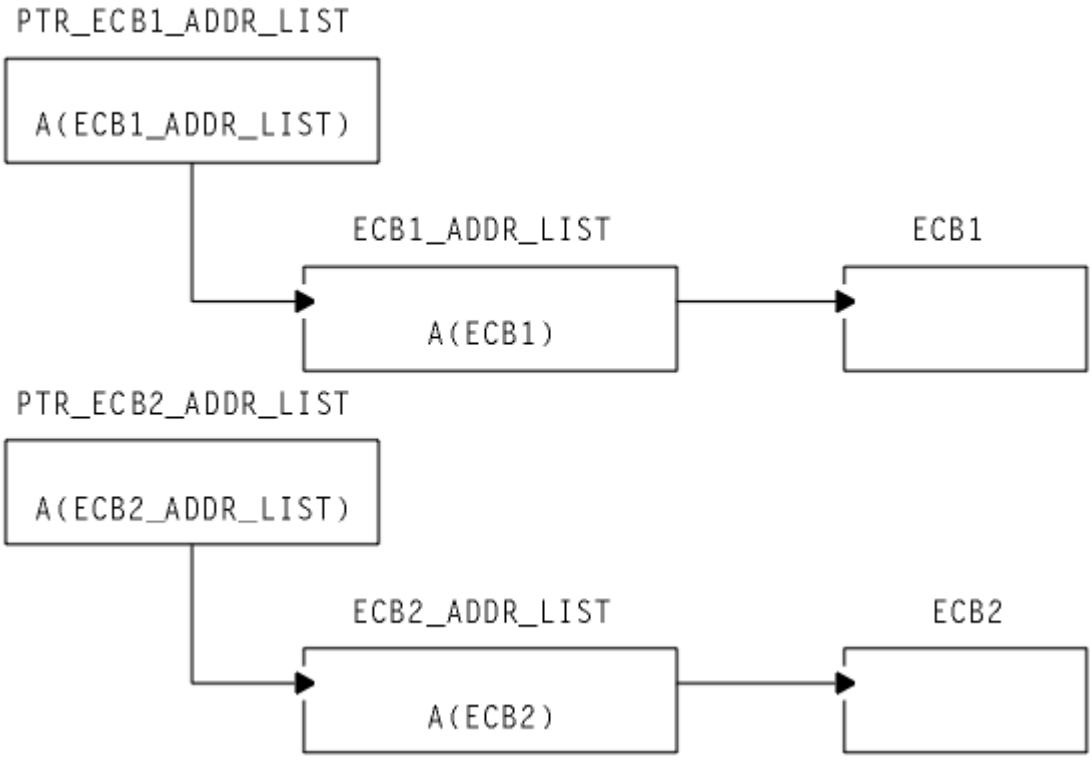


Figure 94. ECB pointers used by WAITCICS example

“Dealing with exception conditions” on page 189 describes how the exception conditions that can occur during processing of a task control command are handled.

CICS storage protection and transaction isolation

Storage control is affected by storage protection and transaction isolation. Storage protection protects CICS code and control blocks from applications, and transaction isolation protects tasks from each other.

The subsystem storage protection facility enables you to prevent your application programs from accidentally overwriting CICS code and control blocks. It does not provide protection against deliberately overwriting CICS code or control blocks. CICS cannot prevent an application from obtaining the necessary access (execution key) to modify CICS storage.

Transaction isolation extends this storage protection to provide protection for transaction data. If an application program accidentally overwrites the transaction data of another transaction, it can affect the reliability and availability of your CICS system, and the integrity of the data in the system.

The use of storage protection is optional; you use CICS system initialization parameters to specify whether to use storage protection facilities. For information about the relevant parameters, see Storage protection.

Storage control

The CICS storage control facility controls requests for main storage to provide intermediate work areas and other main storage needed to process a transaction.

Java and C++

The application programming interface described here is the CICS API, which is not used in Java programs. For information about Java programs using the JCICS classes to access storage control services, see Java development using JCICS and the JCICS Javadoc documentation. For information about C++ programs using the CICS C++ classes, see Using the CICS foundation classes.

CICS storage control facility

CICS makes working storage available within each command-level program automatically, without any specific request from the application program, and provides other facilities for intermediate storage, both within and among tasks. “Design for performance” on page 96 describes storage within individual programs. If you need working storage in addition to the working storage provided automatically by CICS, however, you can use the following commands:

- GETMAIN to get and initialize main storage
- FREEMAIN to release main storage

You can initialize the acquired main storage to any bit configuration by supplying the INITIMG option on the GETMAIN command; for example, zeros or EBCDIC blanks.

CICS releases all main storage associated with a task when the task is ended normally or abnormally. This includes any storage acquired, and not subsequently released, by your application program, except for areas obtained with the SHARED option. This option of the GETMAIN command prevents storage being released automatically when a task completes.

If you use the GETMAIN command with the SHARED option, and the FREEMAIN command, you could create inter-transaction affinities that adversely affect the ability to perform dynamic transaction routing.

To help you identify potential problems with programs that issue these commands, you can use CICS Interdependency Analyzer. See CICS Interdependency Analyzer for z/OS Overview for information about this utility and see “Affinity” on page 156 for information about transaction affinity.

If there is no storage available when you issue your request, CICS suspends your task until space is available, unless you specify the NOSUSPEND option. While the task is suspended, it may be canceled

(timed out) if the transaction definition specifies SPURGE(YES) and DTIMOUT(mmss). NOSUSPEND returns control to your program if storage is not available, allowing you to do alternative processing, as appropriate.

This section describes:

- " CICS storage protection and transaction isolation " on page 292
- "Defining the storage key for applications" on page 295
- "Selecting the execution and storage key " on page 298
- "Protection with transaction isolation" on page 301
- " MVS subspaces " on page 302

Storage protection

CICS allows you to run your application programs in either user-key or CICS-key storage. CICS storage is automatically protected from being overwritten by application programs that execute in user-key storage (the default).

The concept of isolating CICS code and control blocks (CICS internal data areas) from user application programs is illustrated in Figure 95 on page 293.

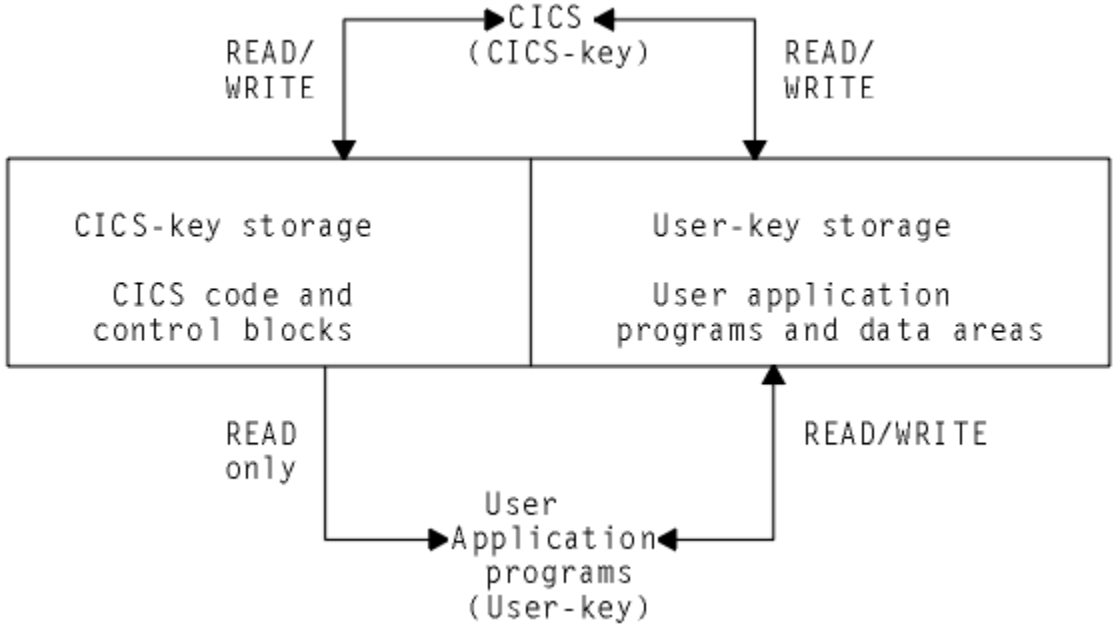


Figure 95. Protecting CICS code and control blocks from user application programs

Storage categories

When you are running with the storage protection facility active, there are two types of storage in which you can run application programs: CICS-key storage and user-key storage.

CICS-key storage
Used for most CICS system code and control blocks and, at the discretion of the installation, other code, and data areas that require protection from overwriting.

In a CICS region with transaction isolation active, a CICS-key program has read/write access to CICS-key and user-key storage.

User-key storage
Used for most application programs and their data areas.

There are two associated execution modes:

1. Many CICS system programs run in **CICS key** . A program running in CICS-key has read-write access to both CICS-key and user-key storage.

2. Application programs normally execute in **user key** . A program running in user-key has read-write access to user-key storage, but only read access to CICS-key storage.

The terms "user key" and "CICS key" therefore apply both to categories of storage and to the execution of programs. These terms are reflected in the resource definition keywords used in TRANSACTION definitions. See TRANSACTION attributes for more information.

The execution key controls the type of access your application programs have to CICS-key storage. The default is that application programs are given control in user key. Define CICS key only for those programs where it is essential that they execute in CICS key. The programs you might select to run in CICS key are typically those that are written by system programmers, and designed to provide special function in support of user applications. Such programs are considered to be an extension of CICS rather than part of an application. Some examples of such programs are described in “CICS-key applications” on page 299.

The storage protection facility does not protect CICS-key code and control blocks from being overwritten by programs that you choose to run in CICS key.

Defining the execution key

To run your programs in CICS key, you use the execution key parameter (EXECKEY) on the program resource definition. See “Selecting the execution and storage key ” on page 298 for an explanation of EXECKEY. The EXECKEY parameter determines the key in which CICS passes control to an application program.

Transaction isolation

Transaction isolation uses the MVS subspace group facility to offer protection between transactions. This ensures that an application program associated with one transaction cannot accidentally overwrite the data of another transaction.

Some of the benefits of transaction isolation, and its associated support are:

- Reducing system outages
- Protecting application data
- Protecting CICS from application programs that pass invalid addresses
- Aiding application development

Reducing system outages

Transaction isolation prevents data corruption and unplanned CICS system outages caused by coding errors in user-key application programs that cause the storage of user-key transactions to be accidentally overwritten. Prevention of accidental transaction data overwrites significantly improves the reliability and availability of CICS regions.

Protecting application data

If an application program overwrites CICS code or data, CICS can fail as a result. If an application program overwrites another application program's **code** , that other application program is likely to fail. Whereas this is a serious interruption in a production region, the effect is immediate and the program can generally be recovered so that the terminal user can retry the failed transaction. However, if an application program of one transaction overwrites the **data** of another transaction, the results often are not immediately apparent; the erroneous data can be written to a database and the error may remain undetected until later, when it may be impossible to determine the cause of the error. The consequences of a data overwrite are often much more serious than a code overwrite.

Protecting CICS from being passed invalid addresses

CICS also protects itself against applications that pass invalid addresses that would result in CICS causing storage violations. This occurs when an application program issues an EXEC CICS command that causes

CICS to modify storage on the program's behalf, but the program does not own the storage. In earlier releases, CICS did not check ownership of the storage referenced by the passed address, and executed such commands with consequent storage violations.

CICS validates the start address of the storage, and establishes that the application program has write access to the storage that begins with that address, before executing the command.

This address checking is controlled using the CMDPROT system initialization parameter. If a program passes an invalid address to CICS as an output field on the API, an AEYD abend occurs. It is completely independent of storage protection and transaction isolation.

Aiding application development

Transaction isolation aids application development in the testing and debugging phase. If an application tries to overwrite CICS or another application, or if it tries to pass a storage address it does not own for CICS to write to, CICS immediately abends the task and reports the rogue program's name and the area it tried to overwrite. This saves much time trying to debug what is a common problem in application development environments.

Defining the storage key for applications

You can choose between user-key storage and CICS-key storage for a number of CICS data areas and application program data areas that your applications can use.

About this task

Depending on the data area, you select the storage key by using one of the following:

- System initialization parameters
- Resource definition option
- Options on the GETMAIN commands

Defining the storage key for storage areas that your applications need to access is described in the following sections.

System-wide storage areas

For each CICS region, your installation can choose between user-key and CICS-key storage for the common work area (CWA) and for the terminal control table user areas (TCTUAs).

If these areas are in user-key storage, all programs have read-write access to them. If these areas are in CICS-key storage, user-key application programs are restricted to read-only access. The storage keys for the CWA and the TCTUAs are set by the system initialization parameters CWAKEY and TCTUAKEY, respectively. In both cases the default option is that CICS obtains user-key storage.

Task-lifetime storage

You can specify whether user-key or CICS-key storage is used for the storage that CICS acquires at transaction attach time, and for those elements of storage directly related to the individual application programs in a transaction.

To specify whether user-key or CICS-key storage is used, you use the TASKDATAKEY option on the transaction resource definition. This option governs the type of storage allocated for the following storage areas:

- The transaction work area (TWA) and the EXEC interface block (EIB)
- The copies of working storage that CICS obtains for each execution of an application program
- Any storage obtained for an application program in response to the following storage requests:
  - Explicit requests as a result of GETMAIN commands
  - Implicit requests as a result of a CICS command that uses the SET option

For information about how to specify the TASKDATAKEY parameter, see TRANSACTION attributes.

For an illustration of what TASKDATAKEY controls for both task lifetime storage and program working storage, see Figure 96 on page 297.

For programming information about EXEC CICS commands, see CICS command summary.

Program working storage specifically for exit and PLT programs

CICS uses the TASKDATAKEY option of the calling transaction to determine the storage key for the storage acquired for global user exits, task-related user exits, user-replaceable modules, and PLT programs.

For programming information about storage key, including details of how this affects the different types of program, see Exit programs and the CICS storage protection facility .

Passing data by a COMMAREA

In a pseudoconversational application, CICS ensures that a COMMAREA you specify on a RETURN command is always accessible in read-write mode to the next program in the conversation.

The same is true when passing a COMMAREA within a transaction that comprises more than one program (using a LINK or XCTL command). CICS ensures that the target program has read-write access to the COMMAREA.

The GETMAIN commands

The GETMAIN and GETMAIN64 commands provide USERDATAKEY and CICSDATAKEY options to enable the application program to explicitly request user-key or CICS-key storage, regardless of the TASKDATAKEY option specified on the associated transaction resource definition.

For example, you can use these options for application programs that are executing with TASKDATAKEY(CICS) specified to obtain user-key storage for passing to, or returning to, a program that is executing in user key.

In a program that is defined with EXECKEY(CICS) that issues GETMAIN commands to obtain CICS-key storage, such storage can be freed explicitly only if the FREEMAIN or FREEMAIN64 command is issued by a program that is also defined with EXECKEY(CICS). If an application program that is defined with EXECKEY(USER) attempts to free CICS-key storage by using FREEMAIN commands, CICS returns the INVREQ condition. However, an application can free user-key storage with FREEMAIN commands regardless of the EXECKEY option.

CICS frees all task-lifetime storage that is acquired by an application at task termination, whether it in CICS key or user key. You can also specify STORAGECLEAR(YES) on this option of the associated transaction resource definition. This clears the storage so that another task cannot view sensitive data accidentally.

For programming information about commands, see CICS API commands . For information about defining resources, see CICS resources.

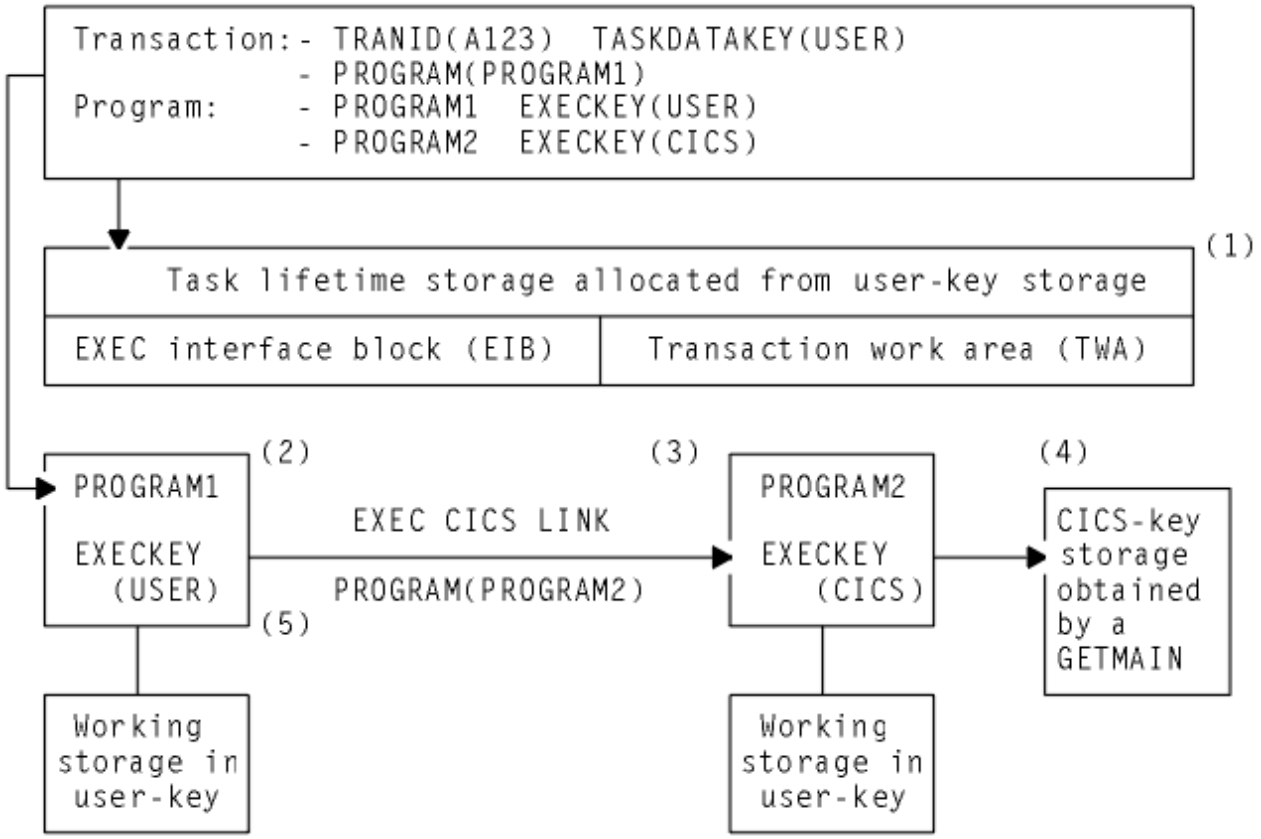


Figure 96. Illustration of the use of the TASKDATAKEY and EXECKEY options

In this example, transaction A123 is defined with TASKDATAKEY(USER) and PROGRAM(PROGRAM1). PROGRAM1 is defined with EXECKEY(USER) and PROGRAM2 is defined with EXECKEY(CICS). PROGRAM1 links to PROGRAM2, which uses a GETMAIN request to obtain CICS-key storage.

Notes:

1. The TASKDATAKEY option ensures that the transaction work area (TWA) and EXEC interface block (EIB) are allocated from user-key storage, as required for PROGRAM1, which executes in user key; specified by EXECKEY(USER).
2. PROGRAM1 executes in user key (controlled by EXECKEY), and has its working storage obtained in user-key storage (controlled by the TASKDATAKEY option). Any other storage the program obtains by using GETMAIN commands, or by using the SET option on a CICS command, is also obtained in user-key storage.
3. PROGRAM2 executes in CICS key (controlled by EXECKEY), but has its working storage obtained in user-key storage, which again is controlled by the TASKDATAKEY option.
4. PROGRAM2 issues an explicit GETMAIN command using the CICSDATAKEY option and, because it executes in CICS key, can store data into the CICS-key protected storage before returning control to PROGRAM1.
5. PROGRAM1 cannot write to the CICS-key protected storage that PROGRAM2 acquired, but can read what PROGRAM2 wrote there.

   When deciding whether you need to specify EXECKEY(CICS) and TASKDATAKEY(CICS), you must consider all the reasons that make these options necessary.

Programs that modify their storage protection key should ensure they are running in the correct key when attempting to access storage. CICS can only use the EXECKEY defined in the program definition when invoking a program.

Selecting the execution and storage key

When you are running CICS with storage protection, the majority of your application programs should execute in user key, with all their storage obtained in user key. You only need to define EXECKEY(CICS) on program definitions, and TASKDATAKEY(CICS) on the associated transaction definitions, for those programs that use facilities that are not permitted in user key, or for any special "system-type" transactions or vendor packages.

About this task

You should only specify TASKDATAKEY(CICS) for those transactions where all the component programs have EXECKEY(CICS), and for which you want to protect their task lifetime and working storage from being overwritten by user-key applications. For example, the CICS-supplied transactions such as CEDF are defined with TASKDATAKEY(CICS).

Note that you cannot specify EXECKEY(USER) on any programs that form part of a transaction defined with TASKDATAKEY(CICS) because, in this situation, a user-key program would not be able to write to its own working storage. Transactions abend with an AEZD abend if any program is defined with EXECKEY(USER) within a transaction defined with TASKDATAKEY(CICS), regardless of whether storage protection is active.

You cannot define a program so that it inherits its caller's execution key. The execution key and data storage keys are derived for each program from its program and associated transaction resource definitions respectively, which you either specify explicitly or allow to default; the default is always user key. Table 39 on page 298 summarizes the various combinations of options.

Table 39. Combinations of KEY options

EXECKEY	TASKDATAKEY	Recommended usage and comments
USER	USER	For normal applications using the CICS API
USER	CICS	**Not permitted** . CICS abends any program defined with EXECKEY(USER) invoked under a transaction defined with TASKDATAKEY(CICS).
CICS	USER	For programs that need to issue restricted MVS requests or modify CICS-key storage.
CICS	CICS	For transactions (and component programs) that function as extensions to CICS, such as the CICS-supplied transactions, or which require the same protection.

User-key applications

For most applications, you should define your programs with EXECKEY(USER), and the related transactions with TASKDATAKEY(USER).

To obtain the maximum benefits from the CICS storage protection facility, you are recommended to run your application programs in user key storage. Specifying USER on these options has the following effect:

EXECKEY(USER)
This specifies that CICS is to give control to the program in user key when it is invoked. Programs defined with EXECKEY(USER) are restricted to read-only access to CICS-key storage. These include:

- Storage belonging to CICS itself
- CICS-key storage belonging to user transactions defined with **TASKDATAKEY(CICS)**
- Application programs defined with **EXECKEY(CICS)** and thus loaded into CICS-key storage
- In a CICS region where transaction isolation is active, a user-key program has read/write access to the user-key task-lifetime storage of its own transaction and any shared DSA storage

TASKDATAKEY(USER)
This specifies that all task lifetime storage, such as the transaction work area (TWA) and the EXEC interface block (EIB), is obtained from the user-key storage.

It also means that all storage directly related to the programs within the transaction is obtained from user-key storage.

However, user-key programs of transactions defined with ISOLATE(YES) have access only to the user-key task-lifetime storage of their own task.

USER is the default for both the EXECKEY and TASKDATAKEY options, therefore you do not need to make any changes to resource definitions for existing application programs.

CICS-key applications

Some application programs need to be defined with **EXECKEY(CICS)** because they need to use certain facilities that are listed later.

Most application programs can be defined with **EXECKEY(USER)**, which is the default value, and this is the option you are recommended to use in most cases. These include programs that use DL/I or Db2 and programs that access vendor products through the resource manager interface (RMI) or a LINK command.

Widespread use of **EXECKEY(CICS)** diminishes the protection offered by the storage protection facility because there is no protection of CICS code and control blocks from being overwritten by application programs that execute in CICS key. The ISOLATE attribute in the transaction definition does not provide any protection against application programs that execute in CICS key—that is, from programs defined with **EXECKEY(CICS)**. Any application program causing a protection exception when defined with **EXECKEY(USER)** must be examined to determine why it is attempting to modify storage it is not allowed to modify. You should change a program's definition to **EXECKEY(CICS)** only if you are satisfied that the application program legitimately uses the facilities described here:

- The program uses MVS macros or services directly, rather than through the CICS API. The only MVS macros that are supported in user-key programs are SPIE, ESPIE, POST, WAIT, WTO, and WTOR. It is also possible to issue GTF trace requests from an EXECKEY(USER) program. If a program uses any other MVS macro or service, it must be defined with **EXECKEY(CICS)**. Some particular examples are:
  - Use of dynamic allocation (DYNALLOC macro, SVC 99)
  - Use of MVS GETMAIN and FREEMAIN or STORAGE requests
  - Use of MVS OPEN, CLOSE, or other file access requests

  Direct use of some MVS macros and services is undesirable, even in a CICS application defined with **EXECKEY(CICS)**. This is because they can cause MVS to suspend the whole CICS region until the request is satisfied.

  Some COBOL, PL/I, C, and C++ language statements, and compiler options, cause operating system functions to be invoked. See Chapter 8, “Developing COBOL applications,” on page 513 , Chapter 7, “Developing C and C++ applications,” on page 467, and Chapter 10, “Developing PL/I applications,” on page 547 for information about which of these should not be used in CICS application programs. It is possible that some of these functions might have worked in previous releases of CICS, or at least might not have caused the application to fail. They **do not work** when the program is defined with **EXECKEY(USER)**. When the use of prohibited options or statements is the cause of a protection exception, you should remove these from the program rather than redefine the program with **EXECKEY(CICS)**. The use of prohibited statements and options can have other side effects on the overall execution of CICS, and these should be removed.
- The program needs to modify the CWA, and the CWA is in CICS-key storage (**CWAKEY(CICS)**).

  If you decide to protect the CWA by specifying **CWAKEY(CICS)**, you should restrict the programs that are permitted to modify the CWA to as few as possible, perhaps only one. See “Using the common work area (CWA)” on page 104 for information about how you can control access to a protected CWA.
- The program needs to modify the TCTUA, and the TCTUAs are in CICS-key storage (**TCTUAKEY(CICS)**).

  See “Using the TCTTE user area (TCTUA)” on page 106 for information about using TCTUAs in a storage protection environment.

- The program can be invoked from PLT programs, from transactions defined with **TASKDATAKEY(CICS)**, from task-related or global user exits programs, or from user-replaceable programs.
- The program modifies CICS control blocks—for example, some vendor products that do need to manipulate CICS control blocks. These must be defined with **EXECKEY(CICS)**.
- The program provides user extensions to CICS and requires protection and data access like CICS system code. For example, you might consider that such programs are a vital part of your CICS installation, and that their associated storage, like CICS storage, should be protected from ordinary application programs.
- CICS always gives control in CICS key to the following types of user-written program, regardless of the option specified on their program resource definitions:
  - Global user exits (GLUEs)
  - Task-related user exits (TRUEs)
  - User-replaceable modules (URMs)
  - Program list table (PLT) programs
  - Custom EP adapters handling synchronous events

  CICS ensures that when control is passed to a PLT program, a global or task-related user exit, or a user-replaceable program, the first program so invoked executes in CICS key, regardless of the EXECKEY specified on its program resource definition. However, if this first program LINKs or XCTLs to other programs, these programs execute under the key specified in their program definitions. If these subsequent programs are required to write to CICS-key data areas, as often occurs in this type of situation, they must be defined as **EXECKEY(CICS)**.

  In a CICS region with transaction isolation active, these TRUEs and GLUEs run in either base space or subspace (see " MVS subspaces " on page 302), depending on the current mode when CICS gives control to the exit program. They can also modify any application storage. The URMs and PLT programs execute in base space.

For programming information about the execution of GLUEs, TRUEs, URMs, and PLT programs in a CICS region running with storage protection, see Exit programs and the CICS storage protection facility.

If two transactions have an affinity by virtue of sharing task lifetime storage, the transactions must be defined as ISOLATE(NO), or the programs must be defined as **EXECKEY(CICS)**. You can use the CICS Interdependency Analyzer to check the causes of transaction affinity. See CICS Interdependency Analyzer for z/OS Overview for more information about this utility. The first of these options is the recommended option, because CICS system code and data is still protected.

Tables

In addition to executable programs, you can define tables, map sets, and partition sets as program resources. EXECKEY has less relevance to these objects, because they are not executed. However, EXECKEY does control where non-executable objects are loaded, and thus affects whether other programs can store into them.

Map sets and partition sets

Map sets are not reentrant (BMS itself updates fields in maps when calculating absolute screen positions). However, map sets should not be modified by application programs; they must be modified only by CICS, which always executes in CICS key. CICS always loads map sets and partition sets into CICS-key storage.

Storage protection exception conditions

If an application program executing in user key attempts to modify CICS-key storage, a protection exception occurs. The protection exception is processed by normal CICS program error handling, and the offending transaction abends with an ASRA abend. The exception condition appears to the transaction as if it had attempted to reference any other protected storage. CICS error handling checks whether

the reference is to a CICS-key dynamic storage area (DSA), and sends a message (DFHSR0622) to the console. Otherwise, CICS does not treat the failure any differently from any other ASRA abend. See Dealing with transaction abend codes for more information about the storage protection exception conditions.

Protection with transaction isolation

In addition to being able to specify the storage and execution key for user transactions, you can also specify whether you want transaction isolation. Transaction isolation is built on top of storage protection, which means that STGPROT=YES must be specified. Transaction isolation uses parameters introduced by storage protection, these being EXECKEY, and TASKDATAKEY.

You can control transaction isolation globally for the whole CICS region with the TRANISO system initialization parameter. For individual transactions, the ISOLATE option of the transaction resource definition allows you to specify the level of protection that should apply to each transaction and program.

```
ISOLATE [
YES
or NO]
```

The defaults for these options mean that, in most cases, no changes to resource definition are needed for existing applications. However, where necessary, protection can be tailored to allow transactions to continue to function where they fail to meet the criteria for full protection, which is the default.

A user-key program invoked by transaction A (TXNA) can read and write to TXNA's user-key task lifetime storage and to shared user storage. Moreover, TXNA has no access to transaction B's (TXNB) user-key task lifetime storage.

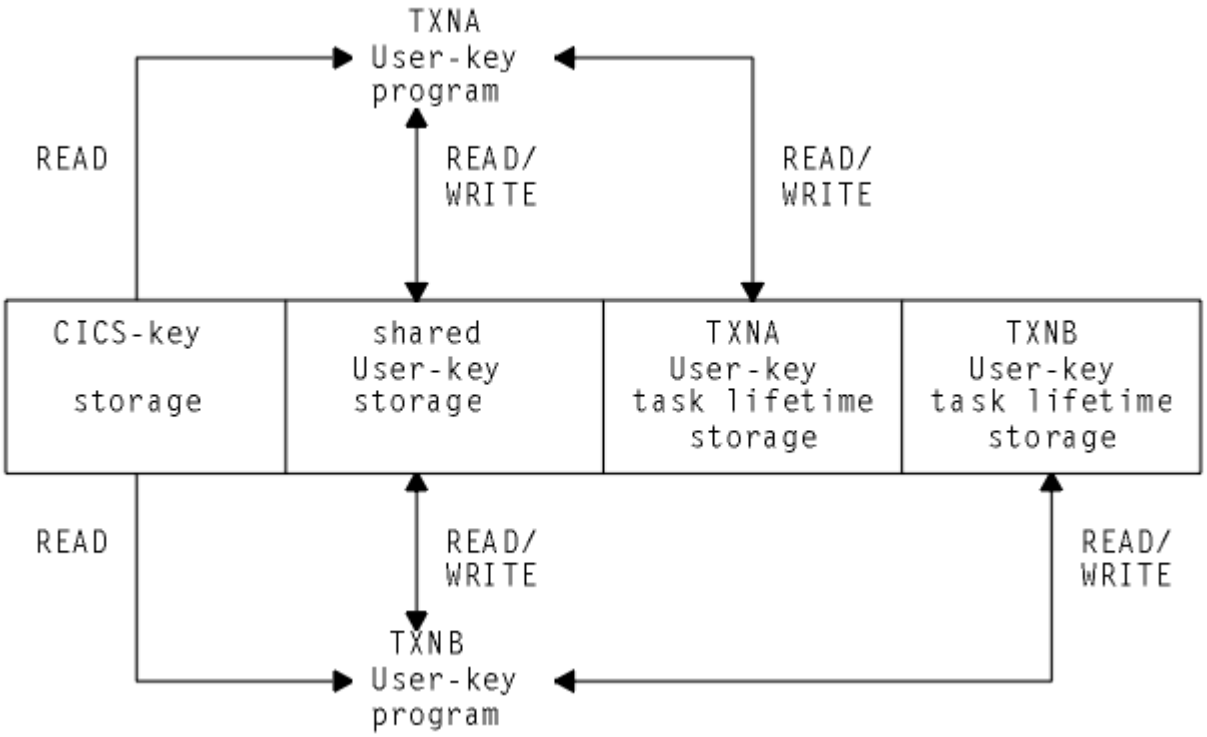


Figure 97. Two transactions defined as ISOLATE(YES)

If a transaction is defined as ISOLATE(NO), its user-key task lifetime is visible to all other transactions also defined as ISOLATE(NO). It is, however, protected from transactions defined as ISOLATE(YES).

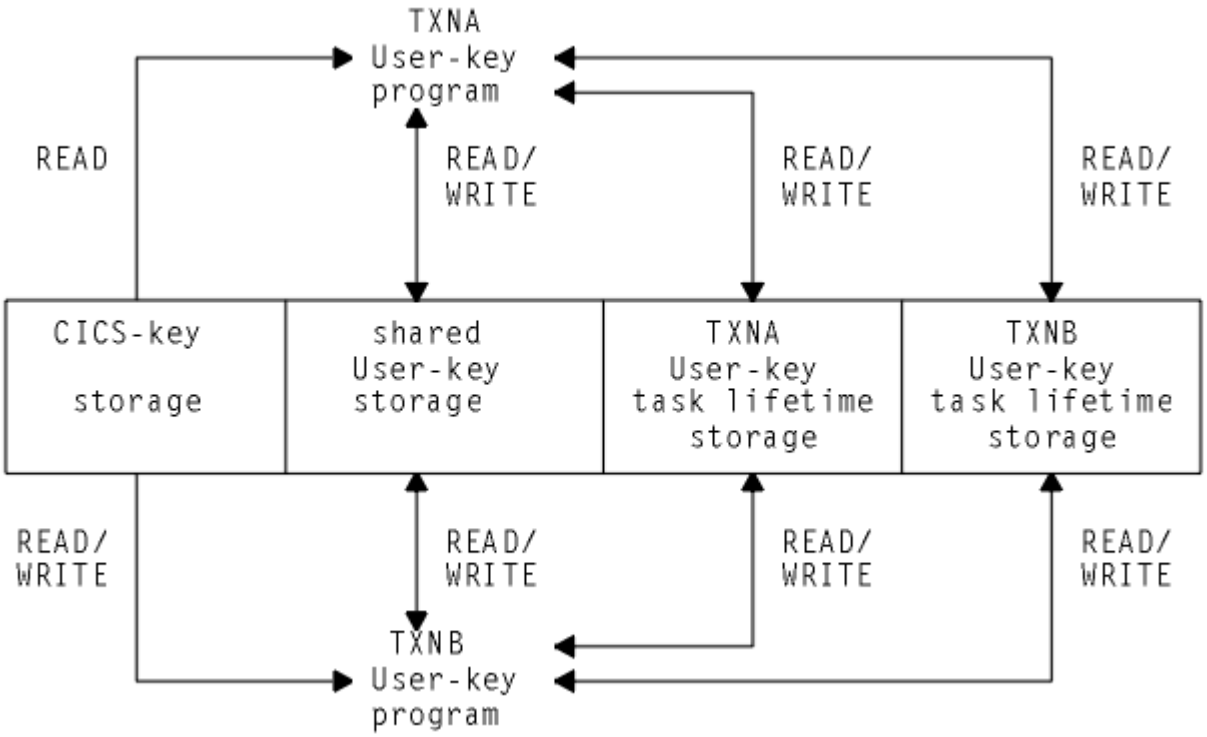


Figure 98. Two transactions defined as ISOLATE(NO) with read/write access to each other's task lifetime storage

User-key storage is not protected from CICS-key programs, even when you set ISOLATE(YES).

MVS subspaces

You can use the subspace group facility in MVS for storage isolation to preserve data integrity within an address space. The subspace-group facility uses hardware to provide protection for transaction data. A subspace-group is a group of subspaces and a single base space, where the base space is the normal MVS address space.

The subspace-group facility provides a partial mapping of the underlying base space, so that only specified areas of storage in the base space are exposed in a particular subspace. Each subspace represents a different subset of the storage in the base space. Transaction isolation, when specified, ensures that programs defined with EXECKEY(USER) execute in their own subspace, with appropriate access to any shared storage, or to CICS storage. Therefore, a user transaction is limited to its own "view" of the address space.

Programs defined with EXECKEY(CICS) run in the base space.

Subspaces and basespaces for transactions

In general, transaction isolation ensures that user-key programs are allocated to separate (unique) subspaces, and have:

- Read and write access to the user-key task-lifetime storage of their own tasks, which is allocated from one of the user dynamic storage areas (UDSA or EUDSA)
- Read and write access to shared storage; that is, storage obtained by GETMAIN commands with the SHARED option (SDSA or ESDSA)
- Read access to the CICS-key task-lifetime storage of other tasks (CDSA or ECDSA)
- Read access to CICS code
- Read access to CICS control blocks that are accessible by the CICS API

They do not have any access to user-key task-lifetime storage of other tasks.

The defaults for new transaction resource definition attributes specify that existing application programs operate with transaction isolation (the default for the ISOLATE option is YES). Existing applications should run unmodified provided they conform to transaction isolation requirements.

However, a minority of applications might need special definition if they do any of the following:

- Issue MVS macros directly
- Modify CICS control blocks
- Have a legitimate need for one task to access, or share, another task's storage

Some existing transactions can share task-lifetime storage in various ways, and this sharing might prevent them running isolated from each other. To allow such transactions to continue to execute, a single common subspace is provided in which all such transactions can execute. They are then isolated from the other transactions in the system that are running in their own subspaces, but able to share each other's data within the common subspace.

CICS-key programs execute in the base space and so have read/write access to all CICS-key storage and user-key storage.

The common subspace and shared storage

You might have some transactions where the application programs access one another's storage in a valid way. One such case is when a task waits on one or more event control blocks (ECBs) that are later posted, either by an MVS POST or "hand posting" , by another task.

For example, a task can pass the address of a piece of its own storage to another task (by a temporary storage queue or some other method) and then WAIT for the other task to post an ECB to say that it has updated the storage. Clearly, if the original task is executing in a unique subspace, the posting task fails when attempting the update and to post the ECB, unless the posting task is executing in CICS key.

CICS supports the following methods to ensure that transactions that must share storage can continue to work in the subspace-group environment:

- You can specify that all the related transactions are to run in the common subspace. The common subspace allows tasks that must share storage to coexist, while isolating them from other transactions in the system. Transactions assigned to the common subspace have the following characteristics:
  - They have read and write access to each other's task-lifetime storage.
  - They have no access of any kind to storage of transactions that run in unique subspaces.
  - They have read access only to CICS storage.

  Any group of related transactions that work in user key in previous releases of CICS should work under CICS Transaction Server for z/OS, Version 5 Release 6 if defined with ISOLATE(NO) to ensure that they run in the common subspace. This provides support for upgrading, allowing the separation of transactions into their own unique subspaces to be staged gradually after installing CICS and related support.
- You can ensure that the common storage is in SHARED storage by obtaining the storage with the SHARED option.
- You can ensure that the application programs of the transactions that are sharing storage are all defined with EXECKEY(CICS). This ensures that their programs execute in base space, where they have read/write access to all storage. However, this method is not recommended because it does not give any storage protection.

You can use the CICS Interdependency Analyzer to help you identify transactions that include the commands such as WAIT EVENT, WAITCICS, WAIT EXTERNAL, and MVS POST. See CICS Interdependency Analyzer for z/OS Overview for more information about this utility.

Transient data control

The CICS transient data control facility provides a generalized queuing facility. Data can be queued (stored) for subsequent internal or external processing. Selected data, specified in the application

program, can be routed to or from predefined symbolic *intrapartition* or *extrapartition* transient data queues.

Transient data queues are intrapartition if they are associated with a facility allocated to the CICS region, and extrapartition if the data is directed to a destination that is external to the CICS region. Transient data queues must be defined and installed before first reference by an application program.

You can:

- Write data to a transient data queue (WRITEQ TD command)
- Read data from a transient data queue (READQ TD command)
- Delete the contents of an intrapartition transient data queue (DELETEQ TD command)

If the TD keyword is omitted, the command is assumed to be for temporary storage. (See "Temporary storage control" on page 306 for more information about temporary storage.)

Java and C++

The application programming interface described here is the CICS API, which is not used in Java programs. For information about Java programs using the JCICS classes to access transient data services, see Java development using JCICS and the JCICS Javadoc documentation. For information about C++ programs using the CICS C++ classes, see Using the CICS foundation classes.

Intrapartition transient data queues

"Intrapartition" refers to data on direct-access storage devices for use with one or more programs running as separate tasks. Data directed to or from these internal queues is referred to as intrapartition data; it must consist of variable-length records.

All intrapartition transient data destinations are held as queues in the same VSAM data set, which is managed by CICS. An intrapartition destination requires a resource definition containing information that locates the queue in the intrapartition data set. Intrapartition queues can be associated with either a terminal or an output data set. When data is written to the queue by a user task, the queue can be used subsequently as input data by other tasks within the CICS region. All access is sequential, governed by read and write pointers. Once a record has been read, it cannot be read subsequently by another task. Intrapartition data may ultimately be transmitted upon request to the terminal or retrieved sequentially from the output data set.

Typical uses of intrapartition data include:

- Message switching
- Broadcasting
- Database access
- Routing of output to several terminals (for example, for order distribution)
- Queuing of data (for example, for assignment of order numbers or priority by arrival)
- Data collection (for example, for batched input from 2780 Data Transmission Terminals)

There are three types of intrapartition transient data queue:

- **Non-recoverable** Non-recoverable intrapartition transient data queues are recovered only on a warm start of CICS. If a unit of work (UOW) updates a non-recoverable intrapartition queue and subsequently backs out the updates, the updates made to the queue **are not** backed out.
- **Physically recoverable** Physically recoverable intrapartition transient data queues are recovered on warm and emergency restarts. If a UOW updates a physically recoverable intrapartition queue and subsequently backs out the updates, the updates made to the queue **are not** backed out.
- **Logically recoverable** Logically recoverable intrapartition transient data queues are recovered on warm and emergency restarts. If a UOW updates a logically recoverable intrapartition queue and subsequently backs out the changes it has made, the changes made to the queue are also backed

out. On a warm or an emergency restart, the committed state of a logically recoverable intrapartition queue is recovered. In-flight UOWs are ignored.

If an application is trying to issue a read, write, or delete request and suffers an indoubt failure, it may receive a LOCKED response if WAIT(YES) and WAITACTION(REJECT) are specified in the queue definition.

Extrapartition queues

Extrapartition queues (data sets) reside on any sequential device (DASD, tape, printer, and so on) that are accessible by programs outside (or within) the CICS region.

In general, sequential extrapartition queues are used for storing and retrieving data outside the CICS region. For example, one task may read data from a remote terminal, edit the data, and write the results to a data set for subsequent processing in another region. Logging data, statistics, and transaction error messages are examples of data that can be written to extrapartition queues. In general, extrapartition data created by CICS is intended for subsequent batched input to non-CICS programs. Data can also be routed to an output device such as a printer.

Data directed to or from an external destination is referred to as extrapartition data and consists of sequential records that are fixed-length or variable-length, blocked or unblocked. The record format for an extrapartition destination must be defined in a TDQUEUE resource definition by the system programmer. See TDQUEUE resources for details about queue definitions.

If you create a data set definition for the extrapartition queue using JCL, the DD statement for the data set must not include the FREE=CLOSE operand. If you do specify FREE=CLOSE , attempts to read the queue after the queue has been closed and then reopened can receive an IOERR condition. Defining data sets to CICS has more information about defining data sets to CICS.

Indirect queues

Intrapartition and extrapartition queues can be used as indirect queues. Indirect queues provide some flexibility in program maintenance in that data can be routed to one of several queues with only the transient data definition, and not the program itself, having to be changed.

When a transient data definition has been changed, application programs continue to route data to the queue using the original symbolic name; however, this name is now an indirect queue that refers to the new symbolic name. Because indirect queues are established by using transient data resource definitions, the application programmer does not usually have to be concerned with how this is done. Further information about transient data resource definition is in TDQUEUE resources .

Automatic transaction initiation (ATI)

For intrapartition queues, CICS provides the option of automatic transaction initiation (ATI). A basis for ATI is established by the system programmer by specifying a nonzero trigger level for a particular intrapartition destination.

When the number of entries (created by WRITEQ TD commands issued by one or more programs) in the queue reaches the specified trigger level, a transaction specified in the definition of the queue is automatically initiated. Control is passed to a program that processes the data in the queue; the program must issue repetitive READQ TD commands to deplete the queue.

When the queue has been emptied, a new ATI cycle begins. That is, a new task is scheduled for initiation when the specified trigger level is again reached, whether execution of the earlier task has ended. The exact point at which a new ATI cycle begins depends on whether the queue is defined as logically recoverable. If the queue is defined with a recoverability attribute (RECOVSTATUS) of No or Physical, the new ATI cycle begins when the queue is read to QZERO. But if the queue is defined with a recoverability attribute of Logical, the new ATI cycle begins only after the task terminates after having read the queue to QZERO.

If an automatically initiated task does not empty the queue, access to the queue is not inhibited. The task can be normally or abnormally ended before the queue is emptied (that is, before a QZERO condition

occurs in response to a READQ TD command). If the contents of the queue are to be sent to a terminal, and the previous task completed normally, the fact that QZERO has not been reached means that trigger processing has not been reset and the same task is reinitiated. A subsequent WRITEQ TD command does not trigger a new task if trigger processing has not been reset.

If the contents of the queue are to be sent to a file, the termination of the task has the same effect as QZERO (that is, trigger processing is reset). The next WRITEQ TD command initiates the trigger transaction (if the trigger level has been reached).

If the trigger level of a queue is zero, no task is automatically initiated.

If a queue is logically recoverable, initiation of the trigger transaction is deferred until the next sync point.

If the trigger level has already been exceeded because the last triggered transaction abended before clearing the queue, or because the transaction was never started because the MXT limit was reached, another task is not scheduled. This is because QZERO has not been raised to reset trigger processing. If the contents of a queue are destined for a file, the termination of the task resets trigger processing and means that the next WRITEQ TD command triggers a new task.

To ensure that an automatically initiated task completes when the queue is empty, the application program should test for a QZERO condition in preference to some other application-dependent factor (such as an anticipated number of records). Only the QZERO condition indicates an emptied queue.

If the contents of a queue are to be sent to another system, the session name is held in EIBTRMID. If a transaction (started with a destination of system) abends, a new transaction is started in the same way as a terminal.

If you use ATI with a transient data trigger mechanism, it might create inter-transaction affinities that adversely affect your ability to perform dynamic transaction routing. See “Affinity” on page 156 for more information about transaction affinity.

A trigger transaction is shunted if it suffers from an indoubt failure. Another trigger transaction is not attached until the shunted UOW commits or backs out the changes it has made following resynchronization.

Transactions that are initiated because of a transient data trigger-level transaction do not create previous hop data and are treated as executing at a point of origin. For more information about previous hop data, see Previous hop data characteristics.

Temporary storage control

The CICS temporary storage control facility provides the application programmer with the ability to store data in temporary storage queues, either in main storage, in auxiliary storage on a direct-access storage device, or in a temporary storage data sharing pool. Data stored in a temporary storage queue is known as temporary data.

For an overview of the different temporary storage locations and the features available for temporary storage queues in each location, see Main temporary storage: Monitoring and tuning .

The CICS temporary storage control facility includes commands for the following purposes. On each of these commands, the TS keyword can be omitted; temporary storage is assumed if it is not specified.

- Write data to a temporary storage queue (WRITEQ TS command).
- Update data in a temporary storage queue (WRITEQ TS REWRITE command).
- Read data from a temporary storage queue (READQ TS command).
- Read the next data from a temporary storage queue (READQ TS NEXT command).
- Delete a temporary storage queue (DELETEQ TS command).

For more information about how programmers can use temporary storage queues, see “Temporary storage queues” on page 91.

Exception conditions that occur during execution of a temporary storage control command are handled as described in “Dealing with exception conditions” on page 189.

If you use these commands, you could create inter-transaction affinities that adversely affect your ability to perform dynamic transaction routing.

To help you identify potential problems with programs that issue these commands, you can use the scanner and collector components of the CICS Interdependency Analyzer. See CICS Interdependency Analyzer for z/OS Overview for more information about this utility and “Affinity” on page 156 for more information about transaction affinity.

Java and C++

The application programming interface described here is the CICS API, which is not used in Java programs. For information about Java programs using the JCICS classes to access temporary storage services, see Java development using JCICS and the JCICS Javadoc documentation. For information about C++ programs using the CICS C++ classes, see Using the CICS foundation classes.

Typical uses of temporary storage control

A temporary storage queue that has only one record can be treated as a single unit of data that can be accessed using its symbolic name. Using temporary storage control in this way provides a typical scratch-pad capability. This type of storage should be accessed using the READQ TS command with the ITEM option; not doing so can cause the ITEMERR condition to be raised.

In general, temporary storage queues of more than one record should be used only when direct access or repeated access to records is necessary; transient data control provides facilities for efficient handling of sequential data sets.

Some uses of temporary storage queues are:

Terminal paging
A task could retrieve a large main record from a direct-access data set, format it into several screen images (using BMS), store the screen images temporarily in auxiliary storage, and then ask the terminal operator which page (screen image) is wanted. The application programmer can provide a program (as a generalized routine or unique to a single application) to advance page by page, advance or back up a relative number of pages, and so on.

A suspend data set
Suppose a data collection task is in progress at a terminal. The task reads one or more units of input and then allows the terminal operator to interrupt the process by some coded input. If not interrupted, the task repeats the data collection process. If interrupted, the task writes its incomplete data to temporary storage and terminates. The terminal is now free to process a different transaction (perhaps a high-priority inquiry). When the terminal is available to continue data collection, the operator initiates the task in a "resume" mode, causing the task to recall its suspended data from temporary storage and continue as though it had not been interrupted.

Preprinted forms
An application program can accept data to be written as output on a preprinted form. This data can be stored in temporary storage as it arrives. When all the data has been stored, it can first be validated and then transmitted in the order required by the format of the preprinted form.

CICS documents and document templates

Application programs can create documents and place data into them using commands in the EXEC CICS DOCUMENT application programming interface. Document templates are portions of a document which can be created offline, or in another CICS program, and used by the application program to create the document.

Documents and document templates are most commonly used to produce Web pages provided by CICS Web support. They can contain HTML that is used as the body of an HTTP request or response. However, they are not limited to this use.

In Java applications, you can use the CICS Java class library (JCICS) to access document services. The **`Document`** class provides the Java implementation of the EXEC CICS DOCUMENT commands. For the

class documentation, see the Javadoc in JCICS class reference. Java applications can retrieve documents that were created by applications written in other programming languages, and work with the documents using the JCICS classes.

Documents

You can create an empty document in an application program using the EXEC CICS DOCUMENT CREATE command, and then build the contents with subsequent DOCUMENT INSERT commands. Or you can use DOCUMENT CREATE to create and build a document in one step. You can create a document using data specified by your application program, or using a document template, or using another document. The document handler returns a token (DOCTOKEN), which is used to identify the document on subsequent calls.

When a document has been created, the contents can be extended by issuing one or more DOCUMENT INSERT commands. Again, you can add data specified by your application program, or a document template, or another document. You can also insert bookmarks into the document between blocks of data, and use the bookmarks to add or replace data in the middle of the document.

Documents created by an application exist only for the length of the CICS task in which they are created. This means that when the last program in the CICS task returns control to CICS , all documents created during the task's lifetime are deleted. It is the application's responsibility to save a document before terminating if the document is going to be used in another task. You can obtain a copy of the document by using the DOCUMENT RETRIEVE command. The application can then save this copy to a location of its choice, such as a temporary storage queue. This copy can then be used to re-create the document.

Document templates

Document templates can be retrieved from several different sources, to suit the way they are used in the application program. The source of a document template can be any one of the following:

- A partitioned data set
- A CICS program
- A CICS file
- A z/OS UNIX System Services file
- A temporary storage queue
- A transient data queue
- An exit program

Document templates are defined using DOCTEMPLATE resource definitions, which specify the source of the document template.

Document templates can contain static data, and also symbols. Symbols represent variable data that is resolved at the time the template is added to the document, that is, at the time the DOCUMENT CREATE or DOCUMENT INSERT command is issued. The values to be substituted for the symbols are specified by the application program using the SYMBOLLIST option of the DOCUMENT CREATE command, or the SYMBOLLIST or SYMBOL options of the DOCUMENT SET command.

Document templates can also contain embedded commands to set default values for symbols, identify symbols, and embed another template.

As well as being used by application programs, document templates can be specified in a URIMAP definition to provide a static response to a Web client's HTTP request, with no application program needed. CICS produces a response using the document template as the body of the Web page.

Symbols and symbol lists

Symbols are used in document templates to enable application programs to customize documents with data appropriate to the current task, such as a user name or order number. Symbols represent variable

data that is resolved at the time the template is added to the document - at the time the DOCUMENT CREATE or DOCUMENT INSERT command is issued.

Each symbol has a name, and it can have a value assigned to it. When a document template containing symbols is inserted into a document, the CICS document handler uses the values assigned to the symbols for the process of **symbol substitution** , where the symbol in the document template is replaced by its value. For example, if the symbol ORDER_NUMBER is assigned a value of `0012345` , and used in a document template as follows:

```
Thank you! Your order number is &ORDER_NUMBER;.
```

the finished document after symbol substitution will be:

```
Thank you! Your order number is 0012345.
```

A **symbol reference** is included in a document template at the place where the value of the symbol is to be substituted in the text of the completed document. The symbol reference consists of the symbol name, preceded with an ampersand (&) and terminated with a semicolon (;). You can use a symbol in any location within the document template. You can also include a default value for the symbol at any location in the document template, using the embedded template command #set.

An application program that creates a document using the document template needs to assign values to the symbols that are to be substituted when the template is used. You can do this using the DOCUMENT SET command or the DOCUMENT CREATE command.

You can specify values for individual symbols using the SYMBOL and VALUE options on the DOCUMENT SET command. Alternatively, you can define several symbols in a single command by specifying a **symbol list** . A symbol list is a character string consisting of one or more definitions, each containing a name, an equals sign, and a value, with single byte separators (by default, an ampersand). It can be specified using the SYMBOLLIST option on the DOCUMENT CREATE or DOCUMENT SET command. For example, here is a symbol list that specifies the value '0012345' for the symbol ORDER_NUMBER, and the value 'Germany' for the symbol COUNTRY:

```
ORDER_NUMBER=0012345&COUNTRY=Germany
```

The values that have been assigned to symbols are held in a **symbol table**. There is a symbol table associated with each document template. When the #set command is used, or when an application provides a value for a symbol, the symbol name and its associated value are added to the symbol table. If the symbol already exists in the symbol table, its value is replaced by the new value. A symbol definition provided by an application overrides a default value provided by a #set command.

When a document template is inserted into a document by the DOCUMENT CREATE or DOCUMENT INSERT command, symbol substitution is carried out for all the symbols that exist in the symbol table, using their current values as specified in the symbol table. If the symbol has not been specified by the application program or by a #set command, no symbol substitution takes place, and the finished document includes the symbol name or the #echo command specifying the symbol.

When you use a template containing symbols, you need to specify the values you require for the symbols **before** inserting the template into a document. If you insert a template, but you have not assigned values to the symbols in it, the symbols will never be substituted. (This can occur if you create a document from a template without specifying a symbol list or symbol values.) After you have inserted the template into a document, you cannot change the values that CICS has put in the document in place of the symbols. If you specify values for symbols after inserting the template, your values are placed into the symbol table and used the next time that the template is inserted into a document, but your changes do not affect the values that have already been inserted into the document.

Data format for symbols and symbol lists

The support for symbols and symbol lists in the DOCUMENT application programming interface is designed to interpret data with a content type of **application/x-www-form-urlencoded** , as described in

the HTML 4 specification topic Form content types . This is the format in which data is returned when it has been entered into a form on a Web page. (Web pages are the most common use for CICS document templates.)

The format of URL-encoded data is a sequence of name and value pairs, delimited by ampersands, for example:

```
firstname=Irving&lastname=Berlin
```

The values are normally data which the user has entered in the form.

However, if the data entered in the form contains characters that have a special meaning in the format of url-encoded data, the characters entered in the form need to be distinguished from those used to delimit the name and value pairs. To achieve this, a process of character escaping is performed. An escaped character is a character that has been replaced by the three-character sequence %xx , where xx is the hexadecimal value of the ASCII encoding of the character.

The ampersand (&), the equals sign (=), and also the percent sign (%), because it is used to introduce the escape sequence, must all be escaped in url-encoded data. Because the space character is often used as a delimiter, it is always escaped too. However, because spaces are so common, the URL-encoded data type allows spaces to be encoded as plus signs (+), rather than as an escape sequence. This in turn means that any plus signs entered in the form must also be escaped.

For example, if a form contains the following names and values:

```
sum 8+11=19
rate 19%
composers George & Ira Gershwin
```

the escaped url-encoded representation of this data is:

```
sum=8%2b11%3d19&rate=19%25&composers=George+%26+Ira+Gershwin
```

The space characters in the values have been encoded as plus signs, and the plus sign, equals sign, percent sign, and ampersand in the values have been encoded as escaped sequences.

A symbol list which you specify using the SYMBOLLIST option of the EXEC CICS DOCUMENT CREATE or DOCUMENT SET commands contains, in principle, a list of name and value pairs in the url-encoded format. However, CICS has extended this syntax in the following ways:

- Spaces do not have to be escaped. A space can be represented as a single space, or as a plus sign, or as the escape sequence %20.
- The delimiter between the name and value pairs does not have to be an ampersand. You can specify an alternative delimiter using the DELIMITER option of the command.

Normally, unescape processing is carried out for the values of symbols when they are put into the symbol table, so any characters specified using special coding are converted into the intended character. For example, plus signs are converted to spaces, and escape sequences are converted to the correct characters. However, if you specify the UNESCAPED option on the DOCUMENT CREATE or DOCUMENT SET command, no conversion takes place, and the symbol values are put into the symbol table exactly as you entered them.

Symbols in HTML comments

Within HTML comments in document templates, symbols are normally ignored, and the CICS document handler does not carry out symbol substitution. HTML comments are delimited by the opening markup <!-- and the closing markup -->.

If you want to have symbol substitution occur within an HTML comment in a document template, you can use symbols in place of the opening and closing delimiters of the comment, and use the #set command in the document template to provide the opening and closing delimiters as the values for those symbols. If you do this, CICS carries out symbol substitution for the whole of any comment around which you use the symbols in place of the delimiters.

For example, a document template could include:

```
<!--#set var=OC value='<!--'-->
<!--#set var=CC value='-->'-->

&OC; A comment containing my text &SYM; &CC;
```

If the application program assigns a value of 'Example text' to the symbol SYM, CICS substitutes the value of that symbol and also of the symbols OC and CC, producing the HTML output:

```
<!-- A comment containing my text Example text -->
```

Caching and refreshing of document templates

To improve performance, the CICS document handler caches a copy of most document templates. When applications reference the template, they use the cached copy, improving performance. You can refresh the cached copy at any time if the document template changes. You can also phase in a new copy of programs and exit programs that are defined as document templates.

CICS always caches a copy of the following types of document template:

- Templates in a partitioned data set
- Templates in a CICS file
- Templates in a z/OS UNIX System Services file
- Templates in a temporary storage queue
- Templates in a transient data queue

When one of these types of document template is installed individually while CICS is running, it is read into the CICS document handler's storage. Requests from applications to access the document template receive the cached copy of the template, so CICS does not need to access the location where the document template is stored each time. Document templates that are installed during CICS startup are not cached at that time; each of these document templates is cached when it is referenced for the first time by an application.

If you make changes to a document template that has been cached, you can refresh the cached copy of the document template using the CEMT or EXEC CICS SET DOCTEMPLATE NEWCOPY command. (Note that with the SET DOCTEMPLATE command, which is not part of the EXEC CICS DOCUMENT API, you need to specify the name of the DOCTEMPLATE resource definition which defines the document template, rather than the 48-character name of the template.)

For the types of document template listed above, the SET DOCTEMPLATE NEWCOPY command deletes the copy of the document template which is currently cached by the CICS document handler, and replaces it with a copy read from the location where the document template is stored. (For templates in a partitioned data set, CICS first performs a BLDL (build list) to obtain the most current directory information, and then rereads the member.) When a new cached copy has been created, subsequent requests to use the document template use the new copy. The new copy will be used by later requests within the same task, as well as requests in other tasks.

If the CICS system becomes short on storage, the document handler deletes some of the cached copies of document templates to attempt to relieve the storage constraint. The document templates to be deleted are selected in order of size, largest first, taking into account the time since the cached copy was created (so that newly created copies are not released immediately).

If the CICS system is restarted with a warm start, the document templates that were previously cached are not reloaded. The cache is repopulated as each document template is referenced for the first time by an application.

The CICS statistics collected for document templates show the number of times each document template is referenced, and the number of times a cached copy was made, refreshed, used and deleted.

Templates in CICS programs

Document templates retrieved from CICS programs are never cached by the document handler, because programs are already cached elsewhere in CICS.

For this type of document template, you can use the SET DOCTEMPLATE NEWCOPY command to phase in a new copy of the program. The command is equivalent to SET PROGRAM PHASEIN for the specified program. Subsequent requests to use the document template use the new copy, including later requests within the same task.

Templates in exit programs

For document templates generated by an exit program, the exit program specifies (in its exit parameter list) whether or not a copy of the document template should be cached by the document handler. The default is that the document template is not cached. Templates that change dynamically should not be cached, but if the template does not change, caching is suitable as it improves the performance of requests. If the exit program does specify caching, the cached copy is made when the document template is referenced for the first time by an application.

For this type of document template, you can use the SET DOCTEMPLATE NEWCOPY command to phase in a new copy of the exit program. The command is equivalent to SET PROGRAM PHASEIN for the specified exit program. When you issue the command, CICS deletes any cached copy of the document template, phases in the new copy of the program, and creates a new cached copy of the document template if the exit program specifies caching. The refreshed exit program can specify a different setting for whether or not caching should take place, and CICS honors the change.

Code page conversion for documents

The documents that an application creates may be transmitted to systems running on other platforms; for example, when a document is used to supply a Web page to a client. Textual data that is in the code pages used by the CICS system must be converted to the code pages used on the target system. This process is known as code page conversion.

A code page used by the CICS system is usually described as a host code page, except when CICS is acting as an HTTP client. A code page used by the target system is described as a client code page, or where the target system is a Web client or server using ASCII, it can be referred to as a character set.

You can make CICS documents include information about the code pages in which they have been created. When you create a document using the EXEC CICS DOCUMENT CREATE and EXEC CICS DOCUMENT INSERT commands, you can use the HOSTCODEPAGE option, together with any of the TEXT, FROM, TEMPLATE and SYMBOL options, to indicate the code page for that block of data. Each block can be specified in a different code page.

When you use the EXEC CICS DOCUMENT RETRIEVE command to retrieve a document for sending, you can specify the CHARACTERSET option to make the document handler convert all the individual blocks from their respective host code pages, into a single client code page that is suitable for use by the target system.

For CICS Web support, when a CICS document is specified for sending by a Web-aware application program using the EXEC CICS WEB API commands, the EXEC CICS DOCUMENT RETRIEVE command is not used by the application program. Instead, the document token of the CICS document is specified, and CICS manages retrieval of the document. Conversion to a client code page is handled by CICS , according to options that the application program specifies on the EXEC CICS WEB API commands.

Also for CICS Web support, when a CICS document template is specified in a URIMAP definition to provide a static response to a Web client, conversion to a client code page is handled by CICS . The host code page in which the template exists, and the client code page to which it should be converted, are specified in the URIMAP definition. When the static response is required, CICS creates a document using the template, retrieves the document, and carries out appropriate code page conversion.

Setting up document templates

Document templates are CICS resources, which you define using DOCTEMPLATE resource definitions.

The name of the template, which is used in EXEC CICS DOCUMENT API commands to refer to the template, is specified in the TEMPLATENAME attribute of the resource definition. The DOCTEMPLATE resource definition also specifies the source of the template, the format of the data (binary or EBCDIC), and whether CICS appends a carriage-return line-feed to each record in the template.

Related information

Encrypting data sets

Templates in a partitioned data set

HTML templates that have been created from BMS map set definitions are stored in partitioned data sets.

Document templates stored in partitioned data sets have the attribute MEMBERNAME in their DOCTEMPLATE resource definitions.

CICS loads a copy of the template from the partitioned data set when you install the corresponding DOCTEMPLATE definition. If you modify the template in the partitioned data set while CICS is running, issue the SET DOCTEMPLATE NEWCOPY command in order to activate the changes. CICS first performs a BLDL (build list) to obtain the most current directory information, and then rereads the member.

A partitioned data set used to store templates may have one of the following record formats:

- FB (fixed length blocked)
- VB (variable length blocked)
- U (unblocked)

Records may contain sequence numbers in the following cases:

- When the record format is FB, and the record length is 80; the sequence numbers must be in positions 73 through 80.
- When the record format is VB; the sequence numbers must be in positions 1 through 8.

In other cases, there must be no sequence numbers in the records. If you use sequence numbers, they must be present in all the records; do not use partially sequenced members.

Templates stored in a partitioned data set are cached by the CICS document handler.

Templates in z/OS UNIX System Services files

z/OS UNIX System Services files can be defined as document templates. You might find that z/OS UNIX files are the most convenient type of document template to create and edit.

Document templates stored in z/OS UNIX files have the attribute HFSFILE in their DOCTEMPLATE resource definitions.

For CICS Web support, you can deliver a z/OS UNIX file directly as a static response using a URIMAP definition, in which case the z/OS UNIX file does not need to be defined as a document template (although it may be). However, if you want to carry out symbol substitution, or a Web-aware application program needs to access the file, it must be defined as a document template.

In the DOCTEMPLATE definition, you need to provide the fully-qualified name of the file, which can be up to 255 characters in length. The CICS region must have permission to access z/OS UNIX , and it must have permission to access the directory containing the file, and the file itself. Giving CICS regions access to z/OS UNIX directories and files explains how to achieve this.

All the data in the file is used as the document template.

Templates stored in a z/OS UNIX file are cached by CICS. You can use the SET DOCTEMPLATE NEWCOPY command to delete the copy of the document template which is currently cached, and replace it with a copy read from the location where the document template is stored.

Templates in CICS files, temporary storage, or transient data

Consider using one of these resources when you want to use dynamic data from an application program in a document template.

Which resource you use depends on:

- How the application program stores its data.
- Whether the existing data can be used directly in the template, or needs to be modified.
- Whether the data must be preserved after it is used in the template.

In general, when a template is inserted into a document, all the data contained in the resource is used.

Document templates stored in these resources have the attributes FILE, TSQUEUE, and TDQUEUE, in their DOCTEMPLATE resource definitions.

If you are using a DOCTEMPLATE defined with the attribute TSQUEUE or TDQUEUE, two bytes (a carriage return and line feed) are added to the end of the DOCTEMPLATE buffer.

Temporary storage
The queue is read, in sequence, by ITEM number, and therefore all records in the queue are read, regardless of which records have been read by other applications.

Transient data

Extrapartition transient data queues are suitable for use as document templates. All records in the queue are read.

Intrapartition transient data queues are not suitable for use as document templates. Because transient data uses a destructive read, when you insert data from a transient data queue into a template, the contents of the queue are no longer available to other applications.

CICS file

- Entry-sequenced data sets (ESDS) are read in sequence of relative byte address.
- Relative record data sets (RRDS) are read in sequence of relative record number.
- Other data sets are read in sequence of key field.

CICS files that are to be used as document templates must be specified with the BROWSE attribute.

Templates stored in any of these formats are cached by CICS. You can use the SET DOCTEMPLATE NEWCOPY command to delete the copy of the document template which is currently cached, and replace it with a copy read from the location where the document template is stored.

Templates in CICS programs

A document template in a program has the least overhead for retrieving the template, although for other formats, caching also minimizes the overhead after the first request. Programs are an appropriate format to choose if the same template is used by several applications. A CICS program is most suitable where the template is not changed frequently, because a program cannot be easily edited and must be recompiled.

About this task

Document templates contained in CICS programs have the attribute PROGRAM in their DOCTEMPLATE resource definitions.

Templates contained in CICS programs are never cached by the CICS document handler. Programs are already cached by the CICS loader.

For this type of document template, you can use the SET DOCTEMPLATE NEWCOPY command to phase in a new copy of the program. The command is equivalent to SET PROGRAM PHASEIN for the specified program. Subsequent requests to use the document template use the new copy, including later requests within the same task.

To code a program which contains a template:

Procedure

1. Code an Assembler CSECT containing:
   a) An ENTRY statement, which denotes the start of the template.
   b) Character constants (DC statements) defining the text that you want to include in your template.
   c) An END statement.
   d) Invocations for the DFHDHTL macro, to generate a prolog and epilog for the document template. "DFHDHTL - program template prolog and epilog macro" on page 315 documents this macro.

   For example:

```
DOCTPROG CSECT
DOCTPROG AMODE 31
DOCTPROG RMODE ANY
DFHDHTL TYPE=INITIAL,ENTRY=WKLYHDR
DC CL4'<HR>'
DC CL29'<H2>Weekly Status Report</H2>'
DFHDHTL TYPE=FINAL
END WKLYHDR
```

   The DFHDHTL macro is used because the template program must be an exact multiple of eight bytes. If this is not the case, the binder might insert spurious binary characters at the end of the template, which can produce unpredictable output when the template is used in a document. The DFHDHTL macro creates the necessary padding.
2. Assemble and link edit the program into your CICS application program library.

   The name you give to the program can be different from the name of the entry point.
3. Create and install a DOCTEMPLATE resource definition which specifies the name of the program in the PROGRAM attribute.

What to do next

CICS will autoinstall the program on first reference, or you can create and install a PROGRAM resource definition.

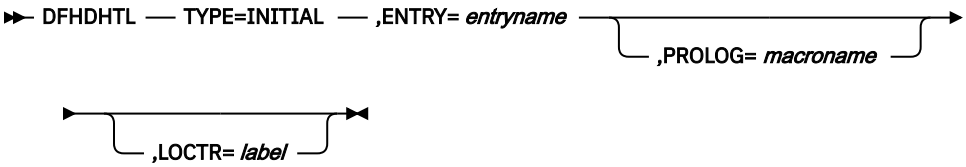
DFHDHTL - program template prolog and epilog macro

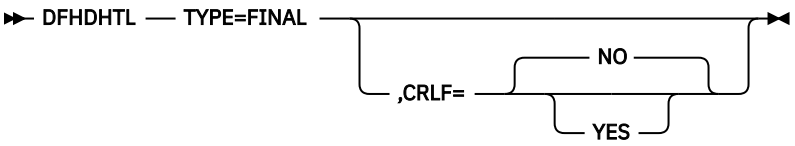
The DFHDHTL macro is used to generate the prolog and epilog for document templates contained in a CICS program (with the attribute PROGRAM in the DOCTEMPLATE resource definition).

These templates are defined as occupying the area of a load module between its entry point and the end of the module. However, if the source of the template program does not generate an exact multiple of eight bytes, the linkage editor or binder might insert spurious binary characters, which have an unpredictable effect when transmitted to a Web browser. The DFHDHTL macro helps to ensure that the template load module is always an exact multiple of eight bytes.

The prolog created by TYPE=INITIAL generates a module eyecatcher by invoking a user-specified macro (defaulting to DFHVM), and a new location counter (LOCTR) which will track the body of the template.

The epilog created by TYPE=FINAL calculates the length of the template so far, and if it is odd, generates a single blank character. This ensures that the entry point is on a halfword boundary. The epilog then rounds up the module size by reverting to the original LOCTR of the prolog, and generating sufficient characters before the entry point to make the module length a multiple of eight bytes.





TYPE= { INITIAL | FINAL }
Specifies whether a prolog (INITIAL) or an epilog (FINAL) is to be generated.

ENTRY= *entryname*
Specifies the label for the entry of the module. This is the point at which the template starts.

PROLOG= *macroname*
Specifies a macro to be invoked to generate module identification data in the prolog. The default is DFHVM.

LOCTR= *label*
Labels the internally generated location counter (LOCTR) statement which separates the generated template from the prolog code. The default is an internally generated name.

CRLF= NO | YES
Specifies whether the epilog code is to terminate with a closing CRLF (carriage return line feed). The default is NO.

Templates in exit programs

An exit program is most suitable where the contents of the template change dynamically, or where you want to retrieve the contents of the template from a non-CICS resource (for example, Db2).

When an application program requests a template which is defined as being created in an exit program, CICS calls the specified program, and passes a communication area.

Templates contained in an exit program are cached by CICS only if you specify this in the exit parameter list for the program. The default is that the templates are not cached. If the exit program does specify caching, the cached copy is made when the document template is referenced for the first time by an application.

You can use the SET DOCTEMPLATE NEWCOPY command to phase in a new copy of the exit program. The command is equivalent to SET PROGRAM PHASEIN for the specified exit program. When you issue the command, CICS deletes any cached copy of the document template, phases in the new copy of the program, and creates a new cached copy of the document template if the exit program specifies caching. The refreshed exit program can specify a different setting for whether or not caching should take place, and CICS honors the change.

Communication area for templates in exit programs

The communication area for an exit program which supplies a document template is mapped by a CICS-supplied copybook.

The supplied copybooks are:

- DFHDHTXD (Assembler)
- DFHDHTXH (C)
- DFHDHTXL (PL/I)
- DFHDHTXO (COBOL)

The communication area contains the following fields:

dhtx_template_name_ptr
Contains a pointer to the name (up to 48 characters) of the template that is being requested.

dhtx_buffer_ptr
Contains the pointer of the CICS-supplied buffer in which the exit program returns the template.

dhtx_buffer_len
(Fullword binary.) Contains the length of the CICS-supplied buffer in which the exit program returns the template.

dhtx_message_len
(Fullword binary.) Use this field to return the length of a message that is issued when the exit program is unable to return a template. If there is no message, return a value of zero.

dhtx_message_ptr
Use this field to return the pointer of a message that explains why the exit program was unsuccessful. CICS writes this message to the CSDH transient data destination. If there is no message, return a value of zero.

dhtx_template_len
(Fullword binary.) Use this field to return the actual length of the template.

dhtx_append_crlf
Use the characters '1' (append) or '0' (do not append) to specify whether or not to add carriage return and line feed characters to the end of each line.

dhtx_return_code
(Fullword binary.) Use this field to indicate whether the exit program has successfully returned a template:

- A return code of 0 indicates that the exit has returned a template.
- A return code of 8 indicates that the exit has not returned a template. In this case, CICS raises a TEMPLATERR condition in the application program.

dhtx_cache_response
Use the characters '1' (cache) or '0' (do not cache) to specify whether or not the output from the exit program should be cached by the CICS document handler. The value of `dhtx_cache_response` is initialized to '0', so the default action is not to cache the response of the exit program, unless the exit changes this value.

When a document template is cached, subsequent requests receive the cached copy. The exit program is not called again as long as the cached copy is available, until the EXEC CICS SET DOCTEMPLATE NEWCOPY command is issued to refresh the exit program and the cached copy. A refreshed exit program can specify a different value for `dhtx_cache_response` , and CICS honors the change.

Templates that change dynamically should not be cached, but if the template does not change, caching is suitable as it improves the performance of requests.

If the template to be returned is longer than `dhtx_buffer_len` , the template must be truncated to length `dhtx_buffer_len` and the exit program must set the length required in `dhtx_template_len` . The exit program is then called again with a larger buffer.

If your exit program sets a return code of 8, you can return an explanatory message, which is written to the CSDH transient data destination. Return the address and length of the message in `dhtx_message_ptr` and `dhtx_message_len` respectively. The storage which contains the message must be accessible to the caller of the exit program. For example, your exit program can issue a GETMAIN command to acquire storage for the message. CICS will release the storage when the task ends, unless you specify the SHARED option on the command.

Using symbols in document templates

Use a symbol reference or #echo command in a document template at the place where the value of the symbol is to be substituted in the text of the completed document. You can use the #set command in the template to specify a default value for a symbol.

Before you begin

The name of a symbol must contain only uppercase and lowercase letters, numbers and the special characters dollar ($), underscore (_), hyphen (-), number sign (#), period (.) and at sign (@). The name is case-sensitive, so uppercase letters are regarded as different from lowercase letters.

Procedure

- You can include a symbol reference in a document template by specifying the symbol name, preceded with an ampersand (&) and terminated with a semicolon (;).
  For example, the symbol ORDER_NUMBER could be specified in a template as follows:

  ```
  Thank you! Your order number is &ORDER_NUMBER;.
  ```

- You can also use the #echo command in the document template to specify a symbol.

  Specify the symbol name in the command, but do not use an ampersand and semicolon.

  For example, the symbol USER_NAME could be specified as follows:

  ```
  Welcome to the site,
  <!--#echo var=USER_NAME-->!
  ```

- You can use the #set command in the template to specify a default value for a symbol.
  For example, you could include the symbol USER_NAME and set a default value for it as follows:

  ```
  <!--#set var=USER_NAME value='New User'-->
  Welcome to the site,
  <!--#echo var=USER_NAME-->!
  ```

  or as follows:

  ```
  <!--#set var=USER_NAME value='New User'-->
  Welcome to the site, &USER_NAME;!
  ```

  When either of these templates is used with the default value for USER_NAME, the document appears as

  ```
  Welcome to the site, New User!
  ```

What to do next

An application program that creates a document using the document template needs to define values for the symbols that are to be substituted when the template is used. If you provide a default value for a symbol, this is used if the application does not define a value for that symbol. Remember that you must define values for the symbols before, or at the time when, you insert the template into the document. You cannot change the substituted values of the symbols after the template has been inserted.

Embedded template commands

The CICS document handler recognizes three commands that can be embedded in a document template. The three commands that are supported are #set, #echo and #include.

Syntax for embedded template commands

Embedded template commands follow the syntax rules for Server Side Include commands. A Server Side Include command starts with the characters left angle bracket, exclamation mark, hyphen, hyphen,

number sign (hash), followed by the command. It is terminated with the characters hyphen, hyphen, right angle bracket. For example:

```
<!--#command-->
```

The characters used to start and end the Server Side Include must be in code page 037, otherwise the command is ignored. The hexadecimal equivalents for these character sequences are X'4C5A60607B' and X'60606E'.

#echo command

The #echo command identifies a symbol that must be substituted when the template is inserted into the document. For example, this #echo command identifies the symbol ASYM to be substituted into a document at the location of the command:

```
This is the <!--#echo var=ASYM--> symbol.
```

When the template is used, the string containing the #echo command is completely replaced by the value defined for the symbol. If no symbol definition has been provided with that name, the #echo command remains in the output data. The value for the symbol can be defined by an application program, or as a default value by a #set command in the template.

#set command

The #set command is used to set the values of symbols. It is useful for setting up default values for symbols. For example, this #set command specifies a default value of 'first' for the symbol ASYM:

```
<!--#set var=ASYM value='first'-->
```

A #set commands can override another #set command. If you include more than one #set command in the template for the same symbol name, the last command is used.

A #set command in the template is ignored if the symbol to which it applies has already been given a value using the DOCUMENT SET command. A symbol which has been assigned a value using the DOCUMENT SET command can only be changed by issuing another DOCUMENT SET command.

The #set command can be used in combination with the #echo command, or it can apply to a symbol reference that has been specified by preceding the symbol name with an ampersand (&) and terminating it with a semicolon (;).

#include command

The #include command allows a template to be embedded within another template. Up to 32 levels of embedding are allowed.

For example:

```
<!--#include template=
templatename-->
```

templatename is the name of the template (the 48 byte name) defined in the DOCTEMPLATE resource definition. The template name can also be enclosed in double quotes.

Programming with documents and document templates

This section explains how to use documents and document templates in your application programs.

Creating a document

You can use the DOCUMENT CREATE command to create either an empty document, or a document containing data. The data can be a character string, a block of binary data, a document template, or a buffer of data.

About this task

To create a document containing data, you can specify options on the EXEC CICS DOCUMENT CREATE command to:

- Include a character string (TEXT option).
- Include a block of binary data (BINARY option).
- Use a document template, specified by its template name (TEMPLATE option).
- Include the contents of a buffer of data (FROM option).
- Give values to any symbols in the document template or the item specified by the FROM option (SYMBOLLIST option).

The DOCUMENT CREATE command has a DOCTOKEN option, which is mandatory and requires a 16–byte data-area. The document handler uses the DOCTOKEN operand to return a token, which is used to identify the document on subsequent calls.

These examples show the different ways in which an application can use EXEC CICS DOCUMENT commands to create a document. In Java applications, you can use the CICS Java class library (JCICS) to access document services. The **`Document`** class provides the Java implementation of the EXEC CICS DOCUMENT commands. For the class documentation, see the Javadoc in JCICS class reference.

Procedure

1. To create an empty document and return its token, use the EXEC CICS DOCUMENT CREATE command with the DOCTOKEN option.
   This example creates an empty document, and returns the token in the 16-character variable MYDOC:

   ```
   EXEC CICS DOCUMENT CREATE
   DOCTOKEN(MYDOC)
   ```

2. Use the TEXT option to create a document that contains a character string specified by your application program.
   For example, if you define a character string variable called DOCTEXT and initialize it to *This is an example of text to be added to a document* , you can use the following command to create a document consisting of this text string:

   ```
   EXEC CICS DOCUMENT CREATE
   DOCTOKEN(MYDOC1)
   TEXT(DOCTEXT)
   LENGTH(53)
   ```

   This string is added to the document unchanged, and CICS does not carry out any symbol substitution for it.
3. Use the BINARY option to create a document that contains binary data, which does not undergo code page conversion when the data is sent.
   This example creates a document consisting of the contents of a data-area as binary data:

   ```
   EXEC CICS DOCUMENT CREATE
   DOCTOKEN(MYDOC2)
   BINARY(DATA-AREA)
   ```

CICS does not carry out any symbol substitution for this data, and marks the data so that it is not converted to a client code page when you send the document to the recipient.

4. Use the TEMPLATE option to create a document using a document template that you have defined to CICS using a DOCTEMPLATE resource definition:
   a) Define a 48-byte variable, such as TEMPLATENAME, and initialize it to the value of the 48-character name of the template, as specified in the TEMPLATENAME attribute of its DOCTEMPLATE resource definition.
   b) If your document template contains no symbols, or you want to use the default values for the symbols, you can use the DOCUMENT CREATE command without the SYMBOLLIST option.
   For example:

```
EXEC CICS DOCUMENT CREATE
DOCTOKEN(MYDOC3)
TEMPLATE(TEMPLATENAME)
```

   It is important to note that you can only specify values for symbol substitution before, or at the time when, the document template is placed into the document. You cannot change the substituted values of the symbols after the template has been inserted.
   c) If you want to set values for symbols in the document template, use the DOCUMENT CREATE command with the SYMBOLLIST option.
   For example:

```
EXEC CICS DOCUMENT CREATE
DOCTOKEN(MYDOC3)
TEMPLATE(TEMPLATENAME)
SYMBOLLIST('ORDER_NUMBER=0012345')
LISTLENGTH(20)
```

5. Use the FROM option to create a document using a buffer of data.
   The buffer of data can contain symbol references that will be substituted in the same way as symbol references contained in document templates.
   For example:

```
EXEC CICS DOCUMENT CREATE
DOCTOKEN(MYDOC4)
FROM(BUFFER)
SYMBOLLIST('ORDER_NUMBER=0012345')
LENGTH(LEN)
```

Defining symbol values

Your application can define values for symbols in a document template using the DOCUMENT SET command or the DOCUMENT CREATE command. The symbol values are substituted when the template is used, either by a DOCUMENT CREATE command or by a DOCUMENT INSERT command.

About this task

You can define individual symbols using the SYMBOL and VALUE options on the DOCUMENT SET command. Alternatively, you can define several symbols in a single command using the SYMBOLLIST option on the DOCUMENT CREATE or DOCUMENT SET command. By default, the symbol separator for the symbol list is an ampersand, but you can override this by using the DELIMITER option on the commands.

When you are designing your application, remember that:

- You must define values for the symbols before, or at the time when, you insert the document template into the document. You cannot change the substituted values of the symbols after the template has been inserted.
- Every symbol in a document template should be defined. If any symbol in the document template is not defined, either by the application program or by a #set command, no symbol substitution takes

place for that symbol, and the finished document includes the symbol name or the #echo command specifying the symbol.

- A symbol definition provided by an application using the DOCUMENT SET command overrides a default value provided for the symbol by a #set command.
- A symbol which has been assigned a value using the DOCUMENT SET command can be changed by issuing another DOCUMENT SET command. If you do this after the document template has been inserted into the document, the new values are placed into the symbol table, and they will be used the next time that the template is inserted into a document. Your changes will not affect the values that have already been inserted into the document.

"Rules for specifying symbols and symbol lists" on page 323 lists the rules relating to symbol names, special characters and spaces in symbol values, and symbol list separator characters.

Procedure

1. To define an individual symbol, use the DOCUMENT SET command with the SYMBOL and VALUE options.

   The SYMBOL option specifies the name of the symbol, and the VALUE option specifies the value for that symbol. Follow the rules in "Rules for specifying symbols and symbol lists" on page 323 . That topic explains the effects of the UNESCAPED option.

   a) Make sure you issue the DOCUMENT SET command **before** you issue the DOCUMENT INSERT command which places the document template in the document.

   b) You cannot use the DOCUMENT CREATE TEMPLATE command if you need to use DOCUMENT SET commands to set symbol values. Instead, you can create an empty document first, then define the symbol values with DOCUMENT SET commands, and then use the DOCUMENT INSERT command to insert the template that contains the symbol references.

2. To provide a symbol list containing multiple symbol definitions, use the SYMBOLLIST option on the DOCUMENT CREATE or DOCUMENT SET command.

   When you use TEMPLATE and SYMBOLLIST together on DOCUMENT CREATE, the symbols from the symbol list are resolved when the template is added to the document.

   If you are using the DOCUMENT SET command, make sure you issue it **before** you issue the DOCUMENT INSERT command which places the document template in the document.

   a) If the default symbol separator **&** (ampersand) is not suitable, because you want to use that character in a symbol value, use the DELIMITER option to specify an alternative character.

   Follow the rules in "Rules for specifying symbols and symbol lists" on page 323.

   b) Use the SYMBOLLIST option to specify a buffer which contains the symbol list.

   Specify each symbol definition, separated by your chosen symbol separator.

   "Symbols and symbol lists" on page 308 shows an example of a symbol list. Follow the rules in "Rules for specifying symbols and symbol lists" on page 323. That topic explains the effects of the UNESCAPED option.

   c) Use the LISTLENGTH option to specify the length of your symbol list.

   Depending on your application, you might find that instead of specifying the exact list length for your symbol list each time you define values for the symbols, it is more convenient to choose a permanent value for the LISTLENGTH option, which will provide a fixed list length for your symbol list. If you do this:

   - The fixed list length that you choose should be long enough to include the maximum length of symbol list that you expect to supply.
   - When your fixed list length is greater than the actual length of the symbols that you supply, include an extra dummy symbol at the end of your symbol list, such as `'&END='` . This avoids the possibility of trailing spaces or unpredictable characters in the value of the last symbol in the list. Do not include this dummy symbol in any templates or documents. Any trailing spaces or unpredictable characters will be assigned to the dummy symbol and will not appear in your documents.

Results

The symbols that you define are placed into the symbol table. If you are defining a symbol list on the DOCUMENT CREATE command, symbol substitution takes place immediately, and the document is created using the document template and your specified symbols. If you are using the DOCUMENT SET command to define individual symbols or a symbol list, the values you have placed in the symbol table are used for symbol substitution when a DOCUMENT CREATE or DOCUMENT INSERT command is issued using that document template.

Rules for specifying symbols and symbol lists

Each symbol has a name and a value. Follow these rules for selecting the symbol name and specifying values for symbols, either individually or in a symbol list.

Symbol names

The name of a symbol must contain only uppercase and lowercase letters, numbers and the special characters dollar ($), underscore (_), hyphen (-), number sign (#), period (.) and at sign (@). The name is case-sensitive, so uppercase letters are regarded as different from lowercase letters.

To include a symbol in a document template, it can be specified as a symbol reference, which is the name of the symbol preceded with an ampersand (&) and terminated with a semicolon (;). Alternatively, the name of the symbol can be specified using the #echo command. When you specify the name of the symbol in your application, do not use an ampersand and semicolon. For example, the symbol reference &mytitle; in a template corresponds to the symbol name *mytitle* in a symbol list.

Separator in a symbol list

The SYMBOLLIST option on the DOCUMENT CREATE and DOCUMENT SET commands specifies a character string consisting of one or more definitions with single byte separators. By default, the symbol separator is an ampersand, but you can override this by using the DELIMITER option on the commands. Try to choose a symbol separator that will never be used within a symbol value in the symbol list, as special handling is required if you want to use the symbol separator within a symbol value. A non-printable character could be used.

There are several disallowed DELIMITER values. The disallowed values are:

- null (binary X'00')
- shift in (binary X'0E')
- shift out (binary X'0F')
- space (binary X'40')
- plus sign (binary X'4E')
- colon (binary X'7A')
- equals (binary X'7E')
- percent sign (binary X'6C')
- backslash (binary X'E0')

Special characters in symbol values

The values of symbols can contain any characters. However, special handling is required if you need to include the following characters in symbol values:

- The plus sign (+).
- The percent sign (%).
- The equals sign (=).
- The character that you have used as the symbol separator for a symbol list. Special handling is only required for this character when the symbol definition is provided in a symbol list, and does not apply

when you use the DOCUMENT SET command to set an individual symbol value with the SYMBOL and VALUE options.

In the symbol value, you can use escape sequences to include characters such as these that have a special meaning. An escape sequence consists of the percent sign, followed by two characters that are hexadecimal digits (that is, 0–9, a-f, and A-F). When the symbol values are put into the symbol table, the percent sign and the two hexadecimal digits following it are replaced by the EBCDIC equivalent of the single ASCII character denoted by the two digits.

Some useful combinations are shown in Table 40 on page 324 .

Table 40. Escape sequences that can be used to represent special symbols

Character	**Escape sequence**
Plus sign +	%2B
Percent sign %	%25
Equals sign =	%3D
Ampersand & (default symbol separator)	%26

If the characters following the percent sign are not two valid hexadecimal digits, the percent sign and the following characters are put into the symbol table as they appear in the symbol list.

If you prefer not to use escape sequences, you can specify the UNESCAPED option on the DOCUMENT CREATE or DOCUMENT SET command. When you specify this option, no conversion takes place, and the symbol values are put into the symbol table exactly as you entered them.

However, the UNESCAPED option does **not** allow you to include the character that you have used as the symbol separator, within a symbol value in a symbol list. If you want to use the UNESCAPED option, choose a symbol separator that will never be used within a symbol value. Alternatively, you can use the SYMBOL and VALUE options on the DOCUMENT SET command to specify symbol values that contain the character you have used as the symbol separator, because the symbol separator has no special meaning when used in the VALUE option.

Spaces in symbol values

If you want to include a space character in a symbol value, CICS allows you to use any of the following representations:

- A space character.
- A percent sign followed by the hexadecimal digits 20 (%20).
- A plus sign.

Any of these representations is interpreted as a space when the symbol value is put into the symbol table. This extends the HTML specification for the content type **application/x-www-form-urlencoded** , which uses a plus sign.

However, you **cannot** use a plus sign or the escape sequence %20 to indicate a space character when the UNESCAPED option is used to prevent CICS from unescaping symbol values contained in a symbol list or in the VALUE option. In these cases, you must use a space character to indicate a space, because plus signs are not converted to spaces when the UNESCAPED option is used.

Example: defining symbols without escape sequences

This example shows you how to pass symbol values to the document handler without using escape sequences. The symbol values in this list contain embedded plus signs, percent signs, and ampersands, none of which are to undergo unescape processing.

```
EXEC CICS DOCUMENT CREATE
DOCTOKEN(ATOKEN)
DELIMITER('!')
```

```
SYMBOLLIST('COMPANY=BLOGGS & SON!ORDER=NUTS+BOLTS')
LISTLENGTH(37)
UNESCAPED
```

In this example, the symbol COMPANY has a value of 'BLOGGS & SON', and the symbol ORDER has a value of 'NUTS+BOLTS'.

The use of a character other than the ampersand as the symbol separator means that an ampersand can be used in 'BLOGGS & SON'. The symbol separator used in this example is '!', but it is best to use a non-printable character that does not appear in the symbol value.

The use of the UNESCAPED option ensures that the plus sign in 'NUTS+BOLTS' does not get converted to a space. Because the UNESCAPED option has been used, you must use a space character, rather than a plus sign, to indicate where spaces are required in the symbol value 'BLOGGS & SON'. This means that the data no longer conforms to the specification for the content type **application/x-www-form-urlencoded**.

Adding more data to a document

When you have created a document, you can extend its contents by issuing one or more DOCUMENT INSERT commands. You can insert text, binary data, a buffer of data, a document template, or the value of a symbol. You can also insert bookmarks in a document, and use these to position later insertions.

About this task

You can specify options on the DOCUMENT INSERT command to:

- Insert a character string (TEXT option).
- Insert a block of binary data (BINARY option).
- Insert a document template, specified by its template name (TEMPLATE option).
- Insert the contents of a buffer of data (FROM option).
- Insert the value of a named symbol from the symbol table (SYMBOL option).

By default, the objects that you specify are added to the end of the document. To insert data in the middle of a document, you can set up one or more bookmarks. Bookmarks allow the application to insert blocks of data in any order yet still control the sequence of the data in the document.

A bookmark is a label placed between blocks of data - it cannot be placed in the middle of a block of data. You can use the DOCUMENT INSERT command to place a bookmark in a document during its construction, and then use the AT option to specify the bookmark when you insert a subsequent object. A special bookmark of 'TOP' is already defined, which you can use to insert data at the top of the document.

Procedure

1. Use the TEXT option to insert a character string specified by your application program.
   For example:

   ```
   EXEC CICS DOCUMENT INSERT
   DOCTOKEN(MYDOC)
   TEXT('Sample line 1.')
   LENGTH(15)
   ```

   This string is added to the document unchanged, and CICS does not carry out any symbol substitution for it.
2. Use the BINARY option to insert a block of binary data, which does not undergo code page conversion when the data is sent.
   For example:

   ```
   EXEC CICS DOCUMENT INSERT
   DOCTOKEN(MYDOC)
   BINARY(DATA-AREA)
   ```

CICS does not carry out any symbol substitution for this data, and marks the data so that it is not converted to a client code page when you send the document to the recipient.

3. Use the TEMPLATE option to insert a document template.

   If you want to set values for the symbols in the document template, use the DOCUMENT SET command to specify individual symbols or a symbol list, **before** you issue the DOCUMENT INSERT command. "Defining symbol values" on page 321 explains how to do this.

4. Use the FROM option to insert a buffer of data.

   The buffer of data can contain symbol references that will be substituted in the same way as symbol references contained in document templates.

5. Use the SYMBOL option to insert the value of a symbol.

   SYMBOL specifies the name of a valid symbol whose value has been set in the symbol table (using the DOCUMENT SET or DOCUMENT CREATE command). The document handler inserts the value that is associated with the symbol into the document.

   Note that after you have inserted a value associated with a symbol into a document, you cannot change that value in the document that is being composed. If you subsequently set a different value for the symbol, the new value will be used the next time that symbol is inserted into a document. Your change will not affect the value that has already been inserted into the document.

6. Use the BOOKMARK option to insert a bookmark into the document.

   Bookmarks are placed during the construction of a document, and they provide an insertion point for the application to insert data that is obtained at a later stage.

   The name of a bookmark must be 16 characters long. It must not contain any imbedded spaces, and if your chosen name is less than 16 characters long, it must be padded on the right with blanks.

   For example, this sequence of commands creates a document with two blocks of text and a bookmark:

```
EXEC CICS DOCUMENT CREATE
DOCTOKEN(MYDOCBOOK)
TEXT('Pre-bookmark text. ')
LENGTH(19)
```

```
EXEC CICS DOCUMENT INSERT
DOCTOKEN(MYDOCBOOK)
BOOKMARK('ABookmark ')
```

```
EXEC CICS DOCUMENT INSERT
DOCTOKEN(MYDOCBOOK)
TEXT('Post-bookmark text. ')
LENGTH(20)
```

7. Use the AT option with any of the other insertion options to place the object at a previously inserted bookmark, or at the special TOP bookmark.
   For example, this command inserts text at the example bookmark ABookmark:

```
EXEC CICS DOCUMENT INSERT
DOCTOKEN(MYDOCBOOK)
TEXT('Inserted at a bookmark. ')
LENGTH(25)
AT('ABookmark ')
```

   The completed document reads as follows:

```
Pre-bookmark text. Inserted at a bookmark. Post-bookmark
text.
```

Replacing data in a document

You can place bookmarks in a document to delimit an area of data that can be replaced by a later insertion, or deleted. This technique lets you provide a default item of text or other data in your document, which can be used if your application finds that no data is available to replace it.

About this task

To set up and replace, or delete, a default item of data in a document:

Procedure

1. Create the document, specifying any data to be used at the beginning of the document.

   For this example, a document is created with some initial text, and its token is returned in the variable MYDOCREP:

   ```
   EXEC CICS DOCUMENT CREATE
   DOCTOKEN(MYDOCREP)
   TEXT('Initial sample text. ')
   LENGTH(21)
   ```

2. Use the DOCUMENT INSERT command to specify the first bookmark:

   ```
   EXEC CICS DOCUMENT INSERT
   DOCTOKEN(MYDOCREP)
   BOOKMARK('BMark1 ')
   ```

3. Use the DOCUMENT INSERT command to specify the item of text or other data to be replaced:

   ```
   EXEC CICS DOCUMENT INSERT
   DOCTOKEN(MYDOCREP)
   TEXT('Text to be replaced. ')
   LENGTH(21)
   ```

4. Use the DOCUMENT INSERT command to specify the closing bookmark:

   ```
   EXEC CICS DOCUMENT INSERT
   DOCTOKEN(MYDOCREP)
   BOOKMARK('BMark2 ')
   ```

5. Use the DOCUMENT INSERT command to add any data to be used at the end of the document:

   ```
   EXEC CICS DOCUMENT INSERT
   DOCTOKEN(MYDOCREP)
   TEXT('Final sample text. ')
   LENGTH(19)
   ```

   At this point, the logical structure of this example document is:

   ```
    Initial sample text. <BMark1>Text to be replaced.
   <BMark2>Final
   sample text.
   ```

   The names of the bookmarks do not appear in the document.

6. To replace the text between the two bookmarks, BMark1 and BMark2, use the DOCUMENT INSERT command with the AT and TO options:

   ```
   EXEC CICS DOCUMENT INSERT
   DOCTOKEN(ATOKEN)
   TEXT('Replacement Text. ')
   LENGTH(18)
   AT('BMark1 ')
   TO('BMark2 ')
   ```

   Now the example document appears as follows:

   ```
    Initial sample text. Replacement Text. Final
   sample text.
   ```

7. To delete the text between the two bookmarks, use the DOCUMENT INSERT command with the AT and TO options as above, but use the TEXT or BINARY option to specify a null string, with a LENGTH of zero.

Retrieving, storing and reusing a document

Documents created by an application exist only for the length of the CICS task in which they are created. To reuse a document, the application needs to retrieve a copy and save it.

About this task

This sequence of commands shows how an application can use EXEC CICS DOCUMENT commands to create a document, retrieve it, store it on a temporary storage queue, and reuse it as a document in the same or another application.

Java applications can retrieve documents that were created by applications written in other programming languages, and work with the documents using the JCICS classes. For the class documentation, see the Javadoc in JCICS Javadoc information.

Procedure

1. In the application program, define and initialize the following variables:
   - A 16-byte field ATOKEN to hold the document token.
   - A 20-byte buffer DOCBUF to hold the retrieved document.
   - A fullword binary field called FWORDLEN to hold the length of the data retrieved.
   - A halfword binary field called HWORDLEN to hold the length for the temporary storage WRITE command.
2. Create the initial document using the DOCUMENT CREATE command:

   ```
   EXEC CICS DOCUMENT CREATE
   DOCTOKEN(ATOKEN)
   TEXT('A sample document.')
   LENGTH(18)
   ```

   To help the application calculate the size of the buffer needed to hold a retrieved document, the document commands which alter the size of the document (the DOCUMENT CREATE and DOCUMENT INSERT commands) have a DOCSIZE option available. This value is the maximum size of the buffer needed to contain a copy of the document in its original code page (including control information), when the RETRIEVE command is issued. However, when the CHARACTERSET option specifies an encoding that requires more bytes than the original EBCDIC data (for example, UTF-8), the maximum size might not be large enough to store the converted document. You can determine the actual document length before allocating the buffer by issuing a **DOCUMENT RETRIEVE** with a dummy buffer and a MAXLENGTH of zero, then handling the LENGERR condition and using the returned LENGTH value.
3. In the same task, issue the **DOCUMENT RETRIEVE** command to obtain a copy of the document in the application's own buffer.

   ```
   EXEC CICS DOCUMENT RETRIEVE
   DOCTOKEN(ATOKEN)
   INTO(DOCBUF)
   LENGTH(FWORDLEN)
   MAXLENGTH(20)
   ```

   By default, when the document is retrieved, the data that is delivered to the application buffer is stored in a form which contains control information necessary to reconstruct an exact replica of the document. CICS inserts tags into the document contents to identify the bookmarks and to delimit the blocks that do not require code page conversion. A document that you create from the retrieved copy is therefore identical to the original document. If you do not need to re-create the original document, you can modify the **DOCUMENT RETRIEVE** command as follows:

a) To request a copy without control information, specify the DATAONLY option.

   With this option, CICS omits all the imbedded tags. The retrieved document contains no bookmarks, and there are no markings to delimit blocks that do not require code page conversion.

b) To convert the whole of the copy into a single client code page, specify the CHARACTERSET option.

4. Store the document on the temporary storage queue:

```
EXEC CICS WRITEQ TS
QUEUE('AQUEUE')
FROM(DOCBUF)
LENGTH(HWORDLEN)
```

5. In the same or another application, read the stored data into the application's buffer:

```
EXEC CICS READQ TS
QUEUE('AQUEUE')
INTO(DOCBUF)
LENGTH(HWORDLEN)
```

6. Use the **DOCUMENT CREATE** command with the FROM option to create a new document using the contents of the data buffer, that is, the retrieved document:

```
EXEC CICS DOCUMENT CREATE
DOCTOKEN(ATOKEN)
FROM(DOCBUF)
LENGTH(FWORDLEN)
```

What to do next

You can also use the DOCUMENT RETRIEVE and DOCUMENT INSERT commands to insert a whole document into an existing document. The following variables must first be defined and initialized in the application program:

- A 16-byte field RTOKEN which contains the document token of the document to be retrieved
- A buffer DOCBUF of sufficient length to hold the retrieved document
- A fullword binary field called RETRIEVLEN to hold the length of the data retrieved
- A fullword binary field called MAXLEN to hold the maximum amount of data the buffer can receive, i.e. the length of DOCBUF
- A 16-byte field ITOKEN which contains the document token of the document that is being inserted into

The following sequence of commands shows a document indicated by RTOKEN being inserted into another document indicated by ITOKEN:

```
 EXEC CICS DOCUMENT RETRIEVE
DOCTOKEN(RTOKEN)
INTO(DOCBUF)
LENGTH(RETRIEVLEN)
MAXLENGTH(MAXLEN)

EXEC CICS DOCUMENT INSERT
DOCTOKEN(ITOKEN)
FROM(DOCBUF)
LENGTH(RETRIEVLEN)
```

The retrieved document is inserted at the end of the document specified in the DOCUMENT INSERT command, and all the control information of the retrieved document will be present in the second document. The LENGTH parameter of the DOCUMENT INSERT command must be equal to the value returned from the DOCUMENT RETRIEVE command into the field RETRIEVLEN.

Deleting a document

You can use the DOCUMENT DELETE command to delete documents that are no longer required during a transaction. On execution of the command, the storage allocated to the document is released

immediately. The DOCSTATUS(DOCDELETE) option of the WEB CONVERSE, WEB SEND (Client) and WEB SEND (Server) commands also allows document deletion.

About this task

DOCUMENT DELETE, WEB CONVERSE, WEB SEND (Client) and WEB SEND (Server) all use the DOCTOKEN to specify the 16–byte binary token of the document. The document token is returned when you create a document using the EXEC CICS DOCUMENT API commands.

To delete a document using the DOCUMENT DELETE command:

1. Specify the DOCTOKEN of the document you want to delete. For example:

```
EXEC CICS DOCUMENT DELETE
DOCTOKEN(MYDOC)
```

2. The document is deleted from the document handling domain and storage is released immediately. If ACTION(EVENTUAL) is specified in the command, the Web domain retains a copy of the document.

WEB CONVERSE, WEB SEND (Client) and WEB SEND (Server) allow you to delete a document by specifying the DOCSTATUS(DOCDELETE) option. This option allows the application to indicate that it no longer requires the document once it has issued the CONVERSE or SEND command. The document is deleted from the document handling and Web domains on completion of the WEB SEND command, and storage is released immediately.

If you issue a WEB SEND, specifying DOCSTATUS(NODOCDELETE) and ACTION(EVENTUAL) in the command, it is possible to retrieve the document using the WEB RETRIEVE command. Using the DOCSTATUS(DOCDELETE) option or using the ACTION(IMMEDIATE) option will remove the document permanently from Web storage, and the document cannot be retrieved. WEB RETRIEVE provides more information on the limitations of document retrieval after deletion of documents.

Named counter servers

CICS provides a facility for generating unique sequence numbers for use by application programs in a Parallel Sysplex environment. This facility is controlled by a named counter server, which maintains each sequence of numbers as a **named counter**.

Each time a sequence number is assigned, the corresponding named counter is incremented automatically. By default, the increment is 1, ensuring that the next request gets the next number in sequence. You can vary the increment when using the EXEC CICS GET command to request the next number.

There are various uses for this facility, such as obtaining a unique number for documents (for example, customer orders, invoices, and despatch notes), or for obtaining a block of numbers for allocating customer record numbers in a customer file.

In a single CICS region, there are various methods you can use to control the allocation of a unique number. For example, you could use the CICS common work area (CWA) to store a number that is updated by each application program that uses the number. The problem with the CWA method is that the CWA is unique to the CICS address space, and cannot be shared by other regions that are running the same application. A CICS shared data table could be used to provide such a service, but the CICS regions would all have to reside in the same MVS image. The named counter facility overcomes all the sharing difficulties presented by other methods by maintaining its named counters in the coupling facility, and providing access through a named counter server running in each MVS image in the sysplex. This ensures that all CICS regions throughout the Parallel Sysplex have access to the same named counters.

When you use a named counter server, each normal request (to assign the next counter value) only requires a single coupling facility access. This provides a significant improvement in performance compared to the use of files for this purpose. The named counter server also performs better than coupling facility data tables in this respect, because at least two coupling facility accesses are required to update a coupling facility data table. Depending on your hardware configuration, you should easily be able to make many thousands of named counter server requests each second.

This section describes:

- "The named counter fields" on page 331
- "Named counter pools" on page 331
- "Using the named counter EXEC interface" on page 332
- "Using the named counter CALL interface" on page 334
- "Named counter recovery" on page 344

The named counter fields

Each named counter consists of the counter name, the current value, the minimum value, and the maximum value.

The counter name
The name can be up to 16 bytes, comprising the characters A through Z, 0 through 9, $ @ # and _. Names less than 16 bytes should be padded with trailing blanks.

The current value
The next number to be assigned to a requesting application program.

The minimum value
Specifies the minimum number for a counter, and the number to which a counter is reset by the server in response to a REWIND command.

The maximum value
Specifies the maximum number that can be assigned by a counter, after which the counter must be explicitly reset by a REWIND command (or automatically by the WRAP option).

All values are stored internally as 8-byte (doubleword) binary numbers. The EXEC CICS interface allows you to use them as either fullword signed binary numbers or doubleword unsigned binary numbers. This can give rise to overflow conditions if you define a named counter using the doubleword command (see "Using the named counter EXEC interface" on page 332) and request numbers from the server using the signed fullword version of the command.

Named counter pools

A named counter is stored in a named counter pool, which resides in a list structure in a coupling facility. Each pool, even if its list structure is defined with the minimum size of 256 KB, can hold up to a thousand named counters.

You create a named counter pool by defining the coupling facility list structure for the pool, and then starting the first named counter server for the pool. Pool names are of 1 to 8 bytes from the same character set for counter names. Although pool names can be made up from any of the allowed characters, names of the form DFHNC *xxx* are recommended.

You can create different pools to suit your needs. You could create a pool for use by production CICS regions (for example, called DFHNCPRD), and others for test and development regions (for example, using names like DFHNCTST and DFHNCDEV). See "Named counter options table" on page 331 for information about how you can use logical pool names in your application programs, and how these are resolved to actual pool names at runtime.

Defining a list structure for a named counter server, and starting a named counter server, is explained in Setting up and running a named counter server.

Named counter options table

The POOL(*name*) parameter is optional on all the EXEC CICS COUNTER and DCOUNTER commands. If you specify the POOL parameter, it can refer to either an actual or a logical pool name. Whether you specify a POOL parameter or omit it, CICS resolves the actual pool name by reference to the named counter options table, which is loaded from the link list.

For more information about EXEC CICS COUNTER and DCOUNTER commands see "Using the named counter EXEC interface" on page 332.

The named counter options table, DFHNCOPT, provides several methods for determining the actual pool name referenced by a named counter API command, all of which are described in the Setting up shared data sets, CSD and SYSIN . This also describes the DFHNCO macro that you can use to create your own options table.

The POOLSEL parameter in the default options table works with the POOL(*name*) option on the API. The default options table is supplied in source and object form. The pregenerated version is in *hlq* .SDFHLINK, and the source version, which is supplied in the *hlq* .SDFHSAMP library (where *hlq* represents the high-level qualifier for the library names, established at CICS installation time), contains the following entries:

```
 DFHNCO POOLSEL=DFHNC*,POOL=YES
DFHNCO POOL=
END DFHNCOPT
```

The default options table entries work as follows:

POOLSEL=DFHNC*
: This pool selection parameter defines a generic logical pool name beginning with the letters DFHNC. If any named counter API request specifies a pool name that matches this generic name, the pool name is determined by the POOL= operand in the DFHNCO entry. Because this is POOL=YES in the default table, the name passed on the POOL(*name*) option of the API command is taken to be an actual name. Thus, the default options table specifies that all logical pool names beginning with DFHNC are actual pool names.

POOL=
: This entry in the default table is the 'default' entry. Because the POOLSEL parameter is not specified, it defaults to POOLSEL=*, which means it is taken to match any value on a POOL parameter that does not find a more explicit match. Thus, any named counter API request that:

  - Specifies a POOL value that begins with other than DFHNC, or
  - Omits the POOL name parameter altogether

  is mapped to the default pool (indicated by a POOL= options table parameter that omits a name operand).

  You can specify the default pool name to be used by a CICS region by specifying the NCPLDFT system initialization parameter. If NCPLDFT is omitted, the pool name defaults to DFHNC001.

You can see that you do not need to create you own options table, and named counter API commands do not need to specify the POOL option, if:

- You use pool names of the form DFHNC *xxx* , or
- Your CICS region uses only one pool that can be defined by the NCPLDFT system initialization parameter.

Note:

1. DFHNCOPT named counter options tables are not suffixed. A CICS region loads the first table found in the MVS link list.
2. There must be a named counter server running, in the same MVS image as your CICS region, for each named counter pool used by your application programs.

Using the named counter EXEC interface

Although all named counter values are held internally as doubleword unsigned binary numbers, the CICS API provides both a fullword (COUNTER) and doubleword (DCOUNTER) set of commands, which you should not mix.

You can use EXEC CICS commands to perform the following operations on named counters:

DEFINE
: Defines a new named counter, setting minimum and maximum values, and specifying the current number at which the counter is to start.

DELETE
Deletes a named counter from its named counter pool.

GET
Gets the current number from the named counter, provided the maximum number has not already been allocated.

Using the WRAP option : If the maximum number has been allocated to a previous request, the counter is in a counter-at-limit condition and the request fails, unless you specify the WRAP option. This option specifies that a counter in the the counter-at-limit condition is to be reset automatically to its defined minimum value. After the reset, the minimum value is returned as the current number, and the counter is updated ready for the next request.

Using the INCREMENT option : By default, a named counter is updated by an increment of 1, after the server has assigned the current number to a GET request. If you want more than one number at a time, you can specify the INCREMENT option, which effectively reserves a block of numbers from the current number. For example , if you specify INCREMENT(50), and the server returns 100 025:

- Your application program can use 100 025 through 100 074
- As a result of updating the current number (100 025) by 50, the current number is then 100 075 ready for the next request.

This example assumes that updating the current value by the INCREMENT(50) option does not exceed the maximum value by more than 1. If the range of numbers between the current value and the maximum value plus 1 is less than the specified increment, the request fails unless you also specify the REDUCE option.

Using the REDUCE option : To ensure that a request does not fail because the remaining range of numbers is too small to satisfy your INCREMENT value (the current number is too near the maximum value), specify the REDUCE option. With the reduce option, the server automatically adjusts the increment to allow it to assign all the remaining numbers, leaving the counter in the counter-at-limit condition.

Using both the WRAP and REDUCE options : If you specify both options, only one is effective depending on the state of the counter:

- If the counter is already at its limit when the server receives the GET request, the REDUCE option has no effect and the WRAP option is obeyed.
- If the counter is not quite at its limit when the server receives the GET request, but the remaining range is too small for increment, the REDUCE option is obeyed and the WRAP option has no effect.

Using the COMPAREMIN and COMPAREMAX options : You can use these options to make the named counter GET (and UPDATE) operation conditional upon the current number being within a specified range, or being greater than, or less than, one of the specified comparison values.

QUERY
Queries the named counter to obtain the current, minimum, and maximum values. Note that you cannot use more than one named counter command in a way that is atomic, and you cannot rely on the information returned on a QUERY command not having been changed by another task somewhere in the sysplex. Even the CICS sysplex-wide ENQ facility cannot lock a counter for you, because a named counter could be accessed by a batch application program using the named counter CALL interface. If you want to make an operation conditional upon the current value being within a certain range, or greater than, or less than, a certain number, use the COMPAREMIN and COMPAREMAX parameters on your request.

REWIND
Rewinds a named counter that is in the counter-at-limit condition back to its defined minimum value.

UPDATE
Updates the current value of a named counter to a new current value. For example, you could set the current value to the next free key in a database. Like the GET command, this can be made conditional by specifying COMPAREMIN and COMPAREMAX values.

Using the named counter CALL interface

In addition to the CICS named counter API, CICS provides a call interface that you can use from a batch application to access the same named counters. This could be important where you have an application that uses both CICS and batch programs, and both must access the same named counter to obtain unique numbers from a specified range. The call interface does not depend on CICS services; therefore, it can also be used in applications running under any release of CICS.

The named counter call interface does not use CICS command-level API; therefore, the system initialization parameter CMDPROT=YES has no effect. If the interface is called from a CICS application program that is executing in user key, it switches to CICS key while processing the request but CICS does not attempt to verify that the program has write access to the output parameter fields.

The first request by an application region that addresses a particular pool automatically establishes a connection to the server for that pool. This connection is associated with the current z/OS TCB (which for CICS is the quasi-reentrant (QR) TCB) and normally lasts until the TCB terminates at end of job. This connection can only be used from the TCB under which the connection was established. A request issued from another TCB will establish a separate connection to the server.

Note: The named counter server interface uses z/OS name/token services internally. A consequence of this is that jobs using the named counter interface cannot use z/OS checkpoint/restart services.

Programming considerations

1. Ensure that your application programs include the appropriate copybook that defines the parameter list definition for the application programming language. The copybook defines symbolic constants for the function codes and return codes, and also defines the callable entry point for high-level languages. The copybook name is of the form DFHNC *xxx* where *xxx* indicates the programming language, as follows:
   - ASM or EQU for Assembler
   - C for C/C++
   - COB for COBOL
   - PLI for PL/I
2. Ensure that the application program is link-edited with the callable interface linkage routine, DFHNCTR.
3. Ensure the named counter server interface module, DFHNCIF, and the options table, DFHNCOPT, are available to the CICS region. That is, these objects must be in a STEPLIB library, in a linklist library, or in the LPA. To support CICS application programs that run in user key, DFHNCIF must be loaded from an APF-authorized library. The default option table and the named counter server interface module are supplied in CICSTS56.CICS.SDFHLINK.

Copybooks

CICS provides copybooks for all the supported languages. The copybook defines symbolic constants for the function codes and return codes, and also defines the callable entry point for high-level languages.

Assembler

Copybooks: DFHNCASM, DFHNCEQU

The standard assembler named counter interface definitions are provided in copybook DFHNCASM. Include these in your application programs using COPY DFHNCASM within a constant CSECT area. The symbolic values are defined as static fullword constants, in the form NC_ *name* DC F' *nnn* '.

Example

```
NC_BROWSE_NEXT DC F'7'
```

An alternative set of definitions is provided as symbolic equated values in copybook DFHNCEQU. These symbols are all of the form NC_EQU_ *name* to avoid conflict with the static constants. Note that

when these equated values are used for function codes or return code comparisons, they should be used as address constant values, so that for example the function code NC_ASSIGN can be replaced by a reference to =A(NC_EQU_ASSIGN).

Note that the CALL macro must specify the VL option to set the end of list indication.

Example

```
CALL DFHNCTR,(NC_ASSIGN,RC,POOL,NAME,CTRLEN,CTR),VL
```

C/C++
The named counter interface definitions for C/C++ are provided in header file DFHNCC. The symbolic constant names are in uppercase. The function name is `dfhnctr` in lowercase.

COBOL
Copybook: DFHNCCOB

The named counter interface definitions for COBOL are provided in copybook DFHNCCOB.

Earlier versions of COBOL that are supported by CICS do not allow underscores in names. The symbolic names provided in copybook DFHNCCOB therefore use a hyphen instead of an underscore (for example NC-ASSIGN and NC-COUNTER-AT-LIMIT).

Note that the RETURN-CODE special register is set by each call, which affects the program overall return code if it is not explicitly set again before the program terminates.

PL/I
The named counter interface definitions for PL/I are provided in include file DFHNCPLI.

The named counter call interface, DFHNCTR
Applications can issue DFHNCTR calls to access named counter services.

Syntax

For illustration purposes, Figure 99 on page 335 shows you the syntax of the PL/I version of the DFHNCTR call. Refer to the copybook relevant to Assembler, C/C++, COBOL and PL/I for language-specific call syntax, parameters and programming information.

```
CALL DFHNCTR(
function,return_code,pool_selector,counter_name,

value_length,current_value,minimum_value,maximum_value,

counter_options,update_value,compare_min,compare_max
);
```

Figure 99. DFHNCTR call syntax for PL/I

Note:

1. All functions that refer to a named counter require at least the first four parameters, but the remaining parameters are optional, and trailing unused parameters can be omitted.

   If you do not want to use an embedded optional parameter, either specify the default value or ensure that the parameter list contains a null address for the omitted parameter. For an example of a call that omits an optional parameter, see “Example of DFHNCTR calls with null parameters” on page 343.
2. The NC_FINISH function requires the first three parameters only.

Parameters

function

Specifies the function to be performed, as a 32-bit integer, using one of the following symbolic constants.

NC_CREATE

Create a new named counter, using the initial value, range limits, and default options specified on the *current_value, minimum_value, maximum_value, update_value* and *counter_options* parameters.

If you omit an optional value parameter, the new named counter is created using the default for the omitted value. For example, if you omit all the optional parameters, the counter is created with an initial value of 0, a minimum value of 0, and a maximum value of high values (the double word field is filled with X'FF').

NC_ASSIGN

Assign the current value of the named counter, and then increment it ready for the next request. When the number assigned equals the maximum number specified for the counter, it is incremented finally to a value 1 greater than the maximum. This ensures that any subsequent NC_ASSIGN requests for the named counter fail (with NC_COUNTER_AT_LIMIT) until the counter is reset using the NC_REWIND function, or automatically rewound by the NC_WRAP counter option (see the *counter_option*s parameter).

This operation can include a conditional test on the current value of the named counter, using the *compare_min* and *compare_max* parameters.

The server returns the minimum and maximum values if you specify these fields on the call parameter list and the request is successful.

You can use the *counter_options* parameter on the NC_ASSIGN request to override the counter options set by the NC_CREATE request.

You can use the *update_value* parameter to specify the increment to be used on this request only for the named counter (the default increment is 1). This enables you to obtain a range of numbers on a single request. For more information, see the description of the *update_value* parameter.

Note that the named counter is incremented by *update_value* after the current value is assigned. For example, if the current value is 109 and *update_value* specifies 25, the named counter server returns 109 and sets the current value to 134 ready for the next NC_ASSIGN request, effectively assigning numbers in the range 109 through 133. The increment can be any value between zero and the limit is determined by the minimum and maximum values set for the named counter. Thus the increment limit is ((*maximum_value* plus 1) minus *minimum value*). An increment of zero causes NC_ASSIGN to operate the same as NC_INQUIRE, except for any comparison options.

When the increment is greater than 1, and the named counter is near the maximum limit, the server may not be able to increment the current number by the whole value of the specified increment. This situation occurs when incrementing the counter would exceed the maximum value plus 1. You control what action the named counter server takes in this situation through the *counter_options* NC_NOREDUCE | NC_REDUCE, and NC_NOWRAP | NC_WRAP. See *counter_option*s parameter for information about the operation of these options.

NC_BROWSE_FIRST

Return the details of the first named counter with a name greater than or equal to the specified name, and update the *counter_name* field accordingly.

NC_BROWSE_NEXT

Return the details of the next named counter with a name greater than the specified name, and update the *counter_name* field accordingly.

NC_DELETE

Delete the specified named counter. The server returns the *current_value*, *minimum_value*, *maximum_value*, and *counter_options* if you specify these fields on the parameter list and the request is successful.

NC_FINISH
Terminate the connection between the current z/OS task (TCB) and the named counter server for the specified pool. If a further request is made to the same pool, a new connection is established.

This function does not apply to a specific counter, therefore the only parameters required are the function, the return code and the pool name.

Use this function only when there is a special reason to terminate the connection (for example, to allow the server to be shut down).

NC_INQUIRE
Return the details (*current_value, minimum_value, maximum_value* and *counter_options*) of a named counter without modifying it. The current value is the value to be returned on the next NC_ASSIGN call. If the maximum value of the named counter has already been assigned, the server returns a current value one greater than the maximum value.

NC_REWIND
Reset the named counter to its minimum value. This function is valid only when the last number permitted by the maximum value has been assigned, leaving the counter in the NC_COUNTER_AT_LIMIT condition. If an NC_ASSIGN call causes the server to assign the last number for the named counter, use the NC_REWIND function to reset the counter.

This operation can include a conditional test on the current value of the named counter, using the compare_min and compare_max parameters.

The server returns the new current value, minimum value, and maximum value if you specify these fields on the parameter list and the request is successful.

If any option parameter or *update_value* parameter was specified on an NC_ASSIGN request which failed because the named counter was at its limit, the same parameter values must also be specified on the NC_REWIND request, so that it can check whether the original NC_ASSIGN would still fail. The NC_REWIND request is suppressed with return code 102 (NC_COUNTER_NOT_AT_LIMIT) whenever the corresponding NC_ASSIGN request would succeed.

If the NC_WRAP option is in effect, or the *update_value* parameter is zero, NC_REWIND is suppressed because NC_ASSIGN always succeeds with these conditions. See the *counter_options* parameter for information about the NC_WRAP option.

NC_UPDATE
Set the named counter to a new value. This operation can include a conditional test on the current value of the named counter, using the *compare_min* and *compare_max* parameters.

You specify the new value on the *update_value* parameter. If you don't specify a new value, the named counter remains unchanged.

You can specify a valid *counter_options* override parameter (or a null address) with this function, but counter options have no effect. Specify either a null address or NC_NONE as the *counter_options* parameter.

return_code
Specifies a 32-bit integer field to receive the return code. The same information is also returned in register 15, which for COBOL callers is stored in the RETURN-CODE special register.

Each return code has a corresponding symbolic constant. See “Return codes” on page 341 for details.

pool_selector
Specifies an 8-character pool selection parameter that you use to identify the pool in which the named counter resides.

This parameter is optional. If you omit the pool selection parameter, a string of 8 blank X'40') characters is assumed.

Acceptable characters:

A-Z 0-9 $ @ # _

The first character cannot be numeric or an underscore. The parameter should be padded to fill 8 characters with trailing spaces as necessary. The parameter can be all spaces to use the default pool for the current region, provided this is mapped by the options table to a valid non-blank named counter name).

Depending on the named counter options table in use, you can use the pool selector parameter either as an actual pool name, or as a logical pool name that is mapped to a real pool name through the options table. The default options table assumes:

- That any pool selection parameter beginning with DFHNC (matching the table entry with POOLSEL=DFHNC*) is an actual pool name
- That any other pool selection parameter (including all blanks) maps to the default pool name.

Note: The default pool name for the call interface is DFHNC001. The default pool name for the **EXEC CICS** API is defined by the **NCPLDFT** system initialization parameter.

See "Named counter options table" on page 331 for information about the pool selection parameter in the DFHNCOPT options table.

counter_name
Specifies a 16-byte field containing the name of the named counter, padded if necessary with trailing spaces.

Acceptable characters:

A-Z 0-9 $ @ # _

The first character cannot be numeric or an underscore. The parameter should be padded to fill 16 characters with trailing spaces as necessary.

You are recommended to use names that have a common prefix of up to 8 bytes to avoid conflicts with other applications. Any named counters used internally by CICS have names starting with DFH.

For the NC_BROWSE_FIRST and NC_BROWSE_NEXT functions, the actual name is returned in this field, which must be a variable for this purpose. For all other functions, this can be a constant.

value_length
Specifies a 32-bit integer field indicating the length of each named counter value field. Values can be in unsigned format (where the high order bit is part of the value), or in positive signed format (where the high order bit is reserved for a zero sign bit). To use unsigned format, specify the length in bytes, in the range 1 to 8, corresponding to 8 to 64 bits. To use values in signed format, specify the length in bytes as a negative number in the range –1 to –8, corresponding to 7 to 63 bits. For compatibility with the named counter EXEC interface, this should be set to –4 for counters that are handled as fullword signed binary values (COUNTER) and 8 for counters that are handled as doubleword unsigned binary values (DCOUNTER).

If no value parameters are used on the call, you can either omit all the trailing unused parameters completely, including the value length, or specify *value_length* as 0.

When input values are shorter than 8 bytes, they are extended with high-order zero bytes to the full 8 bytes used internally. When output values are returned in a short value field, the specified number of low-order bytes are returned, ignoring any higher-order bytes. However, if the value is too large to be represented correctly within the field, then a warning return code might be set, as described in "Checking for result overflow" on page 343.

current_value
Specifies a variable to be used for:

- Setting the initial sequence number for the named counter
- Receiving the current sequence number from the named counter.

For the NC_CREATE function this parameter is an input (sender) field and can be defined as a constant. The default value is low values (binary zeroes). The value can either be within the range specified by the following minimum and maximum values, or it can be one greater than the maximum value, in which case the counter has to be reset using the NC_REWIND function before it is used. No

sign check is made for this field, but a value that has the sign bit set would normally be rejected by the server as being inconsistent with the counter limits. For a counter that has a range consisting of all signed numbers, the counter at limit value does have the sign bit set, and this can be used as a valid input value.

For all other counter functions, this parameter is an output (receiver) field and must be defined as a variable.

minimum_value
Specifies a variable to be used for:

- Setting the minimum value for the named counter
- Receiving from the named counter the specified minimum value.

For the NC_CREATE function this parameter is an input (sender) field and can be defined as a constant. The default value is low values (binary zeroes).

For all other functions, this parameter is an output (receiver) field and must be defined as a variable.

maximum_value
Specifies a variable to be used for:

- Setting the maximum value for the named counter
- Receiving from the named counter the specified maximum value

For the NC_CREATE function, this parameter is an input (sender) field and can be defined as a constant. If you specify a non-zero *value_length* parameter but omit *maximum_value*, then *maximum_value* defaults to high values (or, for signed variables, the largest positive value) for the specified length. If the *value_length* parameter is omitted or is specified as zero, then *maximum_value* defaults to eight bytes of high values. However, if the minimum value is all low values and the maximum value is eight bytes of high values, the maximum value is reduced to allow some reserved values to be available to the server for internal use.

Note:

For DCOUNTER, two doubleword counter values are reserved for internal use by the named counter server, one for the overflow value and the other for a value that is used to guarantee a unique counter value and prevent duplicate counters. This means that if the range from the minimum value to the maximum value (as specified or assumed by default) does not leave two doubleword values unused, as in the cases detailed below, the maximum value is automatically reduced by one or two as necessary to reserve the necessary values for internal use.

This affects only the following cases:

- For a minimum value of `X'0000000000000000'` (as assumed by default) and a maximum value of either `X'FFFFFFFFFFFFFFFF'` (as assumed by default) or `X'FFFFFFFFFFFFFFFE'`, the maximum value is reduced to `X'FFFFFFFFFFFFFFFD'`.
- For a minimum value of `X'0000000000000001'` and a maximum value of `X'FFFFFFFFFFFFFFFF'`, the maximum value is reduced to `X'FFFFFFFFFFFFFFFE'`.

For all other functions, this parameter is an output (receiver) field and must be defined as a variable.

counter_options
Specifies an optional fullword field to indicate named counter options that control wrapping and increment reducing. The valid options are represented by the symbolic values NC_WRAP or NC_NOWRAP and NC_REDUCE or NC_NOREDUCE. The default options are NC_NOWRAP and NC_NOREDUCE.

NC_NOWRAP
The server does not automatically rewind the named counter back to the minimum value in response to an NC_ASSIGN request that fails with the NC_COUNTER_AT_LIMIT condition. With NC_NOWRAP in force, and the named counter in the NC_COUNTER_AT_LIMIT condition, the NC_ASSIGN function is inoperative until the counter is reset by an NC_REWIND request (or the counter option reset to NC_WRAP).

NC_WRAP
The server automatically performs an NC_REWIND in response an NC_ASSIGN request for a counter that is in the NC_COUNTER_AT_LIMIT condition. The server sets the current value of the named counter equal to the minimum value, returns the new current value to the caller, and increments the named counter.

NC_NOREDUCE
If the range of numbers remaining to be assigned (the difference between the current value and the maximum value plus 1) is less than the increment specified on the *update_value* parameter, the assign fails (unless NC_WRAP is in force). NC_NOREDUCE, with NC_NOWRAP, means the NC_ASSIGN request fails with the NC_COUNTER_AT_LIMIT condition.

For example, if a request specifies an update value of 15 when the current number is 199 990 and the counter maximum number is defined as 199 999, the NC_REQUEST fails because the increment would cause the current number to exceed 200 000.

NC_REDUCE
If the range of numbers remaining to be assigned (the difference between the current value and the maximum value plus 1) is less than the increment specified on the *update_value* parameter, the increment is reduced and the assign succeeds. In this case, the NC_ASSIGN request has been assigned a range of numbers less than that specified by the *update_value* , and the named counter is remains in the NC_COUNTER_AT_LIMIT condition. Subsequent NC_ASSIGN requests will fail until the named counter is reset with an NC_REWIND request.

The options specified on NC_CREATE are stored with the named counter and are used as the defaults for other named counter functions. You can override the options on NC_ASSIGN, NC_REWIND or NC_UPDATE requests. If you don't want to specify *counter_options* on a DFHNCTR call, specify the symbolic constant NC_NONE (equal to zero) as the input parameter (or specify a null address).

For the NC_CREATE, NC_ASSIGN, NC_REWIND, and NC_UPDATE functions, this parameter is an input field.

For the NC_DELETE, NC_INQUIRE, and NC_BROWSE functions, this parameter is an output field, which returns the options specified on NC_CREATE.

update_value
Specifies the value to be used to update the counter. For NC_ASSIGN, this is the increment to be added to the current counter value (after the current number is assigned). See the NC_ASSIGN option on the *function* parameter for information on how specifying an increment other than 1 can affect an assign operation.

For NC_UPDATE, this is the new current value for the named counter.

compare_min
Specifies a value to be compared with the named counter's current value. If you specify a value, this parameter makes the NC_ASSIGN, NC_REWIND or NC_UPDATE operation conditional on the current value of the named counter being greater than or equal to the specified value. If the comparison is not satisfied, the operation is rejected with a counter-out-of-range return code (RC 103).

If you omit this parameter by specifying a null address, the server does not perform the comparison.

compare_max
Specifies a value to be compared with the named counter's current value. If you specify a value, this parameter makes the NC_ASSIGN, NC_REWIND or NC_UPDATE operation conditional on the current value of the named counter being less than or equal to the specified value. If the comparison is not satisfied, the operation is rejected with a counter-out-of-range return code (RC 103).

If you specify high values (X'FF') for this parameter, the server does not perform the comparison. You must specify (X'FF') in all the bytes specified by the *value_length* parameter.

If the compare_max value is less than the compare_min value, the valid range is assumed to wrap round, in which case the current value is considered to be in range if it satisfies either comparison, otherwise both comparisons must be satisfied.

Return codes

The named counter call interface has three warning return codes (1 - 3), which indicate result overflows for a request that otherwise completed normally. If more than one warning return code applies for the same request, the highest applicable warning return code is set.

The remaining return codes are divided into ranges (100, 200, 300, and 400) according to their severity. Each range of non-zero return codes begins with a dummy return code that describes the return code category, to make it easy to check for values in each range using a symbolic name.

In the following list, the numeric return code is followed by its symbolic name.

0 (NC_OK)
The request completed normally.

1 (NC_RESULT_OVERFLOW)
The result value overflowed into the sign bit.

2 (NC_RESULT_CARRY)
The result value overflowed, and the leading part was exactly equal to 1.

3 (NC_RESULT_TRUNCATED)
The result value overflowed, and the leading part was greater than 1.

100 (NC_COND)
Return codes in this range indicate that a conditional function did not succeed because the condition was not satisfied:

101 (NC_COUNTER_AT_LIMIT)
An NC_ASSIGN function is rejected because the previous request for this named counter obtained the maximum value and the counter is now at its limit. New counter values cannot be assigned until an NC_REWIND function call is issued to reset the counter.

102 (NC_COUNTER_NOT_AT_LIMIT)
An NC_REWIND FUNCTION is rejected because the named counter is not at its limit value. This is most likely to occur when another task has already succeeded in resetting the counter with an NC_REWIND.

103 (NC_COUNTER_OUT_OF_RANGE)
The current value of the named counter is not within the range specified on the *compare_min* and *compare_max* parameters.

200 (NC_EXCEPTION)
Return codes in this range indicate an exception condition that an application program can handle:

201 (NC_COUNTER_NOT_FOUND)
The named counter cannot be found.

202 (NC_DUPLICATE_COUNTER_NAME)
An NC_CREATE function is rejected because a named counter with the specified name already exists.

203 (NC_SERVER_NOT_CONNECTED)
An NC_FINISH function is rejected because no active connection exists for the selected pool.

300 (NC_ENVIRONMENT_ERROR)
Return codes in this range indicate an environment error. These are serious errors, normally caused by some external factor, that a program might not to be able to handle.

301 (NC_UNKNOWN_ERROR)
The server has reported an error code that is not understood by the interface. Generally, this is not possible unless the interface load module, DFHNCIF, is at a lower maintenance or release level than the server itself.

302 (NC_NO_SPACE_IN_POOL)
A new counter cannot be created because there is insufficient space in the named counter pool.

303 (NC_CF_ACCESS_ERROR)
An unexpected error, such as structure failure or loss of connectivity, has occurred on a macro used to access the coupling facility. Further information can be found in message DFHNC0441 in the application job log.

304 (NC_NO_SERVER_SELECTED)
The pool selection parameter specified in the program cannot be resolved to a valid server name using the current options table.

305 (NC_SERVER_NOT_AVAILABLE)
The interface cannot establish a connection to the server for the appropriate named counter pool. Further information can be found in an AXM services message in the application job log.

306 (NC_SERVER_REQUEST_FAILED)
An abend occurred during server processing of a request. Further information can be found in a message in the aplication job log and the server job log.

307 (NC_NAME_TOKEN_ERROR)
An IEANT *xx* name/token service call in the named counter interface module gave an unexpected return code.

308 (NC_OPTION_TABLE_NOT_FOUND)
The DFHNCOPT options table module, required for resolving a pool name, cannot be loaded.

309 (NC_OPTION_TABLE_INVALID)
During processing of the options table, the named counter interface encountered an unknown entry format. Either the options table is not correctly generated, or the DFHNCIF interface load module is not at the same release level as the options table.

310 (NC_USER_EXIT_NOT_FOUND)
An options table entry matching the given pool name specified a user exit program, but the user exit program is not link-edited with the options table and cannot be loaded.

311 (NC_STRUCTURE_UNAVAILABLE)
The named counter server list structure is temporarily unavailable. For example, one reason for this situation is that a z/OS system-managed rebuild is in progress.

Note: The **EXEC CICS** interface to the named counter uses the CALL interface internally, but it hides this return code from the application program by waiting for one second and retrying the request. The **EXEC CICS** interface continues this wait and retries until it succeeds, with the result that the application program has only a time delay, not an error response. You can use the same technique in your application programs that use the CALL interface.

400 (NC_PARAMETER_ERROR)
Return codes in this range indicate a parameter error, generally the result of a coding error in the calling program.

401 (NC_INVALID_PARAMETER_LIST)
The parameter list is invalid for one of the following reasons:

- Too few parameters are specified (less than four, or less than three for the NC_FINISH function)
- Too many parameters are given (more than eight)
- A parameter address is zero
- The end-of-list marker is missing.

402 (NC_INVALID_FUNCTION)
The function code parameter is not in the supported range.

403 (NC_INVALID_POOL_NAME)
The pool selection parameter contains characters that are not allowed, or embedded spaces.

404 (NC_INVALID_COUNTER_NAME)
The *counter_name* parameter contains characters that are not allowed, or embedded spaces.

405 (NC_INVALID_VALUE_LENGTH)
The value length parameter is not in the range 0 to 8.

406 (NC_INVALID_COUNTER_VALUE)
The specified counter value or increment value is inconsistent with the minimum and maximum limits for the counter.

The counter value specified on the *current_value* parameter for the NC_CREATE function, or the *update_value* for the NC_UPDATE function, cannot be less than the specified minimum value, and cannot be more than (maximum value + 1).

The increment value specified in the *update_value* parameter for the NC_ASSIGN or NC_REWIND function cannot be greater than the total range of the counter ((maximum value - minimum value) + 1).

407 (NC_INVALID_COUNTER_LIMIT)
The maximum value is less than the minimum value.

408 (NC_INVALID_OPTIONS)
The value of the *counter_options* parameter is invalid. It is either a value that does not correspond to any defined option, or it is a value that represents some mutually exclusive options.

Checking for result overflow
The call interface checks for results which do not fit into the specified size of result field, or which overflow into the sign bit when signed variables are used.

If a result field (*counter_value* , *minimum_value* or *maximum_value*) has been defined as a signed variable by specifying *value_length* as a negative value, the call interface checks for results that overflow into the sign bit. In this case, the operation completes normally but the return code NC_RESULT_OVERFLOW is set. As a special case, a result value for a counter which is at its limit value is not checked for this form of overflow, to avoid setting the return code unnecessarily. This means that if a query is made to a counter which is at its limit, and whose maximum value is the maximum positive value, a negative number might be returned as the current counter value without causing this return code.

If a result field (*counter_value* , *minimum_value* or *maximum_value*) is too short to contain the full non-zero part of the result, the operation completes normally, but one of the following return codes is set:

- NC_RESULT_CARRY, if the leading part is exactly equal to 1.
- NC_RESULT_TRUNCATED, if the leading part is greater than 1.

If a 4–byte unsigned counter that has a maximum of high values has reached its limit value, the return code NC_RESULT_CARRY is set, and the counter value is zero.

Example of DFHNCTR calls with null parameters
If you omit an optional parameter on a DFHNCTR call, ensure that the parameter list is built with a null address for the missing parameter. The example that follows illustrates how to issue, from a COBOL program, an NC_CREATE request with some parameters set to null addresses.

DFHNCTR call with null addresses for omitted parameters: In this example, the parameters that are used on the call are defined in the WORKING-STORAGE SECTION, as follows:

Call parameter	COBOL variable	Field definition
function	01 NC-FUNCTION	PIC S9(8) COMP VALUE +1.
return_code	01 NC-RETURN-CODE.	PIC S9(8) COMP VALUE +0.
pool_selector	01 NC-POOL-SELECTOR	PIC X(8).
counter_name	01 NC-COUNTER-NAME	PIC X(16).
value_length	01 NC-VALUE-LENGTH	PIC S9(8) COMP VALUE +4.
current_value	01 NC-CURRENT-VALUE	PIC S9(8) VALUE +0.
minimum_value	01 NC-MIN-VALUE	PIC S9(8) VALUE +0.
maximum_value	01 NC-MAX-VALUE	PIC S9(8) VALUE -1.
counter_options	01 NC-OPTIONS	PIC S9(8) COMP VALUE +0.

Call parameter	COBOL variable	Field definition
update_value	01 NC-UPDATE-VALUE	PIC S9(8) VALUE +1.
compare_min	01 NC-COMP-MIN	PIC S9(8) VALUE +0.
compare_max	01 NC-COMP-MAX	PIC S9(8) VALUE +0.

The variable that is used for the null address is defined in the LINKAGE SECTION, as follows:

```
LINKAGE SECTION.
        01 NULL-PTR USAGE IS POINTER.
```

Using the data names that are specified in the WORKING-STORAGE SECTION as described, and the NULL-PTR name as described in the LINKAGE SECTION, the following example illustrates a call to a named counter-server where *value_length, current_value, minimum_value* and *counter_options* are the only optional parameters specified. The others are allowed to default, or, in the case of trailing optional parameters, omitted altogether.

```
 NAMED-COUNTER SECTION.
*
SET ADDRESS OF NULL-PTR TO NULLS.
*
MOVE 1 TO NC-FUNCTION.
MOVE 100 TO NC-MIN-VALUE NC-CURRENT-VALUE.
MOVE NC-WRAP TO NC-OPTIONS.
MOVE "DFHNC001" TO NC-POOL-SELECTOR.
MOVE "CUSTOMER_NUMBER" TO NC-COUNTER-NAME.
CALL 'DFHNCTR' USING NC-FUNCTION NC-RETURN-CODE NC-POOL-SELECTOR
NC-COUNTER-NAME NC-VALUE-LENGTH NC-CURRENT-VALUE
NC-MIN-VALUE NULL-PTR NC-OPTIONS.
```

If you want to use DCOUNTER instead of COUNTER with DFHNCTR, then you need to change the NC-CURRENT-VALUE, and the NC-MIN-VALUE and NC-MAX-VALUE definitions from signed decimal values (as previously listed) to unsigned doubleword binary definitions, for example PIC 9(16) COMP. Also, MOVE 8 to NC-VALUE-LENGTH.

Named counter recovery

Named counters are only stored in the coupling facility. Applications using named counters might therefore need to implement recovery logic to handle the possible impact of any coupling facility problems.

If the coupling facility or list structure for the named counter pool fails, but another facility is available, it should normally be possible to re-create the named counter pool's list structure very quickly on another facility. The original instance of the server terminates as soon as it detects the problem, and a new instance is normally started immediately by ARM, if it is available, and the installation policy allows the restart. If the pool's list structure is known to have failed, the new server should be able to allocate a new instance immediately, and the pool should be available again within seconds.

However, in some situations, such as a coupling facility power failure, MVS might initially perceive the situation as a loss of connectivity, and be unable to determine that the list structure has failed until the original facility has been restarted. In such situations, recovery can be speeded up by issuing an operator command to force deletion of the existing structure, allowing a new instance to be allocated immediately.

Until the new structure has been created, attempts to obtain a counter value are rejected because the server is unavailable. This means that applications issuing such requests are unavailable while the new structure is being created, unless they have an alternative method of assigning numbers.

If the list structure for a named counter pool is re-created because of a failure, it is empty, and applications will immediately discover that their named counters are no longer found. The standard recovery technique in this situation is as follows:

1. Make the application issue an enqueue command (ENQ) for a resource based on the counter name. This ensures that only one task will be trying to repair each named counter.
2. Check to see if another task has already re-created the named counter.

3. If the named counter has not been re-created, re-create it with an appropriate value that you have reconstructed from other information, using the following methods described.
4. Issue a dequeue command (DEQ) for the resource, to release the named counter.

The enqueue and dequeue process is used in order to avoid multiple tasks wasting time and resources duplicating the same processing. However, if the process that you use to re-create the named counter is simple enough, you can omit the enqueue and dequeue process. If another task has already repaired the named counter, any subsequent attempt to re-create the counter will be rejected with a duplicate counter name indication.

If a named counter is only being used for temporary information that does not need to be preserved across a structure failure (for example, unique identifiers for active processes that would not be expected to survive such a failure), then recovery techniques can be minimal. For example, you could re-create the counter with a standard initial value.

However, if a named counter is being used for persistent information, such as an order number, recreating it may require specific application logic to reconstruct the counter value. For example, the application could locate the highest key that is currently used in the order file. If active transactions might have already acquired new numbers from the counter, but not yet used them, then you should allow for this in the recovery process. Two methods of allowing for values that have been assigned, but not yet recorded, are:

1. Add a safety margin to the last used value, choosing a large enough margin so that the application should not set the counter to a value that might already have been assigned.
2. Treat all counter values as provisional. Restore the counter to the next apparently unused value, and in applications that use the counter, include logic to cover the situation where a counter value has already been assigned, and an application attempts to use it. The duplicate value can be detected by a duplicate key exception at the time the value is used (as a database or file key), at which point the application can obtain a new counter value and try again. Be careful to ensure that no side effects result from the original attempt to use the duplicated value.

The technique of locating the highest used counter value and provisionally assigning the next value can also be used as a backup method of assigning numbers when the named counter server is unavailable. However, it requires careful verification and testing, because the logic to handle duplicate keys is normally only exercised in unusual recovery situations.

If it is difficult to re-create the counter value from existing data repositories, then another possibility is that every so often (for example, once every 100 or 1000 numbers), the counter value could be logged to a record in a file. The recovery logic could re-create a suitable value for the named counter by taking the number logged in the file, and adding a safety margin, such as twice the interval at which the values are logged.

For systems running z/OS Release 3 or higher, system-managed duplexing can be used to maintain duplexed copies of the named counters in different coupling facilities. This greatly reduces the risk of losing access to the counters, but it involves some cost in performance and resources. There is still some theoretical risk of losing the structure, perhaps because of operational errors or software problems, and any data in the coupling facility cannot be considered permanent, so some method of reconstructing counter values might still be required.

Printing and spool files

CICS does not provide special commands for printing, but there are options on BMS and terminal control commands that apply only to printers, and for some printers you use transient data or SPOOL commands.

There are two issues associated with printing that do not usually occur in other types of end-user communication:

1. There are additional formatting considerations, especially for 3270 printers
2. The task that needs to print may not have direct access to the printer.

In addition, there are two distinct categories of printer, which have different application programming interfaces:

CICS printers
Printers defined as terminals to CICS and managed directly by CICS. They are usually low-speed devices located near the users, suitable for printing on demand of relatively short documents. The 3289 and 3262 are usually attached as CICS printers.

Non-CICS printers
Printers managed by the operating system or another application. These printers are usually high-speed devices located at the central processing site, appropriate for volume printing that does not have to be available immediately. They may also be advanced function or other printers that require special connections, management, or sharing.

Formatting for CICS printers

The application programming interface for writing to a printer terminal is essentially the same as for writing to a display. You can use terminal control commands (SENDs) for any CICS printer, and most of them are supported by BMS too (SEND MAP, SEND TEXT, and SEND CONTROL).

The problem of arranging that your task has the printer as its principal facility is discussed in "Using CICS printers" on page 353.

"BMS support levels" on page 367 lists the devices that BMS supports. For printers that are components of an outboard controller or LU Type 4, you can use batch data interchange (BDI) commands as well as terminal control and BMS. BDI commands are described in "Using batch data interchange" on page 272.

The choice between using BMS and terminal control is based on the same considerations as it is for a display terminal. Like displays, printers differ widely from one another, both in function and in the implementation of that function, and the differences are reflected in the data streams and device controls they accept.

When you use terminal control commands, your application code must format the output in the manner required by the printer. For line printers and similar devices, formatting has little programming impact. For high-function printers, however, the data stream often is very complex; formatting requires significant application code and introduces device dependencies into program logic.

For some of these terminals, coding effort is greatly reduced by using BMS, which relieves the programmer of creating or even understanding device data streams. BMS also removes most data stream dependencies from the application code so that the same program can support many types of printers, or a mixture of printers and displays, without change. BMS does not remove all device dependencies and imposes a few restrictions on format. It also involves extra path length; the amount depends on how many separate BMS requests you make, the complexity of your requests, and the corresponding path length avoided in your own program.

Requests for printed output

A CICS print request asks CICS to copy what is on the requesting screen to the first available printer on the same control unit. The overhead involved depends on whether a printer is available, and whether the requesting terminal is remote or local to CICS.

If no printer is available, and the request is from a remote or a local device:

- CICS reads the buffer to the display terminal. This involves transmitting every position on the screen, including nulls.

  For requests from a local device, the READ BUFFER command takes place at channel speeds, so that the large input message size does not increase response time too much, and does not monopolize the line.
- An error task is generated so that the terminal error program can dispose of the message. If a printer is available and the request is from a local device, this step is not needed.
- The 3270 print task (CSPP) is attached to write the entire buffer to the printer when it is available.

If a printer is available, and the request is from a remote device, CICS sends a very short data stream to the control unit asking for a copy of the requesting device buffer to be sent to the output device buffer.

CICS 3270 printers

Most of the additional format controls for printers that BMS provides are for a specific type of CICS printer, the 3270 printer. A 3270 printer is any printer that accepts the 3270 data stream, it is the hardcopy equivalent of a 3270 display.

It has a page buffer, corresponding to the display buffer of a 3270 display device. (See “The 3270 buffer” on page 275 for an introductory discussion of the 3270 data stream.) We discuss 3270 printers first and defer the simpler, non-3270 printers, until “Non-3270 CICS printers” on page 350.

A 3270 printer accepts two different types of formatting instructions: **buffer control orders** and **print format orders** . Buffer control orders are executed as they are received by the control unit, and they govern the way in which the buffer is filled. These are same orders that are used to format a 3270 display screen. We have already described some of the important ones in “Orders in the data stream” on page 279. For example, SBA (set buffer address) tells the control unit where in the buffer to place the data that follows, SF (start field), which signals an attributes byte and possibly field data, and so on. You can find a complete list in IBM 3270 Data Stream Programmers Reference.

In contrast, print format orders are not executed when they are received, but instead are stored in the buffer along with the data. These orders—NL (new line), FF (form feed), and so on—are interpreted only during a print operation, at which time they control the format of the printed output. They have no effect on displays, other than to occupy a buffer position; they look like blanks on the screen.

If you are writing to a 3270 printer, you can format with either buffer control orders or print format orders or a mixture of both. We show an example of formatting with buffer control orders in “Outbound data stream example” on page 282 . If you send this same data stream to a 3270 printer, it prints an image of the screen shown in Figure 92 on page 282 . You might choose to format printed output with buffer control orders so that you can send the same data stream to a display and a printer.

On the other hand, you might choose to format with print format orders so that you can send the same stream to a 3270 printer and a non-3270 printer (print format orders are the same as the format controls on many non-3270 printers). See the discussion of the NLEOM option on “NLEOM option” on page 349 for more details about this choice.

Here is a data stream using print format orders that produces the same *printed* output as the data stream on “Outbound data stream example” on page 282 , which uses buffer control orders.

Table 41. Example of data stream using print control orders

Bytes	Contents	Notes
1	X'FF'	“Formfeed” (FF) order, to cause printer to space to a new page.
2-23	blanks	22 blanks to occupy columns 1-22 on first line.
24-33	Car Record	Text to be printed, which appears in the next available columns (23-32) on line 1.
34	X'1515'	Two successive “new line” (NL) orders, to position printer to beginning of third line.
35-80	Employee No: _____ Tag _____ State: __	Text to be printed, starting at first position of line 3.
81	X'19'	“End-of-message” (EM) print order, which stops the printing.

Notice that the field structure is lost when you use print format orders. This does not matter ordinarily, because you do not use the printer for input. However, even if you format with print control orders, you might need to use buffer control orders as well, to assign attributes like color or underscoring to an area of text.

CICS 3270 printer options

For BMS, the special controls that apply to 3270 printers take the form of command options; PRINT, ERASE, L40, L64, L80, HONEOM, NLEOM, FORMFEED, and PRINTERCOMP.

In terminal control commands, ERASE is also expressed as an option, but the other controls are expressed directly in the data stream. The IBM 3270 Data Stream Programmers Reference and the IBM 3270 Data Stream Programmers Reference tell you how to encode them; the discussion that follows explains what they do.

PRINT option and print control bit

Writing to a 3270 display or printer updates the device buffer. On a display, the results are reflected immediately on the screen, which is a driven from the buffer. For a printer, however, there might be no visible effect, because printing does not occur until you turn on the appropriate bit in the "write control character".

The WCC is part of the 3270 data stream; see "The output datastream" on page 275 . For BMS, you turn on the print bit by specifying the PRINT option on a SEND MAP, SEND TEXT, or SEND CONTROL command, or in the map used with SEND MAP. If you are using terminal control SENDs, you must turn on the print bit with the CTLCHAR option.

A terminal write occurs on every terminal control SEND, and on every SEND MAP, SEND TEXT, or SEND CONTROL unless you are using the ACCUM or PAGING options. ACCUM delays writing until a page is full or the logical message is ended. When you use ACCUM, you should use the same print options on every SEND command for the same page. PAGING defers the terminal writes to another task, but they are generated in the same way as without PAGING.

The fact that printing does not occur until the print bit is on allows you to build the print buffer in stages with multiple writes and to change data or attribute bytes already in the buffer. That is, you can use the hardware to achieve some of the effects that you get with the ACCUM option of BMS. The NLEOM option affects this ability, however; see the discussion at "NLEOM option" on page 349.

ERASE option

Like the 3270 display buffer, the 3270 printer buffer is cleared only when you use a write command that erases. You do this by specifying the ERASE option, both for BMS and terminal control SENDs.

If the printer has the alternate screen size feature, the buffer size is set at the time of the erase, as it is for a display. Consequently, the first terminal write in a transaction should include erasing, to set the buffer to the size required for the transaction and to clear any buffer contents left over from a previous transaction.

Line width options: L40, L64, L80, and HONEOM

In addition to the print bit, the write control character contains a pair of bits that govern line length on printing.

If you are using terminal control commands, you use the CTLCHAR option to set these bits. For BMS, the default is the one produced by the HONEOM option, which stands for "honor end-of-message". With this setting, the printer formats according to the buffer control and print format orders only, stopping printing at the first EM (end-of-message) character in the buffer. Only if you attempt to print beyond the maximum width for the device (the platen width) does the printer move to a new line on its own.

However, you also can specify that the line length is fixed at 40, 64, or 80 characters (options L40, L64 and L80). If you do, the printer ignores certain print format orders, moves to a new line when it reaches the specified line size, and prints the entire buffer. The print format orders that are ignored are NL (new line), CR (carriage return), and EM (end-of-message). Instead they are printed, as graphics.

If you use L40, L64, or L80 under BMS, only use the value that corresponds to the page width in your terminal definition (see "Determining the characteristics of a CICS printer" on page 351). The reason is that BMS calculates buffer addresses based on the page size, and these addresses are wrong if you use a different page width.

NLEOM option

BMS ordinarily uses buffer control orders, rather than print format orders, to format for a 3270 printer, whether you are using SEND TEXT or SEND MAP. However, you can tell BMS to use print format orders only, by specifying the NLEOM option.

If you do, BMS formats the data entirely with blanks and NL (new line) characters, and inserts an EM (end-of-message) character after your data. NLEOM implies HONEOM. (NLEOM support requires **standard** BMS; it is not available in minimum BMS.)

You might want to do this in order to maintain compatibility with an SCS printer (print format orders are compatible with the corresponding SCS control characters). The following operational differences might cause you to choose or avoid NLEOM.

Blank lines

The 3270 printer suppresses null lines during printing. That is, a line that has no data fields and appears blank on the display screen is omitted when the same map is sent to a printer. Under BMS, you can force the printed form to look exactly like the displayed form by placing at least one field on every line of the screen; use a field containing a single blank for lines that would otherwise be empty. Specifying NLEOM also has this effect, because BMS uses a new line character for every line, whether or not there is any data on it.

Multiple sends

With NLEOM, data from successive writes is stacked in the buffer, since it does not contain positioning information. However, BMS adds an EM (end-of-message) character at the end of data on each SEND with NLEOM, unless you are using the ACCUM option. When printing occurs, the first EM character stops the printing, so that only the data from the first SEND with NLEOM (and any unerased data up to that point in the buffer) gets printed. The net effect is that you cannot print a buffer filled with multiple SEND commands with NLEOM unless you use the ACCUM option.

Page width

BMS always builds a page of output at a time, using an internal buffer whose size is the number of character positions on the page. (See “Determining the characteristics of a CICS printer” on page 351 for a discussion of how BMS determines the page size.) If you are using buffer control orders to format, the terminal definition must specify a page width of 40, 64, 80 or the maximum for the device (the platen size); otherwise your output might not be formatted correctly. If you are using NLEOM, on the other hand, the terminal definition may specify any page width, up to the platen size.

Total page size

If you are using buffer control orders, the product of the number of lines and the page width must not exceed the buffer size, because the buffer is used as an image of the page. Unused positions to the right on each line are represented by null characters. If you use NLEOM, however, BMS does not restrict page size to the buffer capacity. BMS builds the page according to the page size defined for the terminal and then compresses the stream using new-line characters where possible. If the resulting stream exceeds the buffer capacity, BMS uses multiple writes to the terminal to send it.

FORMFEED

The FORMFEED option causes BMS to put a form feed print format order (X'0C') at the beginning of the buffer, if the printer is defined as capable of advancing to the top of the form (with the FORMFEED option in the associated TYPETERM definition). CICS ignores a form feed request for a printer defined without this feature.

If you issue a SEND MAP using a map that uses position (1,1) of the screen, you overwrite the order and lose the form feed. This occurs whether you are using NLEOM or not.

If you use FORMFEED and ERASE together on a SEND CONTROL command, the results depend on whether NLEOM is present. Without NLEOM, SEND CONTROL FORMFEED ERASE sends the form feed character followed by an entire page of null lines. The printer suppresses these null lines, replacing them with a single blank line. With NLEOM, the same command sends the form feed character followed by one new line character for each line on the page, so that the effect is a full blank page, just as it is on a non-3270 printer.

PRINTERCOMP option

When you SEND TEXT to a printer, there is one additional option that affects page size. This is the PRINTERCOMP option, which is specified in the PROFILE associated with the transaction you are executing, rather than on individual SEND TEXT commands. In the default profile that CICS provides, the PRINTERCOMP value is NO.

Under PRINTERCOMP(NO), BMS produces printed output consistent with what it would send to a 3270 display. For the display, BMS precedes the text from each SEND TEXT command with an attribute byte, and it also starts each line with an attribute byte. These attribute bytes take space on the screen, and therefore BMS replaces them with blanks for printers if PRINTERCOMP is NO. If PRINTERCOMP is YES, BMS suppresses these blanks, allowing you to use the full width of the printer and every position of the buffer. New line characters that you embed in the text are still honored with PRINTERCOMP(YES), as they are with PRINTERCOMP(NO).

You should use PRINTERCOMP(NO) if you can, for compatibility with display devices and to ensure consistent results if the application uses different printer types, even though it reduces the usable line width by one position.

Non-3270 CICS printers

A **non-3270 printer** is any printer that does not accept the 3270 data stream, such as an SNA character set (SCS) printer. A non-3270 printer can be a 3270-family device, and many devices, like the 3287 and 3262, can be either 3270 printers or SCS (non-3270) printers, depending on how they are defined at the control unit.

Non-3270 printers do not have page buffers, and therefore do not understand buffer control orders. Formatting is accomplished entirely with print control orders. For compatibility with 3270 printers, BMS formats for them by constructing an image of a page in memory, and always prints a full page at a time. However, you can define any size page, provided you do not exceed the platen width, as there is no hardware buffer involved. BMS transmits as many times as required to print the page, just as it does for a 3270 printer using the NLEOM option.

BMS formats for these printers with blanks and NL (new line) characters. It uses form feed (FF) characters as well, if the definition of your terminal indicates form feed support.

BMS also uses horizontal tabs to format if the terminal definition has the HORIZFORM option and the map contains HTAB specifications. Similarly, it uses vertical tabs if the terminal definition specifies VERTICALFORM and your map includes VTAB. Tab characters can shorten the data stream considerably. If tabs are used, BMS assumes that the current task, or some earlier one, has already set the tabs on the printer. On an SCS printer, you set the tabs with a terminal control SEND command, as explained in IBM 3270 Data Stream Programmers Reference . For other non-3270 printers, you should consult the appropriate device guide.

For SEND TEXT to an SCS printer, BMS does not recognize any non-3270 control codes in the input datastream except newline (X'15') and set attribute (X'28'). All other characters are assumed to be display characters. In particular, the datastream might be affected if you attempt to use the transparency control order (X'35') under BMS. This control order normally causes the data that follows it to be ignored (the next byte contains the length of the data to be ignored). However, because BMS does not recognize the X'35' control order, it processes the data that follows the transparency control order as if it were a normal part of the datastream.

If this data cannot be processed correctly, BMS might remove it from the datastream; for example, if the X'28' character is encountered in the transparency sequence it is mistaken for a set attribute control order, in which case the two bytes following it are mistaken for an attribute description, and all three

bytes might be removed from the datastream. The X'0C' character (formfeed) is also liable to be removed from the datastream. If you want to send a datastream including a transparency sequence which contains characters that might be recognized and altered by BMS, the recommended method is to use a terminal control SEND command, rather than BMS.

SCS input

SCS printers have limited input capability, in the form of *program attention* keys.

However, these keys are not like the attention keys that are described in “Input from a 3270 terminal” on page 284 . Instead, they transmit an unformatted data stream consisting of the characters 'APAK *nn* ', where *nn* is the 2-digit PA key number; 'APAK 01' for PA key 1, for example.

You can capture such input by defining a transaction named 'APAK' (APAK is the transaction identifier, not the TASKREQ attribute value, because SCS inputs do not look like other PA key inputs.) A program invoked by this transaction can determine which PA key was pressed by issuing a RECEIVE and numeric positions of the input.

TCP/IP-attached printers

You can print to a TCP/IP-attached printer from CICS TS. Printing to a TCP/IP-attached printer can be accomplished by using z/OS Communications Server (VTAM) or by using a printing software.

Using z/OS Communications Server (VTAM)

TCP/IP-attached printers can be defined to VTAM and the corresponding VTAM resources can be acquired and used by CICS just like any other VTAM printer LU. No additional software beyond z/OS is required.

CICS TS talks indirectly through the TCP/IP Telnet to print to a TCP/IP-attached printer (or a TCP/IP-attached PCOM terminal). The printer communicates with the TCP/IP Telnet, so CICS sends the printout to the Telnet LU and the TCP/IP Telnet passes it onto the printer.

Therefore, you should define the printer as a regular VTAM LU just like you did when using SNA, except the LU name is one of the LUs in the pool of TCP/IP Telnet LUs. CICS does not know that it is communicating to a TCP/IP-attached printer. It does not even know it is talking to TCP/IP Telnet. To CICS, it is just a VTAM luname. For more information about how to define the printer, see Advanced LU name mapping: The associated printer function in the *z/OS Communications Server: IP Configuration Guide*.

Using a printing software

You can print to a TCP/IP-attached printer by using a printing software such as the IBM z/OS Infoprint Server.

If you use the IBM z/OS Infoprint Server, you can use NetSpool to simulate a VTAM printer and write the printing output to the MVS job entry system (JES) SPOOL. Then you could use IP PrintWay to send the print output from the SPOOL to the TCP/IP-attached printer. NetSpool and IP PrintWay perform complementary functions. NetSpool converts print requests from VTAM applications into S/390® line data and creates output data sets on the JES spool, using information in a NetSpool print-characteristics data set created by the installation. IP PrintWay can transmit data sets created by NetSpool from the JES spool to printers in a TCP/IP network.

Using printers with CICS

You can use the following to show you how to program for CICS printers and non-CICS printers.

Determining the characteristics of a CICS printer

If you are writing a program that supports more than one type of CICS printer, you might need to determine the characteristics of a particular printer. You can use the ASSIGN and INQUIRE TERMINAL commands for this purpose.

The ASSIGN options that apply to terminals, including several that are specific to printers are shown in Table 34 on page 266.

The INQUIRE TERMINAL options that apply specifically to printers and the corresponding parameters in the terminal definition are shown in Table 42 on page 352 :

Table 42. INQUIRE TERMINAL options for printers

INQUIRE option	Source in TERMINAL or TYPETERM definition	Description
PAGEHT	x of PAGESIZE(x,y)	Number of lines per page (for alternate screen size terminals, reflects default size)
PAGEWD	y of PAGESIZE(x,y)	Number of characters per line (for alternate screen size terminals, reflects default size)
DEFPAGEHT	x of PAGESIZE(x,y)	Number of lines per page in default mode (alternate screen size terminals only)
DEFPAGEWD	y of PAGESIZE(x,y)	Number of characters per line in default mode (alternate screen size terminals only)
ALTPAGEHT	x of ALTPAGE(x,y)	Number of lines per page in alternate mode (alternate screen size terminals only)
ALTPAGEWD	y of ALTPAGE(x,y)	Number of characters per line in alternate mode (alternate screen size terminals only)
DEVICE	DEVICE	The device type (see the Introduction to System programming commands for possible values)
TERMMODEL	TERMMODEL	The model number of the terminal (either 1 or 2)

BMS page size, 3270 printers

BMS uses both the terminal definition and the profile of the transaction that is running to determine the page size of a CICS printer. The profile is used when the terminal has the alternate screen size feature, to determine whether to use default or alternate size. The default profile in CICS specifies “default” size for the screen. The following table lists the values used.

Table 43. Priority of parameters defining BMS page size. BMS uses the first value in the appropriate column that has been specified in the terminal definition.

Terminals with alternate screen size, using alternate size	Terminals with alternate screen size, using default size	Terminals without alternate screen size feature
ALTPAGE	PAGESIZE	PAGESIZE
ALTSCREEN	DEFSCREEN	TERMMODEL
DEFSCREEN	TERMMODEL	(12,80)
TERMMODEL	(12,80)	
(12,80)		

The definition of a “page” is unique to BMS. If you are printing with terminal control SEND commands, you define what constitutes a page, within the physical limits of the device, by your print format. If you need to know the buffer size to determine how much data you can send at once, you can determine this from the SCRNHT and SCRNWD values returned by the ASSIGN command.

Supporting multiple printer types

When you are writing programs to support printers that have different page sizes, it is not always possible to keep device dependencies like page size out of the program. However, BMS helps with this problem in two ways.

1. You can refer to a map generically and have BMS select the map that was designed for the terminal associated with your task.
2. If you are using SEND TEXT, BMS breaks the text into lines at word boundaries, based on the page size of the receiving terminal. You can also request header and trailer text on each page.

Using CICS printers

You can print to a CICS printer using the following methods; printing with a START command, printing with transient data, or printing with BMS routing.

About this task

As we noted, the second issue that frequently arises in printing concerns ownership of the printer. Requests for printing often originate from a user at a display terminal. The task that processes the request and generates the printed output is associated with the user's terminal and therefore cannot send output directly to the printer.

If your task does not own the printer it wants to use, it must create another task, which does, to do the work using one of the following methods.

1. Create the task with a START command.
2. Write to an intrapartition transient data queue that triggers the task.
3. Direct the output to the printer in a BMS ROUTE command.
4. Use the ISSUE PRINT command, if you need only a screen copy.

Printing with a START command

The first technique for creating the print task is to issue a START command in the task that wants to print. The command names the printer as the terminal required by the STARTed task in the TERMID option and passes the data to be printed, or instructions on where to find it, in the FROM option.

About this task

START causes CICS to create a task whose principal facility is the designated terminal when that terminal is available.

The program executed by the STARTed task, which you must supply, retrieves the data to be printed (using a RETRIEVE command), and then writes it to its terminal (the printer) with SEND, SEND MAP, or SEND TEXT. For example:

```
(build output in OUTAREA, formatted as expected by the
STARTed task)
EXEC CICS START TRANSID(PRNT) FROM(OUTAREA) TERMID(PRT1)
LENGTH(OUTLNG) END-EXEC.
```

Figure 100. Task that wants to print (on printer PRT1)

```
EXEC CICS RETRIEVE INTO(INAREA) LENGTH(INLNG) END-EXEC.
(do any further data retrieval and any formatting required)
EXEC CICS SEND TEXT FROM(INAREA) LENGTH(INLNG) ERASE PRINT END-EXEC.
(repeat from the RETRIEVE statement until a NODATA condition
arises)
```

Figure 101. STARTed task (executing transaction PRNT)

The task associated with the printer loops until it exhausts all the data sent to it, in case another task sends data to the same printer before the current printing is done. Doing this saves CICS the overhead

of creating new tasks for outputs that arrive while earlier ones are still being printed; it does not change what finally gets printed, as CICS creates new tasks for the printer as long as there are unprocessed START requests.

Printing with transient data
The second method for creating the print task involves transient data. A CICS intrapartition transient data queue can be defined to have a property called a *trigger*. When the number of items on a queue with a trigger reaches the trigger value, CICS creates a transaction to process the queue.

About this task

The queue definition tells CICS what transaction this task executes and what terminal, if any, it requires as its principal facility.

You can use this mechanism to get print data from the task that generates it to a task that owns the printer. A transient data queue is defined for each printer where you direct output in this way. A task that wants to print puts its output on the queue associated with the required printer (using WRITEQ TD commands). When enough items are on the queue and the printer is available, CICS creates a task to process the queue. (For this purpose, the trigger level of "enough" is typically defined as just one item.) The triggered task retrieves the output from the queue (with READQ TD commands) and writes it to its principal facility (the printer), with SEND, SEND MAP, or SEND TEXT commands.

As in the case of a STARTed printer task, you have to provide the program executed by the task that gets triggered. The sample programs distributed with CICS contain a complete example of such a program, called the "order queue print sample program". Samples describes this program in detail.

Task that wants to print (on printer PRT1):
This example shows you the syntax for writing data to a transient data queue called 'PRT1' which is the predefined symbolic destination for printer PRT1.

```
 (do any formatting or other processing required)
EXEC CICS WRITEQ TD QUEUE('PRT1') FROM(OUTAREA)
LENGTH(OUTLNG) END-EXEC.
```

Task that gets triggered:
This example shows you how the print task determines the name of its queue using an ASSIGN command rather than a hard-coded value so that the same code works for any queue (printer). Like its START counterpart, this task loops through its read and send sequence until it detects the QZERO condition, indicating that the queue is empty.

```
EXEC CICS ASSIGN QNAME(QID) END-EXEC.
EXEC CICS READQ TD QUEUE(QID) INTO(INAREA) LENGTH(INLNG)
RESP(RESPONSE) END-EXEC.
IF RESPONSE = DFHRESP(QZERO) GO TO END-TASK.
(do any error checking, further data retrieval and formatting required)
EXEC CICS SEND FROM(INAREA) LENGTH(INLNG) END-EXEC.
(repeat from READQ command)
```

While this is just an efficiency issue with the STARTed task, it is critical for transient data; otherwise unprocessed queue items can accumulate under certain conditions. (See “Automatic transaction initiation (ATI)” on page 305 for details on the creation of tasks to process transient data queues with triggers.)

If you use this technique, you need to be sure that output to be printed as a single unit appears either as a single item or as consecutive items on the queue. There is no fixed relationship between queue items and printed outputs; packaging arrangements are strictly between the programs writing the queue and the one reading it. However, if a task writes multiple items that need to be printed together, it must ensure that no other task writes to the queue before it finishes. Otherwise the printed outputs from several tasks may be interleaved.

If the TD queue is defined as recoverable, CICS prevents interleaving. Once a task writes to a recoverable queue, CICS delays any other task that wants to write until the first one commits or removes what it has written (by SYNCPOINT or end of task). If the queue is not recoverable, you need to perform this function

yourself. One way is to ENQUEUE before writing the first queue item and DEQUEUE after the last. (See “Transient data control” on page 303 for a discussion of transient data queues.)

Printing with BMS routing
A task also can get output to a printer other than its principal facility with BMS routing. This technique applies only to BMS logical messages (the ACCUM or PAGING options) and thus is most appropriate when you are already building a logical message.

About this task

When you complete a routed message, CICS creates a task for each terminal named in a route list. This task has the terminal as its principal facility, and uses CSPG, the CICS-supplied transaction for displaying pages, to deliver the output to the printer. So routing is similar in effect to using START commands, but CICS provides the program that does the printing. (See “Message routing” on page 418 for more information about routing.)

Using Non-CICS printers

The steps listed here show you how to format your output, deliver your output, and notify the application when output is ready for printing, on a printer managed outside CICS :

Procedure

1. Format your output in the manner required by the application or subsystem that controls the printer you want to use.
2. Deliver the output to the application or subsystem that controls the printer in the form required by that application.
3. If necessary, notify the application that the output is ready for printing.

Formatting for non-CICS printers
For some printers managed outside CICS, you can format output with BMS. However, for most printers you need to meet the format requirements of the application that drives the printer.

Formatting output with BMS, is explained in “Programming for non-CICS printers” on page 356 . The format requirements of the application that drives the printer can be the device format or an intermediate form dictated by the application. For conventional line printers, formatting is a matter of producing line images and, sometimes, adding carriage-control characters.

Non-CICS printers: Delivering the data
Print data is typically conveyed to an application outside of CICS by placing the data in an intermediate file, accessible to both CICS and the application. The type of file, as well as the format within the file, is dictated by the receiving application.

It is typically one of the file types listed in the first column of Table 44 on page 355. The second column of the table shows which groups of CICS commands you can use to create such data.

Table 44. Intermediate files for transferring print data to non-CICS printers

File type	Methods for writing the data
Spool files	CICS spool commands (SPOOLOPEN, SPOOLWRITE, etc.) Transient data commands (WRITEQ TD) Terminal control and BMS commands (SEND, SEND MAP, etc.)
BSAM	CICS spool commands (SPOOLOPEN, SPOOLWRITE, etc.) Transient data commands (WRITEQ TD)
VSAM	CICS file control commands (WRITE)
Db2	EXEC SQL commands
IMS	EXEC DLI commands or CALL DLI statements

Programming for non-CICS printers
If you are using VSAM, Db2, or IMS, the CICS application programming commands you can use are determined by the type of file you are using. For BSAM and spool files, however, you have a choice. The CICS definition of the file (or its absence) determines which commands you use.

The file can be:

- An extra-partition transient data queue (see "Transient data control" on page 303 for information about transient data queues)
- The output half of a sequential terminal (see "Using sequential terminal support" on page 270 and "Support for non-3270 terminals" on page 380)
- A spool file (see "CICS interface to JES" on page 358)

Both transient data queue definitions and sequential terminal definitions point to an associated data definition (DD) statement in the CICS start-up JCL, and it is this DD statement that determines whether the file is a BSAM file or a spool file. Files created by CICS spool commands do not require definition before use and are spool files by definition.

If the printing application accepts BSAM or spool file input, there are several factors to consider in deciding how to define your file to CICS:

System definitions
Files created by the SPOOLOPEN command do not have to be defined to CICS or the operating system, whereas transient data queues and sequential terminals must be defined to both before use.

Sharing among tasks
A file defined as a transient data queue is shared among all tasks. This allows you to create a print file in multiple tasks, but it also means that if your task writes multiple records to the queue that must be printed together (lines of print for a single report, for example), you must include enqueue logic to prevent other tasks from writing their records between yours. This is the same requirement that was cited for intrapartition queues in "Printing with transient data" on page 354 . In the case of extra-partition transient data, however, CICS does not offer the recoverability solution, and your program must prevent the interspersing itself.

In contrast, a file created by a SPOOLOPEN can be written only by the task that created it. This eliminates the danger of interleaving output, but also prevents sharing the file among tasks.

A spool file associated with a sequential terminal can be written by only one task at a time (the task that has the terminal as its principal facility). This also prevents interleaving, but allows tasks to share the file serially.

Release for printing
Both BSAM and spool files must be closed in order for the operating system to pass them from CICS to the receiving application, and therefore printing does not begin until the associated file is closed. Files created by SPOOLOPEN are closed automatically at task end, unless they have already been closed with a SPOOLCLOSE command. In contrast, an extrapartition transient data queue remains open until some task closes it explicitly, with a SET command. (It must be reopened with another SET if it is to be used later.) So transient data gives you more control over release of the file for processing, at the cost of additional programming.

A file that represents the output of a sequential terminal does not get closed automatically (and so does not get released for printing) until CICS shutdown, and CICS does not provide facilities to close it earlier. If you use a sequential terminal to pass data to a printer controlled outside of CICS, as you might do in order to use BMS, you should be aware of this limitation.

Formatting
If you define your file as a sequential terminal, you can use BMS to format your output. This feature allows you to use the same maps for printers managed outside of CICS—for example, line printers managed by the MVS job entry subsystem (JES)—that you use for CICS display and printer terminals.

If you choose this option, remember that BMS always sends a page of output at a time, using the page size in the terminal definition, and that the data set representing the output from a sequential terminal is not released until CICS shutdown.

Spool file limits

Operating systems identify spool files by assigning a sequential number. There is an upper limit to this number, after which numbers are reused. The limit is typically large, but it is possible for a job that runs a long time (as CICS can) and creates a huge number of spool files (as an application under CICS can) to exceed the limit. If you are writing an application that generates a large number of spool files, consult your system programmer to ensure that you are within system limits. A new spool file is created at each SPOOLOPEN statement and each open of a transient data queue defined as a spool file.

Notifying the print application

When you deliver the data to a print application outside CICS, you might need to notify the application that you have data ready to process.

You do not need to do this if the application runs automatically and knows to look for your data. For example, to print on a printer owned by the MVS job entry system (JES), all you need to do is create a spool file with the proper routing information. JES does the rest.

However, sometimes you need to submit a job to do the processing, or otherwise signal an executing application that you have work for it.

To submit a batch job from a CICS task, you need to create a spool file which contains the JCL for the job, and you need to direct this file to the JES internal reader. You can create the file in any of the three ways listed for spool files in Table 44 on page 355, although if you use a sequential terminal, the job does not execute until CICS shuts down, as noted earlier. For files written with spool commands, the information that routes the file to the JES internal reader is specified in the SPOOLOPEN command. For transient data queues and sequential terminals, the routing information appears on the first record in the file, the “JOB card”.

The output to be printed can be embedded in the batch job (as its input) or it can be passed separately through any form of data storage that the job accepts.

Printing display screens

If your printing requirement is to copy a display screen to a printer, you have choices additional to the ones already described. Some of these are provided by the terminal hardware itself, and some by CICS.

Some of the CICS support also depends on hardware features, and so your options depend on the type of terminals involved and, in some cases, the way in which they are defined to CICS . For more information, see TERMINAL resources.

CICS print key

The first such option is the *CICS print key* (also called the *local copy key*). This option allows a user to request a printed copy of a screen by pressing a program attention key, provided the terminal is a 3270 display or a display in 3270 compatibility mode. Print key support is optional in CICS ; the system programmer decides whether to include it and what key is assigned. The default is PA1. See the PRINT option in the DISPLAY and PRINT options for combined lists of information.

The print key copies the display screen to the first available printer among those defined as eligible. Which printers are eligible depends on the definition of the display terminal from which the request originates, as follows:

- For z/OS Communications Server 3270 displays defined without the “printer-adapter” feature, the printers named in the PRINTER and ALTPRINTER options of the terminal definition are eligible. PRINTER is to be used if available; ALTPRINTER is second choice. If both are unavailable, the request is queued for execution when PRINTER becomes available.
- For the 3270 compatibility mode of the 3790 and a 3650 host conversational (3270) logical unit, the same choices apply.
- For z/OS Communications Server 3270 displays defined with the printer-adapter feature, copying is limited to printers on the same control unit as the display. The printer authorization matrix within the control unit determines printer eligibility.

- For a 3270 compatibility mode logical unit of the 3790 with the printer-adapter feature, the 3790 determines eligibility and allocates a printer for the copy.
- For a 3275 with the printer-adapter feature, the print key prints the data currently in the 3275 display buffer on the 3284 attached to the display.

Where CICS chooses the printer explicitly, as it does in the first three cases above, the printer has to be in service and not attached to a task to be "available" for a CICS print key request. Where a control unit or subsystem makes the assignment, availability and status are determined by the subsystem. The bracket state of the device typically determines whether it is available or not.

ISSUE PRINT and ISSUE COPY

An application can initiate copying a screen to a printer as well as the user, with the **ISSUE PRINT** and **ISSUE COPY** commands. **ISSUE PRINT** simulates the user pressing the CICS print key, and printer eligibility and availability are the same as for CICS print key requests.

You can use the **ISSUE COPY** command to copy a screen in a task that owns the printer, as opposed to the task that owns the terminal which is to be copied. It copies the buffer of the terminal named in the TERMID option to the buffer of the principal facility of the issuing task. The method of copying and the initiation of printing once the copy has occurred is controlled by the "copy control character" defined in the CTLCHAR option of the **ISSUE COPY** command; for the bit settings in this control character, see IBM 3270 Data Stream Programmers Reference. The terminal whose buffer is copied and the printer must both be 3270 logical units and they must be on the same control unit.

Hardware print key

Some 3270 terminals also have a *hardware print key* . Pressing this key copies the screen to the first available and eligible printer on the same control unit as the display. This function is performed entirely by the control unit, whose configuration and terminal status information determine eligibility and availability. If no printer is available, the request fails; the user is notified by a symbol in the lower left corner of the screen and must retry the request later.

BMS screen copy

Both the CICS and hardware print keys limit screen copies to a predefined set of eligible printers, and if more than one printer is eligible, the choice depends on printer use by other tasks. For screens created as part of a BMS logical message, a more general screen copy facility is available. Users can print any such screen with the "page copy" option of the CICS transaction for displaying logical messages, CSPG. With page copy, you name the specific printer to receive the output, and it does not have to be on the same control unit as the display. For information about CSPG, see CSPG - page retrieval.

CICS interface to JES

CICS provides a programming interface to **JES** (the Job Entry Subsystem component of MVS) that allows CICS applications to create and retrieve **spool** files.

Spool files are managed by JES and are used to buffer output directed to low-speed **peripheral** devices (printers, punches, and plotters) between the job that creates them and actual processing by the device. Input files from card readers are also spool files and serve as buffers between the device and the jobs that use the data.

The interface consists of five commands:

- SPOOLOPEN INPUT, which opens a file for input
- SPOOLOPEN OUTPUT, which opens a file for output
- SPOOLREAD, which retrieves the next record from an input file
- SPOOLWRITE, which adds one record to an output file
- SPOOLCLOSE, which closes the file and releases it for subsequent processing by JES

“Input” and “output” here refer to the CICS point of view here; what is spool output to one job is always spool input to another job or JES program.

These commands can be used with either the JES2 or JES3 form of JES, although some restrictions apply to each (see “Spool interface restrictions” on page 361). The term JES refers to both. You can support the requirements of other products to exchange files with other systems connected through a JES remote spooling communications subsystem (RSCS) network.

You can use the spool commands to do the following types of things:

- Create an (output) file for printing or other processing by JES. JES manages most of the “unit record” facilities of the operating system, including high-speed printers, and card readers. In order to use these facilities, you pass the data to be processed to JES in a spool file. See Figure 102 on page 359

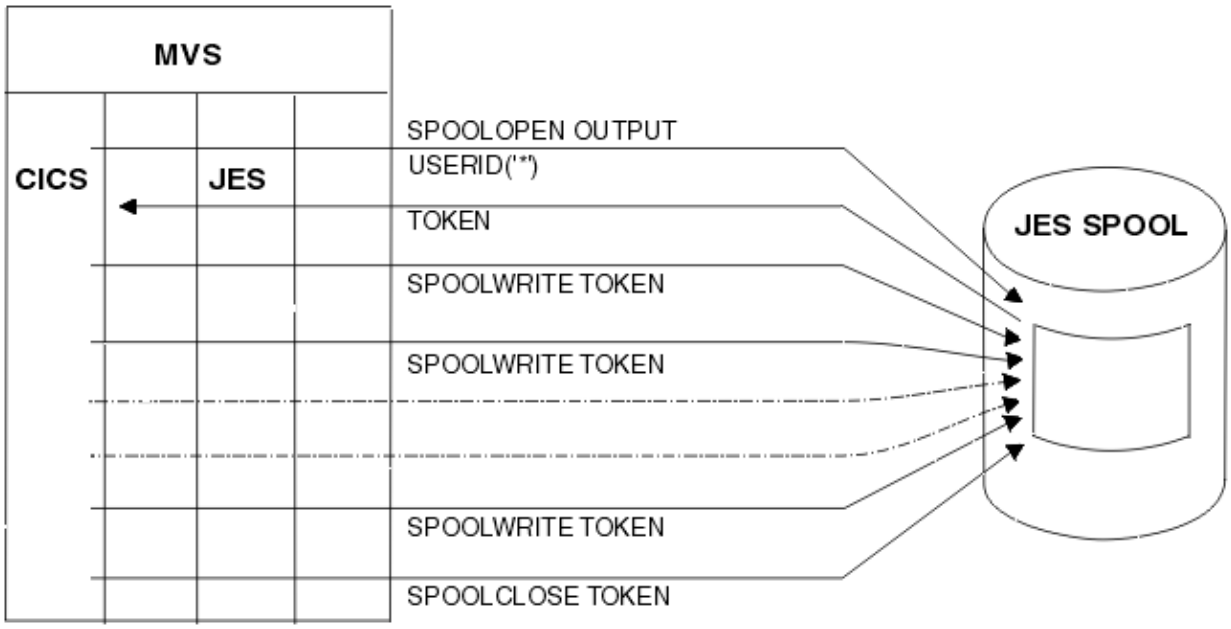


Figure 102. Create a file and write it to the JES spool

- Submit a batch job to MVS. Spool files directed to the JES “internal reader” are treated as complete jobs and executed.
- Create an (output) file to pass data to another job (outside of your CICS), that runs under MVS.
- Retrieve data passed from such a job. See Figure 103 on page 360

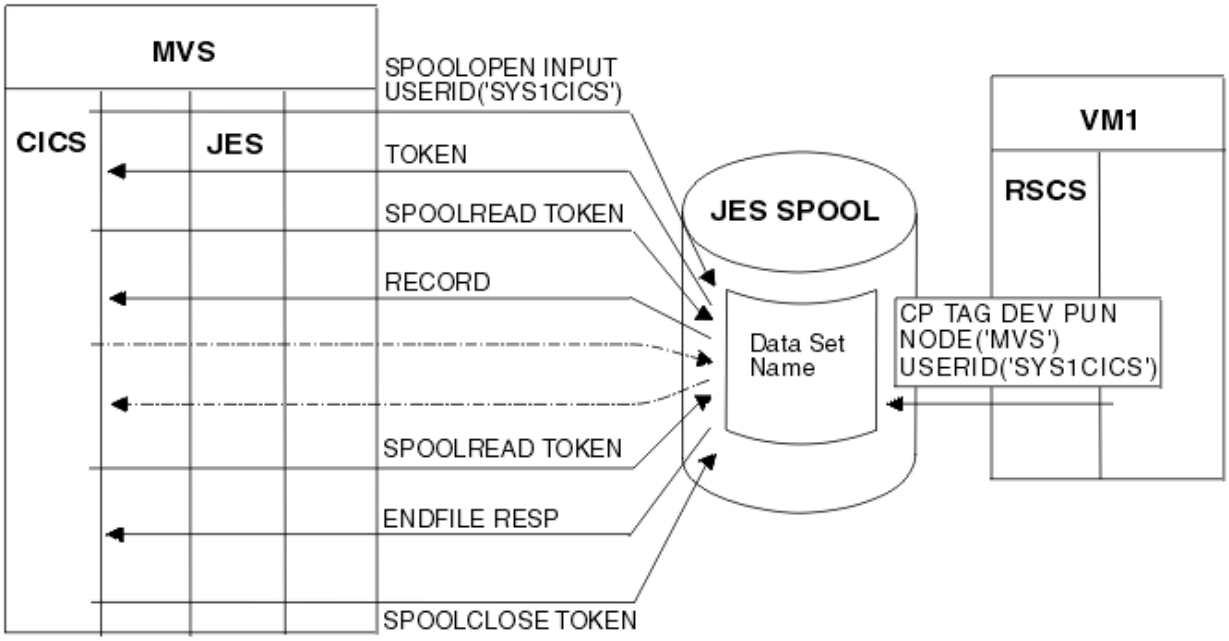


Figure 103. Retrieve data from the JES spool

- Create a file to pass data to another operating system, such as VM, VSE/ESA , or an MVS system other than the one under which your CICS is executing. See Figure 104 on page 360.

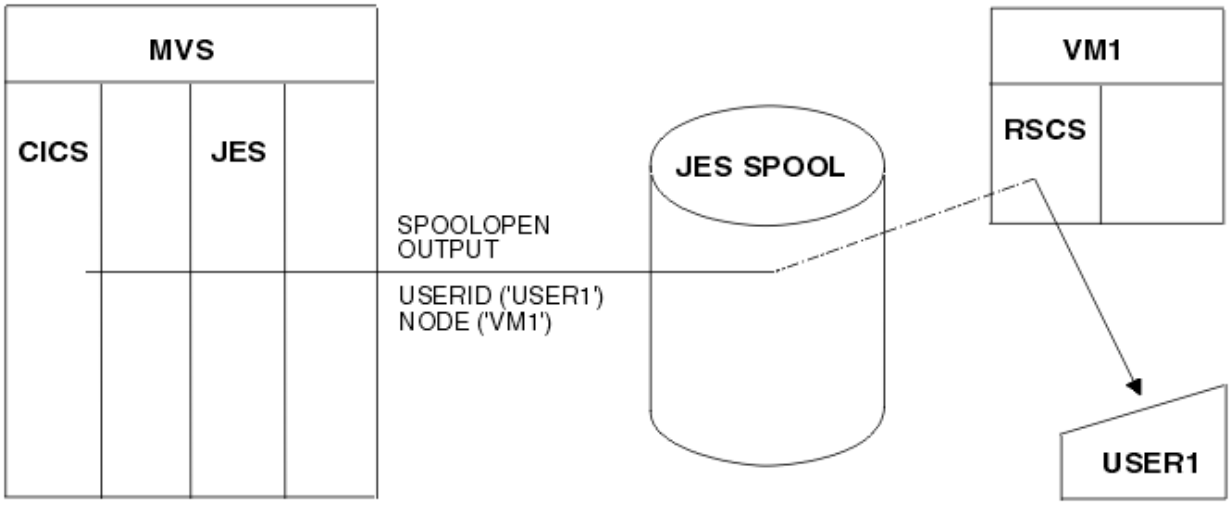


Figure 104. Send a written file to a remote destination

Using the CICS interface to JES

To use the CICS interface to JES, you must define the DFHSIT SPOOL=YES system initialization parameter in your CICS startup JCL.

About this task

You must specify RESP or NOHANDLE on the EXEC CICS SPOOLCLOSE, SPOOLOPEN, SPOOLREAD, and SPOOLWRITE commands. RESP bears a one-to-one correspondence with HANDLE CONDITION. If you do not code RESP, your program abends. You can also code the RESP2 option.

Transactions that process SYSOUT data sets larger than 1000 records, either for INPUT or for OUTPUT, are likely to have a performance impact on the rest of CICS. When you cannot avoid such a transaction, you should carefully evaluate general system performance. You should introduce a pacing mechanism if the effects on the rest of CICS are unacceptable.

All access to a JES spool file must be completed within one logical unit of work. Issuing an EXEC CICS SYNCPOINT command implicitly issues a SPOOLCLOSE command for any open report.

Spool interface restrictions
There are internal limits in JES that you must consider when you are designing applications. Some apply to JES2, some to JES3 and some to both.

In particular:

- JES2 imposes an upper limit on the total number of spool files that a single job (such as CICS) can create. If CICS exceeds this limit during its execution, subsequent SPOOLOPEN OUTPUT commands fail with the ALLOCERR condition.
- JES3 does not impose such a limit explicitly, but for both JES2 and JES3, some control information for each file created persists for the entire execution of CICS. For this reason, creating large numbers of spool files can stress JES resources; you should consult your system programmer before designing such an application.
- Spool files require other resources (buffers, queue elements, disk space) until they are processed. You need to consult your systems staff if you are producing large files or files that can wait a long time for processing at their destinations.
- Code NODE('*') and USERID('*') if you want to specify the local spool file and to enable the OUTDESCR operand to override the NODE and USERID operands. Do not use NODE('*') with any other userid. If the NODE and USERID operands specify explicit identifiers, the OUTDESCR operands cannot override them.
- Ensure that your system is defined so that data sets produced by CICS are not in HELD status in JES. CICS does not search for data sets in HELD status when the EXEC CICS SPOOLOPEN INPUT command is issued.

Creating output spool files

To create an output spool file, the program starts by issuing a SPOOLOPEN OUTPUT command, to allocate an output data set. Records are added to the file using the SPOOLWRITE command after which the SPOOLCLOSE command is used to release the file to JES for delivery or processing.

About this task

The NODE and USERID options on the SPOOLOPEN OUTPUT command tell JES what to do with the file when it is complete, and there are other options to convey formatting and other processing to JES if appropriate. SPOOLOPEN returns a unique token in the TOKEN field, which must be used in all subsequent SPOOLWRITE and SPOOLCLOSE commands to identify the file being written.

Thereafter, the task puts data into the file with SPOOLWRITE commands that specify the token value that was returned on the SPOOLOPEN OUTPUT command. Spool files are sequential; each SPOOLWRITE adds one record to the file. When the file is complete, the task releases the file to JES for delivery or processing by issuing a SPOOLCLOSE with the token that identifies the file.

A task can create multiple output spool files, and it can have more than one open at a time; operations on different files are kept separate by the token. However, a spool file cannot be shared among tasks, or across logical units of work in the same task. It can be written only by the task that opened it, and if that task fails to close the file before a SYNCPOINT command or task end, CICS closes it automatically at these points.

If the node is a remote system, the data set is queued on the JES spool against the destination userid. The ID of this destination user was specified on the SPOOLOPEN OUTPUT USERID parameter. If the node is a remote VM system, the data is queued in the VM RDR queue for the ID that was specified on the same USERID parameter.

Note: If you want the job you submit to execute as soon as possible, you should end your spool file with a record that contains /*EOF in the first five characters. This statement causes JES to release your file for processing, rather than waiting for other records to fill the current buffer before release.

Writing output spool files to the MVS internal reader
To write your output to the MVS internal reader, specify USERID("INTRDR") on the **SPOOLOPEN OUTPUT** command, and also use an explicit node name. Do not use NODE('*'). INTRDR is an IBM-reserved name that identifies the internal reader.

About this task

The output records written by your **SPOOLWRITE** commands must be JCL statements, starting with a JOB statement.

Ensure that you specify the NOCC option on the **SPOOLOPEN** command.

The system places your output records for the internal reader into a buffer in your address space. When this buffer is full, JES places the contents on the spool; later, JES retrieves the job from the spool. See "Identifying spool files" on page 363 for more information about the naming of spool files.

An alternative way of writing to the internal reader is to use an extrapartition TDQ. To do this, the TDQ must refer to a DD card similar to the following:

```
//ddname DD SYSOUT=(A,INTRDR)
```

CICS can be configured to perform a surrogate user ID check to verify if the user is authorized to submit a job with the user ID specified on the job card. If there is no USER option on the JOB statement, you can configure which user ID the job will run under. For more information about the security options, and how to configure them, see Security for submitting a JCL job to the internal reader.

Reading input spool files

The command sequence for reading a spool file is like that for creating one. You start with a SPOOLOPEN INPUT command that selects the file. Then you retrieve each record with a SPOOLREAD command. When the file is exhausted or you have read as much as required, you end processing with a SPOOLCLOSE command.

About this task

CICS provides you with a token to identify the particular file when you open it, just as it does when you open an output file, and you use the token on all subsequent commands against the file.

Like an output spool file, an input spool file is exclusive to the task that opened it. No other task can use it until the first one closes it. The file must be read in the same logical unit of work that opened it, and CICS closes it automatically at a SYNCPOINT command or at task end if the task does has not done so. However, you can close the file in such a way that your task (or another one) can read it again from the beginning.

In contrast to output files, a task can have only one spool file open for input at once. Moreover, *only one* CICS task can have a file open for input at any given time. This single-threading of input spool files has several programming implications:

- A task reading a spool file should keep it open for as little time as possible, and should close it explicitly, rather than letting CICS do so as part of end-of-task processing. You might want to transfer the file to another form of storage if your processing of individual records is long.
- If another task is reading a spool file, your SPOOLOPEN INPUT command fails with a SPOLBUSY condition. This is not an error; wait briefly and try again.
- If you read multiple input files, you should delay your task briefly between closing one and opening the next, to avoid monopolizing the input thread and locking out other tasks that need it.

A remote application must route any files intended for a CICS transaction to a specific user name at the system where CICS resides. See Figure 103 on page 360 for an example of a CP command used by a VM system to do this. The figure also shows the EXEC CICS SPOOL commands you use to retrieve the data.

The CICS transaction issues the SPOOLOPEN command, specifying the writer name on the USERID parameter and optionally the class of output within the writer name. The normal response is:

1. No input for this external writer.
2. The single-thread is busy.
3. The file is allocated to you for retrieval, and is identified by the “token” returned by CICS. The token must be included on every SPOOL command for retrieving the data set.

In cases (1) and (2), the transaction should retry the SPOOLOPEN after a suitable interval, by restarting itself.

In case (3), the transaction should then retrieve the file with SPOOLREAD commands, and proceed to SPOOLCLOSE as rapidly as possible to release the path for other users. This is especially important for **input** from JES because the input path is **single-threaded** . When there is more than one transaction using the interface, their files can be differentiated by using different writer names or different classes within a single writer name. Furthermore, you should ensure that the transactions either terminate or wait for a short period between SPOOLCLOSE and a subsequent SPOOLOPEN. If you do not do this, one transaction can prevent others from using the interface.

JES exits

Both JES2 and JES3 provide a way of screening incoming files. For JES2, the TSO/E Interactive Data Transmission Facility Screening and Notification exit is used. The JES3 equivalent is the Validate Incoming Netdata File exit.

You should review any use your installation makes of these exits to ensure that files that are to be read using the CICS interface to JES are correctly processed.

Identifying spool files

Input spool files are identified by the USERID and CLASS options on the SPOOLOPEN INPUT command.

On input, the USERID is the name of a JES external writer. An external writer is a name defined to JES at JES startup representing a group of spool files that have the same destination or processing. For files that JES processes itself, an external writer is typically associated with a particular hardware device, for example, a printer. The names of these writers are reserved for JES use.

For the transfer of files between applications, as occurs when a CICS task reads a spool file, the only naming requirement is that the receiver (the CICS task) knows what name the sender used, and that no other applications in the operating system of the receiver uses the same name for another purpose. To ensure that CICS tasks do not read spool files that were not intended for them, CICS requires that the external writer name that you specify matches its own z/OS Communications Server applid in the first four characters. Consequently, a job or system directing a file to CICS must send it to an external writer name that begins with the first four characters of the CICS applid.

JES categorizes the files for a particular external writer by a 1-character CLASS value. If you specify a class on your SPOOLOPEN INPUT command, you get the first (oldest) file in that class for the external writer you name. If you omit the class, you get the oldest file in any class for that writer. The sender assigns the class; A is used when the sender does not specify a class.

On output, you identify the destination of a SPOOL file with both a NODE and a USERID value. The NODE is the name of the operating system (for example, MVS , VM) as that system is known to z/OS Communications Server in the MVS system in which your CICS is executing).

The meaning of USERID varies with the operating system. In VM, it is a specific user; in MVS , it might be a JES external writer or another JES destination, a TSO user, or another job executing on that system. One such destination is the JES internal reader, which normally has the reserved name INTRDR. If you want to submit a job to an MVS system, you write a spool file to its internal reader. This file must contain all the JCL statements required to execute the job, in the same form and sequence as a job submitted through a card reader or TSO.

The following example shows a COBOL program using SPOOLOPEN for an internal reader. In this example, you must specify the NOCC option (to prevent use of the first character for carriage control) and use JCL record format.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OUTPUT-FIELDS.
03 OUTPUT-TOKEN PIC X(8) VALUE LOW-VALUES.
03 OUTPUT-NODE PIC X(8) VALUE 'MVSESA31'.
03 OUTPUT-USERID PIC X(8) VALUE 'INTRDR '.
03 OUTPUT-CLASS PIC X VALUE 'A'.
PROCEDURE DIVISION.
EXEC CICS SPOOLOPEN OUTPUT
TOKEN(OUTPUT-TOKEN)
USERID(OUTPUT-USERID)
NODE(OUTPUT-NODE)
CLASS(OUTPUT-CLASS)
NOCC
PRINT
NOHANDLE
END-EXEC.
```

Figure 105. An example of a COBOL program using SPOOL commands for an internal reader

OUTDESCR specifies a pointer variable to be set to the address of a field that contains the address of a string of parameters to the OUTPUT statement of MVS JCL.

The following example shows a COBOL program using the OUTDESCR operand:

```
WORKING-STORAGE SECTION.
01 F.
02 W-POINTER USAGE POINTER.
02 W-POINTER1 REDEFINES W-POINTER PIC 9(9) COMP.
01 RESP1 PIC 9(8) COMP.
01 TOKENWRITE PIC X(8).
01 ....
01 W-OUTDIS.
02 F PIC 9(9) COMP VALUE 43.
02 F PIC X(14) VALUE 'DEST(A20JES2 )'.
02 F PIC X VALUE ' '.
02 F PIC X(16) VALUE 'WRITER(A03CUBI)'.
02 F PIC X VALUE ' '.
02 F PIC X'11' VALUE 'FORMS(BILL)'.
LINKAGE SECTION.
01 DFHCOMMAREA PIC X.
01 L-FILLER.
02 L-ADDRESS PIC 9(9) COMP.
02 L-OUTDIS PIC X(1020).
PROCEDURE DIVISION.
EXEC CICS GETMAIN SET(W-POINTER) LENGTH(1024)
END-EXEC.
SET ADDRESS OF L-FILLER TO W-POINTER.
MOVE W-POINTER1 TO L-ADDRESS.
ADD 4 TO L-ADDRESS.
MOVE W-OUTDIS TO L-OUTDIS.
EXEC CICS SPOOLOPEN
OUTPUT
PRINT
RECORDLENGTH(1000)
NODE('*')
USERID('*')
OUTDESCR(W-POINTER)
TOKEN(TOKENWRITE)
RESP(RESP1)
NOHANDLE
END-EXEC.
EXEC CICS SPOOLWRITE..
.
```

Note:

1. It is essential to code a GETMAIN command.
2. L-FILLER is not a parameter passed by the calling program. The BLL for L-FILLER is then substituted by the SET ADDRESS. The address of the area obtained by using the GETMAIN request is then moved to the first word pointed to by L-FILLER being L-ADDRESS (thus pointing to itself). L-ADDRESS is then changed by plus 4 to point to the area (L-OUTDIS) just behind the address. L-OUTDIS is then filled with the OUTDESCRIPTOR DATA. Therefore W-POINTER points to an area that has a pointer pointing to the OUTDESCR data.

Examples of SPOOL commands

The following examples show you SPOOL commands using COBOL, PL/I, C, and ASSEMBLER.

COBOL

```
 DATA DIVISION.
WORKING-STORAGE SECTION.
01 RESP PIC 9(8) BINARY.
01 RESP2 PIC 9(8) BINARY.
01 TOKEN PIC X(8).
01 OUTLEN PIC S9(8) BINARY VALUE +80.
77 OUTPRT PIC X(80) VALUE
'SPOOLOPEN FUNCTIONING'.
01 PARMSPTR POINTER.
01 PARMS-AREA.
03 PARMSLEN PIC S9(8) BINARY VALUE 14.
03 PARMSINF PIC X(14) VALUE
'WRITER(MYPROG)'.
03 PARMADDR POINTER.
PROCEDURE DIVISION.
SET PARMSPTR TO ADDRESS OF PARMS-AREA
SET PARMADDR TO PARMSPTR
SET PARMSPTR TO ADDRESS OF PARMADDR
EXEC CICS SPOOLOPEN OUTPUT
NODE ('*')
USERID ('*')
RESP(RESP) RESP2(RESP2)
OUTDESCR(PARMSPTR)
TOKEN(TOKEN)
END-EXEC
EXEC CICS SPOOLWRITE
FROM(OUTPRT)
RESP(RESP) RESP2(RESP2)
FLENGTH(OUTLEN)
TOKEN(TOKEN)
END-EXEC
EXEC CICS SPOOLCLOSE
TOKEN(TOKEN)
RESP(RESP) RESP2(RESP2)
END-EXEC.
```

PL/I

```
DCL
RESP FIXED BIN(31),
RESP2 FIXED BIN(31),
TOKEN CHAR(8),
OUTLEN FIXED BIN(31) INIT(80),
OUTPRT CHAR(80) INIT('SPOOLOPEN FUNCTIONING'),
PARMADDR POINTER,
PARMSPTR POINTER;
DCL
1 PARMS,
2 PARMSLEN FIXED BIN(31) INIT(14),
2 PARMSINF CHAR(14) INIT('WRITER(MYPROG)')
ALIGNED;
PARMADDR=ADDR(PARMS);
PARMSPTR=ADDR(PARMADDR);
EXEC CICS SPOOLOPEN OUTPUT NODE('*') USERID('*')
TOKEN(TOKEN) OUTDESCR(PARMSPTR) RESP(RESP)
RESP2(RESP2);
EXEC CICS SPOOLWRITE FROM(OUTPRT) FLENGTH(OUTLEN)
RESP(RESP) RESP2(RESP2) TOKEN(TOKEN);
EXEC CICS SPOOLCLOSE TOKEN(TOKEN) RESP(RESP)
RESP2(RESP2);
```

C

```
 #define PARMS struct _parms
PARMS
{
int parms_length;
char parms_info[200];
PARMS * pArea;
};
PARMS ** parms_ptr;
PARMS parms_area;
char userid[8]= "*";
char node[8]= "*";
char token[8];
long rcode1, rcode2;
/* These lines will initialize the outdescr area and
set up the addressing */
parms_area.parms_info[0]= '\0';
parms_area.pArea = &parms_area;
parms_ptr = &parms_area.pArea;
/* And here is the command with ansi carriage controls
specified and no class*/
EXEC CICS SPOOLOPEN OUTPUT
NODE ( node )
USERID ( userid )
OUTDESCR ( parms_ptr )
TOKEN ( token )
ASA
RESP ( rcode1 )
RESP2 ( rcode2 );
```

ASSEMBLER

```
OUTPRT DC CL80'SPOOLOPEN FUNCTIONING'
PARMSPTR EQU 6
RESP DC F'0'
RESP2 DC F'0'
TOKEN DS 2F
OUTPTR DC A(PARMSLEN)
PARMSLEN DC F'14'
PARMSINF DC C'WRITER(MYPROG)'
LA PARMSPTR,OUTPTR
EXEC CICS SPOOLOPEN OUTPUT OUTDESCR(PARMSPTR)
NODE('*') USERID('*') RESP(RESP)
RESP2(RESP2) TOKEN(TOKEN)
EXEC CICS SPOOLWRITE FROM(OUTPRT)
TOKEN(TOKEN) RESP(RESP) RESP2(RESP2)
EXEC CICS SPOOLCLOSE TOKEN(TOKEN) RESP(RESP)
RESP2(RESP2)
```

Basic mapping support

Basic mapping support (BMS) is an application programming interface between CICS programs and terminal devices.

BMS is one of two sets of commands for this purpose. The other one, terminal control, is described in Terminal control.

For many applications, BMS has several advantages. First, BMS removes device dependencies from the application program. It interprets *device-independent* output commands and generates *device-dependent* data streams for specific terminals. It also transforms incoming device-dependent data into device-independent format. These features eliminate the need to learn complex device data streams. Allowing you to use the same program for various devices, because BMS determines the device information from the terminal definition, not from the application program.

Second, BMS separates the design and preparation of formats from application logic, reducing the impact of one on the other. Both of these features make it easier to write new programs and to maintain existing code.

BMS support levels

There are three levels of BMS support: minimum, standard, and full. Most installations use full BMS. If yours does, you can use all the features of BMS. If your installation uses minimum or standard BMS, note the features that require levels beyond yours.

The features are summarized here, and they are noted again whenever a facility that is not in minimum BMS is covered.

Minimum BMS

Minimum BMS supports all the basic functions for 3270 terminals, including everything described in our example and in the discussion of simple mapped output and mapped input.

Note: Minimum BMS has a substantially shorter path length than standard or full BMS. It is included in the larger versions and invoked as a "fast path" on commands that do not require function beyond what it provides. Specifically, it is used for SEND MAP and SEND CONTROL commands without the ACCUM, PAGING, SET, OUTPARTN, ACTPARTN, LDC, MSR, or REQID options, and for RECEIVE MAP commands, when your principal facility is a 3270 display or printer whose definition does not include outboard formatting. You can tell whether a particular BMS request used the fast path by looking at the CICS trace table. When fast path is used, the trace table contains duplicate entries for the BMS entry and exit code.

Standard BMS

Standard BMS adds:

- Support for terminals other than 3270s
- Text output commands
- Support for special hardware features: partitions, logical devices codes, magnetic slot readers, outboard formatting, and so on
- Additional options on the SEND command: NLEOM and FMHPARM

Standard BMS supports these terminals:

- Sequential terminals (composed of card readers, line printers, tape, or disk)
- TWX Model 33/35
- 1050
- 2740-1 (no buffer receive), 2740-2, 2741
- 2770
- 2780
- 2980, models 1, 2 and 4
- 3270
- 3600 (3601) LU
- 3650 (3653 and 3270 host conversational LUs)
- 3650 interpreter LU
- 3767/3770 interactive LU
- 3770 batch LU
- 3780
- LU type 4

Full BMS

Full BMS is required for:

- Sending BMS output other than directly to your own terminal (the SET and PAGING options, and BMS routing)
- Messages built cumulatively, with multiple BMS SEND commands (the ACCUM and PAGING options)

BMS is fully supported on CICS Transaction Server for z/OS and CICS Transaction Server for VSE.

BMS 3270 Intrusion Detection Service

This feature allows CICS to detect if a 3270 emulator has invalidly modified a protected field generated by a BMS map. You can opt into this capability with a feature toggle, as described in Specifying feature toggles.

Modification of protected fields might compromise the security of an application. This feature works together with the 3270 Intrusion Detection Service provided by IBM Communications Server. If configured, IBM Communication Server handles protection of all 3270 applications.

When both services are enabled, BMS generated 3270 data will be handled by CICS , and non-BMS 3270 data will be handled by IBM Communications Server. The advantage of enabling both is to ensure full coverage of all 3270 applications, but make use of BMS, to maximize performance and to enhance the information returned about any intrusion.

See 3270 Intrusion Detection Service in z/OS Communications Server: SNA Network Implementation Guide for the configuration and usage of 3270 IDS.

Note: If you need to configure BMS 3270 IDS to be specific about which applications or maps the service applies to, you can also use the URM DFHBMSX to configure BMS 3270 IDS. In general, this would only be necessary if an application made unusual use of the 3270 data stream and reported false hits.

Feature toggle for enabling this feature

`com.ibm.cics.bms.ids={true| false }`

Feature toggles for setting configuration options

com.ibm.cics.bms.ids.action={abend|ignore| log }
Specifies how CICS handles the detection of a protected field that is overwritten by a 3270 emulator. The values are as follows:

abend
CICS abends transaction ABSX.

ignore
CICS does not take any action.

log
CICS issues a DFHTF0200 message with the details of the overwrite. This is the default.

This configuration option sets the default that is passed to the URM DFHBMSX. If you want to configure the CICS action in a more granular way, use the URM DFHBMSX for configuration. The URM DFHBMSX overrides this configuration option.

com.ibm.cics.bms.ids.vtamignore={ true |false}
Specifies whether CICS informs IBM Communications Server that it is taking responsibility for checking the data when it is sending 3270 data related to a BMS request. This notifies IBM Communications Server's intrusion detection services that it can ignore the request. Use this option only under the guidance of IBM service.

A BMS output example

This example shows how BMS creates a formatted screen. It takes a list of data items from a program and displays them on the screen according to a predefined format.

BMS creates a formatted screen by merging variable data supplied by the program with constant data in the format (titles, labels for variable fields, default values for these fields). It builds the data stream for the terminal to which you are writing, to show this merged data in the designated screen positions, with

the proper attributes (color, highlighting, and so on). You do not have to know anything about the data stream, and you do not need to know much about the format to write the required CICS commands.

Note: For simplicity, this section is concerned with display screens, but most of it applies equally to printers. “Printing and spool files” on page 345 discusses differences between displays and printers and covers additional considerations that apply to printing. Furthermore, the examples and discussion assume a standard 3270 terminal because BMS is designed to support the features of the 3270. Other terminals are discussed in “Support for non-3270 terminals” on page 380.

You define the formats, called *maps* , separately from the programs that use them. Allowing you to reposition fields, change their attributes, and change the constant text without modifying your programs. If you add or remove variable data, you need to change the programs which use the affected fields.

The basics of how this works are explained by an atypically simple example. In real life, requirements are always more complex, but this gives you the essentials without too much confusing detail. There are more realistic and complete BMS examples among the CICS sample applications. These programs are included in source form on the CICS distribution tape. For more information, see Samples.

This example assumes that you need to write the code for a transaction used in a department store that checks a customer's balance before a charge sale is completed. The transaction is called a “quick check”, because all it does is check that the customer's account is open and that the current purchase is permissible, given the state of the account. The program for the *output* part of this transaction gets an account number, and produces the screen shown in Figure 106 on page 369 in response:

```
QCK Quick Customer Account Check
                   Account: 0000005
                   Name: Thompson Chris
                   Max charge: $500.00
```

Figure 106. Normal "quick check" output screen

The program uses the entered account number to retrieve the customer's record from the account file. From the information in this record, it completes the account number and customer name in the map, and computes the maximum charge allowed from the credit limit, outstanding balance, and purchases posted after the last billing period. If the amount comes out negative, you are supposed to show a value of zero and add an explanatory message. You also need to alert the clerk if the charge card is listed as lost, stolen, or canceled with a message as shown in Figure 107 on page 369:

```
QCK Quick Customer Account Check
                   Account: 0000005
                   Name: Thompson Chris
                   Max charge: $0.00
                   STOLEN CARD - SECURITY NOTIFIED
```

Figure 107. "Quick check" output screen with warning message

This message is to be highlighted, to draw the clerk's attention to it.

You must first define the screen. We explain how to do so for this particular map in “Creating the map” on page 371 . For the moment, however, let us assume that one of the outputs of this process is a data structure like the one in Figure 108 on page 370 . (We show the COBOL-coded version of the structure, because we are using COBOL to code our examples. However, BMS produces the structure in any language that CICS supports.) The map creation process stores this source code in a library from which you copy it into your program.

```
01 QCKMAPO.
02 FILLER PIC X(12).
02 FILLER PICTURE X(2).
02 ACCTNOA PICTURE X.
02 ACCTNOO PIC X(7).
02 FILLER PICTURE X(2).
02 SURNAMEA PICTURE X.
02 SURNAMEO PIC X(15).
02 FILLER PICTURE X(2).
02 FNAMEA PICTURE X.
02 FNAMEO PIC X(10).
02 FILLER PICTURE X(2).
02 CHGA PICTURE X.
02 CHGO PIC $,$$0.00
02 FILLER PICTURE X(2).
02 MSGA PICTURE X.
02 MSGO PIC X(30).
```

Figure 108. Symbolic map for "quick check"

The data names in this structure come from the map definition. You assign names to the fields that the program may have to change in any way. For our example, this category includes the fields where you display the account number, last name, first name, maximum charge, and explanatory message. It does not include any of the field labels or screen titles that never change, such as **Quick Customer Account Check** and **Account**.

Each field that you name on the screen generates several fields in the data structure, which are distinguished by a 1-character suffix added to the name you assigned in the map. Two appear here: the A suffix for the field attributes byte, and the O suffix for the output data. If we were creating a map to use special device features like color and highlighting, or were using the map for input as well as output, there would be many more. We tell you about these other fields in "Setting the display characteristics" on page 387 and "Receiving mapped data" on page 396.

The key fields for this particular exercise are the ones suffixed with O. These are where you put the data that you want displayed on the screen. You use the A subfields if you want to change how the data is displayed. In our example, we use MSGA to highlight the message if our customer is using a dubious card.

Here is an outline of the code that is needed for the example. You have to copy in the data structure (Figure 108 on page 370) produced by creating the map, and the COPY QCKSET statement in the third line does this. (Ordinarily, you would use a copy statement for the account record format too. We show it partly expanded here so that you can see its contents.)

```
 WORKING-STORAGE SECTION.
C COPY IN SYMBOLIC MAP STRUCTURE.
01 COPY QCKSET.
01 ACCTFILE-RECORD.
02 ACCTFILE-ACCTNO PIC S9(7).
02 ACCTFILE-SURNAME PIC X(15).
02 ACCTFILE-FNAME PIC X(10).
02 ACCTFILE-CREDIT-LIM PIC S9(7) COMP-3.
02 ACCTFILE-UNPAID-BAL PIC S9(7) COMP-3.
02 ACCTFILE-CUR-CHGS PIC S9(7) COMP-3.
02 ACCTFILE-WARNCODE PIC X.

PROCEDURE DIVISION.

EXEC CICS READ FILE (ACCT) INTO (ACCTFILE-RECORD) RIDFLD (CKNO)
... END-EXEC.
MOVE ACCTFILE-ACCTNO TO ACCTNOO.
MOVE ACCTFILE-SURNAME TO SURNAMEO.
MOVE ACCTFILE-FNAME TO FNAMEO.
COMPUTE CHGO = ACCTFILE-CREDIT-LIM - ACCTFILE-UNPAID-BAL
- ACCTFILE-CUR-CHGS.
IF CHGO < ZERO, MOVE ZERO TO CHGO
MOVE 'OVER CHARGE LIMIT' TO MSGO.
IF ACCTFILE-WARNCODE = 'S', MOVE DFHBMBRY TO MSGA
MOVE 'STOLEN CARD - SECURITY NOTIFIED' TO MSGO
EXEC CICS LINK PROGRAM('NTFYCOPS') END-EXEC.
EXEC CICS SEND MAP ('QCKMAP') MAPSET ('QCKSET') END-EXEC.
EXEC CICS RETURN END-EXEC.
```

Figure 109. BMS output example

Creating the map

BMS provides three assembler language macro instructions (**macros**) for defining maps. This method of map definition is still widely used, and this is the method that is described here.

However, there are also other products for creating maps which exploit the facilities of the display terminal to make the map creation process easier. They produce the same outputs as the BMS macros, generally with less programmer effort.

One of these is the Screen Definition Facility II (SDF II). SDF II allows you to build your screens directly from a display station, testing the appearance and usability as you go. You can find out more about SDF II in Screen Definition Facility II home site.

The three assembler macros used to define BMS maps are:

DFHMDF
defines an individual field on a screen or page.

DFHMDI
defines a single map as a collection of fields.

DFHMSD
groups single maps into a map set.

The explanation of this process begins by telling you how to define individual fields. Then it goes on to describe how to get from the fields to a complete map, and from a map to a map set (the assembly unit). BMS is designed principally for 3270-type terminals, although it supports nearly all types. See "The 3270 family of terminals" on page 274 for information on 3270 terms.

Defining map fields: DFHMDF

You should design the layout of your screen before you attempt to code any macros. After you have done that, you define each field on the screen (page) with a DFHMDF macro.

About this task

In it, you indicate:

- The position of the field on the screen
- The length of the field
- The default contents (unless you always intend to provide them in the program)
- The **field** display attributes, governing whether and what the operator can key into the field, whether the cursor stops there, the intensity of the characters, and the initial state of the modified data tag
- For some terminals, **extended** display attributes, such as color, underlining, highlighting
- The name by which you refer to the field in your program, if you ever modify its contents or attributes

Fields that are referenced by the application must be allocated field names. The length of the field name and the characters that can be used to form field names must conform to the following rules. (Note that these rules apply to currently supported compilers and assemblers.)

The characters used must be valid for names of assembler ordinary symbols. This character set consists of the alphabetic characters A - Z (upper or lowercase), $, #, @, numeric digits 0 - 9, and the underscore (_) character.

There is one exception to this rule. The hyphen (-) character can be used in field names if:

- The map set is only used by application programs written in COBOL.
- The map set is generated using the High Level Assembler.

The first character of the field name must be alphabetic, but the other characters can be any from the character set described previously.

In addition, the characters used in field names must conform to the character set supported by the programming language of the application using the map. For example, if the application language is

COBOL, you cannot use either the @ character or (in earlier versions) an underscore. Refer to the appropriate Language Reference manual for information about these character sets.

The DFHMDF macro allows the length of field names to be from one through 30 characters. DFHMDF derives additional variable names by appending one of several additional characters to the defined name to generate a symbolic description map. These derived names can therefore be up to 31 characters in length. The assembler, PL/I, and C languages all support variable names of at least 31 characters. However the COBOL language only allows up to 30 characters, which means that field names used in maps must not exceed 29 characters for COBOL applications. For example, the following field definition is valid for all languages except COBOL:

```
 ThisIsAnExtremelyLongFieldName DFHMDF
LENGTH=10,POS=(2,1)
```

and the following field definition is only valid for COBOL:

```
Must-Not-Exceed-29-Characters DFHMDF LENGTH=10,POS=(2,1)
"
```

Not all the options for field definition are described here; the rest are described in BMS macro DFHMDF.

Figure 110 on page 372 shows the field definitions for the map we considered in Figure 107 on page 369.

```
 DFHMDF
POS=(1,1),LENGTH=3,ATTRB=(ASKIP,BRT),INITIAL='QCK'
DFHMDF POS=(1,26),LENGTH=28,ATTRB=(ASKIP,NORM), X
INITIAL='Quick Customer Account Check'
DFHMDF POS=(3,1),LENGTH=8,ATTRB=(ASKIP,NORM),INITIAL='Account:'
ACCTNO DFHMDF POS=(3,13),LENGTH=7,ATTRB=(ASKIP,NORM)
DFHMDF POS=(4,1),LENGTH=5,ATTRB=(ASKIP,NORM),INITIAL='Name:'
SURNAME DFHMDF POS=(4,13),LENGTH=15,ATTRB=(ASKIP,NORM)
FNAME DFHMDF POS=(4,30),LENGTH=10,ATTRB=(ASKIP,NORM)
DFHMDF POS=(5,1),LENGTH=11,ATTRB=(ASKIP,NORM),INITIAL='Max charge:'
CHG DFHMDF POS=(5,13),ATTRB=(ASKIP,NORM),PICOUT='$,$$0.00'
MSG DFHMDF LENGTH=20,POS=(7,1),ATTRB=(ASKIP,NORM)
```

Figure 110. BMS map definitions

1. The **POS** (position) parameter indicates the row and column position of the field, relative to the upper left corner of the map, position (1,1). It must be present. Remember that every field begins with a field attributes byte; POS defines the location of this byte; the contents of the field follow immediately to the right.
2. The **LENGTH** option tells how many characters long the field is. The length does *not* include the attributes byte, so each field occupies one more column than its LENGTH value. In the case of the first field in our map, for example, the attributes byte is in row 1, column 1, and the display data is in columns 2-4. Fields can be up to 256 characters long and can wrap from one line to another. (Take care with fields that wrap if your map is smaller than your screen. See Outside the map in "Building the output screen" on page 391 for further information.)
3. The **ATTRB** (attributes) option sets the **field attributes** of the field, which we discussed in "3270 field attributes" on page 277. It is not required; BMS uses a default value of (ASKIP, NORM)—autoskip protection, normal intensity, modified data tag off—if you omit it. There are other options for each of the extended attributes, none of which was used in this map; these are described in "Setting the display characteristics" on page 387.
4. The **INITIAL** value for the field is not required either. You use it for label and title fields that have a constant value, such as 'QCK', and to assign a default value to a field, so that the program does not always have to supply a value.
5. The **PICOUT** option on the definition of the field CHG tells BMS what PICTURE clause to generate for the field. It lets you use the edit facilities of COBOL or PL/I directly, as you move data into the map. If you omit PICOUT, and also the numeric (NUM) attribute, BMS assumes character data. Figure 108 on page 370 shows the effects of the PICOUT option for CHG and, in the other fields, its absence. You can omit the LENGTH option if you use PICOUT, because BMS infers the length from the picture.

6. The **GRPNAME** and **OCCURS** options do not appear in our simple example, because they are for more complex problems. GRPNAME allows you to subdivide a map field within the program for processing, and OCCURS lets you define adjacent, like map fields so that you can treat them as an array in the program. These options are explained in "Using complex fields" on page 378 after some further information about maps.

Defining the map: DFHMDI

After all the fields on your map are defined, you tell BMS that they form a single map by preceding them with a DFHMDI macro.

About this task

This macro tells BMS:

- The name of the map
- The size, in rows and columns
- Where it appears on the screen (you can put several maps on one screen)
- Whether it uses 3270 extended display attributes and, if so, which ones
- The defaults for these extended attributes for fields where you have not assigned specific values on the DFHMDF macro
- Device controls associated with sending the map (such as whether to sound the alarm, unlock the keyboard)
- The type of device the map supports, if you intend to create multiple versions of the map for different types of devices (see "Device-dependent maps" on page 382)

The map name and size are the critical information about a DFHMDI macro but, for documentation purposes, you should specify your other options explicitly rather than letting them default. The DFHMDI macro for our example might be:

```
QCKMAP DFHMDI SIZE=(24,80),LINE=1,COLUMN=1,CTRL=ALARM
```

We have named the map QCKMAP. This is the identifier we use in SEND MAP commands. It is 24 lines long, 80 columns wide, and starts in the first column of the first line of the display. We have also indicated that we want to sound the alarm when the map is displayed.

Defining the map set: DFHMSD

You create a map using the DFHMSD macro which defines a map set

About this task

Maps are assembled in groups called map sets. Typically you group all the maps used by a single transaction or several related transactions. (We discuss reasons for grouping maps further in "Grouping maps into map sets" on page 377 .) A map set need not contain more than one map, incidentally, and in our simple example, the map set consists of just the "quick check" map.

One DFHMSD macro is placed in front of all the map definitions in the map set. It gives:

- The name of the map set
- Whether you are using the maps for output, input, or both
- Defaults for map characteristics that you did not specify on the DFHMDI macros for the individual maps
- Defaults for extended attributes that you did not specify in either the field or map definitions
- Whether you are creating physical or symbolic maps in the current assembly (see "Physical and symbolic map sets" on page 376)
- The programming language of programs that use the maps
- Information about the storage that is used to build the maps

Here's the DFHMSD macro we need at the beginning of our example:

```
QCKSET DFHMSD
TYPE=MAP,STORAGE=AUTO,MODE=OUT,LANG=COBOL,TIOAPFX=YES
```

This map set definition tells BMS that the maps in it are used only for output, and that the programs using them are written in COBOL. It assigns the name QCKSET to the map set. TIOAPFX=YES causes inclusion of a 12-byte “prefix” field at the beginning of each symbolic map (you can see the effect in the second line in Figure 108 on page 370). You always need this filler in command language programs and you should specify it explicitly, as the default is sometimes omission. MAP and STORAGE are explained in “Sending BMS mapped output” on page 385.

You need another DFHMSD macro at the end of your map definitions, to tell the assembler that it has reached the end of last map in the map set:

```
 DFHMSD TYPE=FINAL
```

Writing BMS macros

Because a BMS macro is an assembler language statement, you must follow assembler language syntax rules.

The following set of rules work, but they are more restrictive than the actual rules. For the full set of assembler language syntax rules, see High Level Assembler Language Reference.

1. Start names in column 1. Map and map set names can be up to seven characters long. The maximum length for field names (the DFHMDF macro) depends on the programming language. BMS creates labels by adding one-character suffixes to your field names. These labels must not be longer than the target language allows, because they get copied into the program. Hence the limit for a map field name is 29 characters for COBOL, 30 for Pl/I and Assembler H, and 7 for Assembler F. For C and C++, it is 30 if the map is copied into the program as an internal data object, and six if it is an external data object (see “Acquiring and defining storage for the maps” on page 385 for more information about copying the map).
2. Start the macro identifier in column 10, or leave one blank between it and the name if the name exceeds eight positions. For field definitions, the identifier is always DFHMDF; for map definitions, DFHMDI; and for the map set macros that begin and end the map set, DFHMSD.
3. The rest of the field description consists of keywords (like POS for the position parameter) followed by values. Sometimes a keyword does not have a value, but if it does, an equals sign (=) always separates the keyword from its value.
4. Leave one blank after your macro identifier and then start your keywords. They can appear in any order.
5. Separate keywords by one comma (no blanks), but do not put a comma after the last one.
6. Keywords can extend through column 71. If you need more space, stop after the comma that follows the last keyword that fits entirely on the line and resume in column 16 of the next line.
7. Initial values (the INITIAL, XINIT, and GINIT keywords) are exceptions to the rule, because they might not fit even if you start on a new line. Except when double-byte characters are involved, you can split them at any point after the first character of the initial value itself. When you split in this way, use all of the columns through 71 and continue in column 16 of the next line. Double-byte character set (DBCS) data is more complicated to express than ordinary single-byte (SBCS) data. See Step “12” on page 375 if you have DBCS initial values.
8. Surround initial values by single quote marks. If you need a single quote *within* your text, use two successive single quotes (the assembler removes the extra one). Ampersands also have special significance to the assembler, and you use the same technique: use two ampersands where you want one, and the assembler removes the extra.
9. If you use more than one line for a macro, put a character (any one except a blank) in column 72 of all lines *except the last*.

10. If you want comments in your map, use comment lines between macros, not among the lines that make up a single macro. Comment lines have an asterisk in column 1 and a blank in column 72. Your comments can appear anywhere among columns 2-71.
11. Use uppercase only, except for values for the INITIAL parameter and in comments.
12. For initial values containing DBCS. If you have initial data that is *entirely* DBCS, use the GINIT keyword for your data and specify the keyword PS=8 as well. If your data contains both DBCS and SBCS characters, that is, if it is mixed, use INITIAL and specify SOSI=YES. (We need to explain a third alternative, XINIT, because you may find it in code you are maintaining. You should use GINIT and INITIAL if possible, however, as XINIT is more difficult to use and your data is not validated as completely. XINIT can be used for either pure or mixed DBCS. XINIT with PS=8 follows the rules for GINIT, and XINIT with SOSI=YES follows those for INITIAL (mostly, at least). The main difference is that you express your data in hexadecimal with XINIT, but you use ordinary characters for GINIT and INITIAL.)

    This is how you write DBCS initial values:

    - You enclose your data with single quotes, as you do with the ordinary INITIAL parameter.
    - You use two ordinary characters for each DBCS character in your constant (two pairs of hexadecimal digits with XINIT) and one for each SBCS character (one pair with XINIT).
    - You bracket each DBCS character string with a shift-out (SO) character immediately preceding and a shift-in (SI) character immediately after. SO is hexadecimal X'0E', which appears as '<' on most keyboards, and SI is X'0F' ('>'). (XINIT with PS=8 is an exception; the SO/SI brackets are implied and you do not key them.) For example, all of these define the same initial value, which is entirely DBCS. (Ignore the LENGTH values for the moment; we explain those in a moment.)

      ```
       GINIT='<D1D2D3D4D5>',PS=8,LENGTH=10
      INITIAL='<D1D2D3D4D5>',SOSI=YES,LENGTH=12
      XINIT='C4F1C4F2C4F3C4F4C4F5',PS=8,LENGTH=10
      XINIT='0EC4F1C4F2C4F3C4F4C4F50F',SOSI=YES,LENGTH=12
      ```

    - SBCS and DBCS sequences can follow each other in any combination with INITIAL (and XINIT with SOSI=YES). If we add 'ABC' in front of the DBCS string in the previous example, and 'def' following the string, we have:

      ```
      INITIAL='ABC<D1D2D3D4D5>def',SOSI=YES,LENGTH=18
      XINIT='C1C2C30EC4F1C4F2C4F3C4F4C4F50F848586',SOSI=YES,LENGTH=18
      ```

    - To calculate the length of your initial value, count two for each DBCS character and one for each SBCS character, whether you express them in ordinary characters or hexadecimal pairs. With GINIT (and XINIT with PS=8), you do not count the SO and SI characters, but with INITIAL (and XINIT with SOSI=YES), you add one for each SO and for each SI. (Note the different LENGTH values for the same constants in the previous examples.) In all cases, your LENGTH value must not exceed 256.
    - For GINIT and INITIAL, if your constant does not fit on one line, you use “extended” continuation rules, which are a little different from the ones described earlier. With extended continuation, you can stop after any full character (SBCS character, DBCS pair, or the SI ending a DBCS string) within your initial value. If you are in the middle of a DBCS string, add an SI (the SOs and SIs on one line must balance). Then fill out the line through column 72 with a continuation character. Any character will do, so long as it is different from the last meaningful character on the line.

      If you have stopped within a DBCS string, put an SO character in column 16 of the next line and resume in 17; otherwise just resume in 16, thus:

      ```
      GXMPL1 DFHMDF
      POS=(02,21),LENGTH=20,PS=8,GINIT='<D1D2D3D4D5D6>******
      <D7D8D9D0>'
      IXMPL1 DFHMDF
      POS=(02,21),LENGTH=23,PS=8,INITIAL='abc<D1D2D3D4>ABC**
      DEFGHIJ'
      ```

      You cannot use extended continuation with XINIT; use the rules described in Step “7” on page 374.

- If your LENGTH specification exceeds the length of the initial value you provide, the value is filled out on the right with DBCS blanks to your LENGTH value if you have used GINIT (or XINIT with PS=8). If you have used INITIAL, the fill character is an SBCS blank if the last part of the constant was SBCS, a DBCS blank if the last part was DBCS. If you use XINIT with SOSI=YES, the fill character is always an SBCS blank.

Assembling the map

Before you start coding, you must assemble and link edit your map set.

About this task

You typically have to assemble twice, to create the map set in two different forms. The TYPE option in the DFHMSD macro tells the assembler the form to produce in any particular assembly.

Physical and symbolic map sets
You can use the TYPE=MAP assembly, followed by a link-edit to produce a load module called the physical map set. The TYPE=DSECT assembly is used to produce a series of data structures, collectively called the symbolic map set.

The physical map set contains format information in encoded form. CICS uses it at execution time for constant fields and to determine how to merge in the variable data from the program.

The physical map set normally is stored in the same library as your application programs, and it requires a MAPSET resource definition within CICS, just as a program requires a PROGRAM resource definition.

The output of a TYPE=DSECT assembly is a series of data structures, coded in the source language specified in the LANG option. There is a structure for each map used for input, called the symbolic input map, and one for each map used for output, called the symbolic output map.

Symbolic map sets are used at compile (assembly) time. You copy them into your program, and they allow you to refer to the fields in the maps by name and to pass the variable data in the form dictated by the physical map set. We have already shown you an example of a symbolic output map in COBOL (see Figure 108 on page 370) and used it in the example code. Symbolic map sets are typically stored in the library your installation defines for source code that gets copied into programs. Member names are typically the same as the map set names, but they need not be.

You need the TYPE=DSECT assembly before you compile or assemble your program. You can defer the TYPE=MAP assembly and link-edit until you are ready to test, because the physical map set is not used until execution time. However, because you must do both eventually, many installations provide a cataloged procedure to do this automatically; the procedure copies the source file for the map set and processes it once using TYPE=MAP and again using TYPE=DSECT. You also can use the SYSPARM option in your assembler procedure to override the TYPE value in a particular assembly. See High Level Assembler Language Reference for a description of SYSPARM with map assemblies, and “Installing map sets and partition sets” on page 706 for more information about assembling maps.

Note:

1. The fact that symbolic map sets are coded in a specific language does not prevent you from using the same map in programs coded in different languages. You assemble with TYPE=DSECT for each LANG value you need, taking care to store the outputs in different libraries or under different names. The LANG value does not affect the TYPE=MAP assembly, which need be done only once.
2. If you modify an existing map in a way that affects the symbolic map, you *must recompile (reassemble)* any programs using it, so that the compilation uses the symbolic structure that corresponds to the new physical structure. Changes to unnamed map fields do not affect the symbolic map, but addition, deletion, rearrangement, and length changes of named fields do.Rearrangement refers to the DFHMDF macros; the order of the fields on the screen does not affect the symbolic map, although it is more efficient to have the DFHMDF macros in same order as the fields on the screen. So make changes to the DSATTS option in the map definition—this option states the extended attributes you can want to change by program. It is always safest to recompile.

The SDF II alternative
If you use the IBM licensed program Screen Definition Facility II the assembly and link-edit steps are not required. SDF II creates both the symbolic map set and the physical map set in the final step of the interactive map creation process.

SDF II can run under either MVS (Program 5665-366) or VM (5664-307). See Screen Definition Facility II home site.

Grouping maps into map sets
Because they are assembled together, all of the physical maps in a map set constitute a single load module. BMS gains access to all of them with a single load request, issued on the first use of the map set in the task.

No further loads are required unless you request a map in a different set, in which case BMS releases the old map set and loads the new one. If you go back to the first map set subsequently, it gets loaded again. Loading and deleting does not necessarily involve I/O, but you should consider the path length when grouping your maps into map sets. Generally, if maps are used together, they should be in the same map set; those not used together should be in different map sets.

The limit to the number of maps in a set is 9 998, but you should also keep the size of any given load module reasonable. So you might keep infrequently used maps separate from those normally used in a given process.

Similarly, all of the symbolic maps for a map set are in a single symbolic structure. This affects the amount of storage you need while using the maps, as explained in “BASE and STORAGE options” on page 386 . Depending on the programming language, it also may affect high-level names, and this may be a reason for separating or combining maps as well.

The Application Data Structure (ADS)
The symbolic map generated by the BMS macros is also known as the application data structure (ADS).

Physical maps produced by CICS Transaction Server for z/OS Version 1.2 and later also include an ADS descriptor in the output load module. This is provided to allow interpretation of the BMS Application Data Structure (the structure used by the application program for the data in SEND and RECEIVE MAP requests), without requiring your program to include the relevant DSECT or copybook at compile time.

The ADS descriptor contains a header with general information about the map, and a field descriptor for every field that appears in the ADS (corresponding to every named field in the map definition macro). It can be located in the mapset from an offset field in DFHMAPDS.

The ADS descriptor is generated for all maps. You can choose to map the long or short form of the ADS by specifying the DSECT=ADS|ADSL option. The default is ADS,the short (normal) form. The long form of the ADS aligns all fields on 4-byte boundaries and is required for some interfaces with other products, such as IBM MQ.

Map sets generated with CICS releases before CICS Transaction Server for z/OS Version 1.2 do not contain the ADS descriptor.

The format of the ADS descriptor is contained in the following copybooks:

Table 45. ADS descriptor copybooks

Language	**Copybook**
Assembler	DFHBRARD
C	DFHBRARH
PL/I	DFHBRARL
COBOL	DFHBRARO

For further information about the ADS descriptor, see Developing for external interfaces.

If you need to reassemble maps but have no access to the source, a utility program DFHBMSUP is provided to re-create BMS macro source from a mapset load module.

For more information about DFHBMSUP, see BMS macro generation utility (DFHBMSUP).

Using complex fields

BMS provides two options to the DFHMDF macro for defining complex fields; GRPNAME and OCCURS.

About this task

The symbolic maps we have shown so far consisted of a fixed set of fields for each named map field (the A and O subfields, and so on, in Figure 108 on page 370). Such fields are the most common, but BMS provides two options for field definition which produce slightly different structures, to account for two common programming situations.

Composite fields: the GRPNAME option

The GRPNAME option is used to generate symbolic storage definitions and to combine specific fields under one group name. The same group name must be specified for each field that is to belong to the group.

About this task

Sometimes, you have to refer to subfields within a single field on the display. For example, you can have a date field that appears on the screen as follows:

```
 03-17-92
```

It is one field on the screen (with one attributes byte, just before the digit “0”), but you must be able to manipulate the month, day, and year components separately in your program.

You can do this with a “group field”, using the GRPNAME option of the DFHMDF macro. To create one, you code a DFHMDF macro for each of the component subfields; each definition names the same group field in the GRPNAME option. To define the previous date as a group field starting at the beginning of line 10, for example, we would write:

```
MO DFHMDF POS=(10,1),LENGTH=2,ATTRB=BRT,GRPNAME=DATE
SEP1 DFHMDF POS=(10,3),LENGTH=1,GRPNAME=DATE,INITIAL='-'
DAY DFHMDF POS=(10,4),LENGTH=2,GRPNAME=DATE
SEP2 DFHMDF POS=(10,6),LENGTH=1,GRPNAME=DATE,INITIAL='-'
YR DFHMDF POS=(10,7),LENGTH=2,GRPNAME=DATE
```

These definitions produce the following in the symbolic output map:

```
 02 DATE.
03 FILLER PICTURE X(2).
03 MOA PICTURE X.
03 MOO PIC X(2).
03 SEP1 PIC X(1).
03 DAO PIC X(2).
03 SEP2 PIC X(1).
03 YRO PIC X(2).
```

Several rules must be observed when using a group field:

- There is only one attributes byte; it precedes the whole group field and applies to the whole field. You specify it just once, on the DFHMDF macro for the first subfield, MO here.
- Because there is only one attributes byte, the cursor behaves as if the group field were a single field. In our example, the cursor does not move from the last position of month to the first of day, or day to year, skipping over the hyphens. This is because the group really is a single field as far as the hardware goes; it is subdivided only for program access to the component subfields.
- Although subfields after the first do not have an attributes byte, you define the POS option as if they did, as shown in the example. That is, POS points to one character before the subfield begins, and can overlap the last character of the previous subfield, as occurs in our example.

- Although all the component subfields are adjacent in this example, they do not have to be. There can be gaps between the subfields, provided you do not define any other field in the gap. The group field spans all the columns from its first subfield to its last, and you must put the component DFHMDF macros in the order the subfields appear on the screen. The group ends with the first DFHMDF macro that does not specify its name.
- You must assign a field name to every subfield, even if you do not intend to refer to it (as we did in the SEP1 and SEP2 subfields in the example).
- You cannot use the OCCURS option (explained in the next section) for a group field or any of its components.

Repeated fields: the OCCURS option
The OCCURS option specifies that the indicated number of entries for the field are to be generated in a map, and that the map definition is to be generated in such a way that the fields are addressable as entries in a matrix or an array.

About this task

Sometimes a screen contains a series of identical fields that you want to treat as an array in your program. Suppose, for example, that you need to create a display of 40 numbers, to be used when a clerk assigns an unused telephone number to a new customer. (The idea is to give the customer some choice.) You also want to highlight numbers which have been in service recently, to warn the customer of the possibility of calls to the previous owner.

You can define the part of your screen which shows the telephone numbers with a single field definition:

```
TELNO DFHMDF POS=(7,1),LENGTH=9,ATTRB=NORM,OCCURS=40
```

This statement generates 40 contiguous but separate display fields, starting at position (7,1) and proceeding across the rows for as many rows as required (five, in our case). We have chosen a length that (with the addition of the attributes byte) divides the screen width evenly, so that our numbers appear in vertical columns and are not split across row boundaries. The attributes you specify, and the initial value as well, apply to each field.

The description of these fields in the symbolic map looks like this in COBOL:

```
 02 TELNOG OCCURS 40.
03 FILLER PICTURE X(2).
03 TELNOA PICTURE X.
03 TELNOO PIC X(9).
```

This structure lets you fill the map from an array in your program (or any other source) as follows:

```
 PERFORM MOVENO FOR I FROM 1 THROUGH 40.
...
MOVENO.
MOVE AVAIL-NO (I) TO TELNOO (I).
IF DAYS-SINCE-USE (I) < 90, MOVE DFHBMBRY to TELNOA (I).
```

(DFHBMBRY is a CICS -supplied constant for setting the field intensity to bright; we explain more in “Attribute value definitions: DFHBMSCA” on page 389 .)

Labels for OCCURS fields vary slightly for the different languages that CICS supports, but the function is the same.

Each element of an array created by the OCCURS option is a single map field. If you need to repeat a series of fields (an array of structures, in other words), you cannot use OCCURS. To use such an array in a program, you must define all the fields individually, without OCCURS, to produce the necessary physical map. Then you can modify the resulting symbolic map, replacing the individual field definitions with an array whose elements are the structure you need to repeat. You must ensure that the revised symbolic map has the same field structure as the original. An alternative is to use SDF II, which allows you to define such an array directly.

Block data

BMS provides an alternate format for the symbolic map, called block data format, that can be useful in specific circumstances. In block data format, the symbolic output map is an image of the screen or page going to the terminal.

The symbolic output map has the customary field attributes (A) and output value (O) subfields for each named map field, but the subfields for each map field are separated by filler fields such that their displacement in the symbolic map structure corresponds to their position on the screen. There are no length subfields, and symbolic cursor positioning is unavailable as a consequence.

For example, the symbolic map for the "quick check" screen in Figure 107 on page 369 would look like this in block data format (assuming a map 80 columns wide). Compare this with the normal "field data". format (in Figure 108 on page 370) from the same map definition.

```
01 QCKMAPO.
02 FILLER PIC X(12). <---TIOAPFX still present
02 FILLER PICTURE X(192). <---Spacer
02 ACCTNOA PICTURE X. <---Position (3,13)
02 ACCTNOO PIC X(7).
02 FILLER PICTURE X(72). <---Spacer
02 SURNAMEA PICTURE X. <---Position (4,13)
02 SURNAMEO PIC X(15).
02 FNAMEA PICTURE X. <---Position (4,30),
02 FNAMEO PIC X(10).
02 FILLER PICTURE X(52). <---Spacer
02 CHGA PICTURE X. <---Position (5,13)
02 CHGO PIC $,$$0.00
02 FILLER PICTURE X(139). <---Spacer
02 MSGA PICTURE X. <---Position (7,1).
02 MSGO PIC X(30).
```

Figure 111. Symbolic map for "quick check" in block data format

You can set only the field attributes in the program; BMS ignores the DSATTS option in the map and does not generate subfields for the extended attributes in block data format. You can use block data for input as well. The input map is identical in structure to the output map, except that the flag (F) replaces the field attributes (A) subfield, and the input (I) replaces the output (O) subfield, as in field format.

Block data format can be useful if the application program has built or has access to a printer page image which it needs to display on a screen. For most situations, however, the normal field data format provides greater function and flexibility.

Support for non-3270 terminals

Support for non-3270 terminals requires standard BMS; minimum BMS supports only 3270 displays and printers.

Minimum

BMS supports only 3270 displays and printers. This category includes the 3178, 3290, 8775 and 5520, LU type 2 and LU type 3 devices, and any other terminal that accepts the 3270 data stream. For a full list, see IBM 3270 Data Stream Programmers Reference.

Standard

BMS expands 3270 support to SCS printers (3270 family printers *not* using the 3270 data stream) and all of the terminal types listed in Table 46 on page 383. See "Non-3270 CICS printers" on page 350 for more information about BMS and SCS datastreams.

Because of functional differences among these terminal types, it is not possible to make BMS work in exactly the same way for each of them. The sections which follow outline the limitations in using BMS on devices which lack the hardware basis for certain features.

Output considerations for non-3270 devices
Because BMS separates the device-dependent content of the output data stream from the logical content, there are only a few differences between 3270 and non-3270 devices that you need to consider in creating BMS output.

The primary difference between 3270 and non-3270 devices is that the 3270 is *field-oriented* , and most others are not. Consequently, there are neither field attributes nor extended attributes associated with output fields sent to non-3270 terminals. BMS can position the output in the correct places, field by field, but the field structure is not reflected in the data stream. BMS can even emulate some field attributes on some terminals (it may underline a highlighted field, for example), but there is no modified data tag, no protection against keying into the field, and so on.

If you specify attributes on output that the terminal does not support, BMS ignores them. You do not need to worry about them, provided the output is understandable without the missing features.

Differences on input
The absence of field structure has more impact on input operations than on output operations, because many of the features of BMS depend on the ability to read—by field—only those fields that were modified. If the hardware does not provide the input field-by-field with its position on the screen, you must provide equivalent information.

You can do this in either of two ways. The first is to define a field-separator sequence, one to four characters long, in the FLDSEP option of the map definition. You place this sequence between each field of your input and supply the input fields in the same order as they appear on the screen or page. You must supply every field on the screen, up to the last one that contains any data. If there is no input in a field, it can consist of only the terminating field-separator sequence. On hardcopy devices, input cannot overlay the output because of paper movement. On displays that emulate such terminals, the same technique is generally used. Input fields are entered in an area reserved for the purpose, in order, separated by the field-separator sequence.

The second method is to include control characters in your input. If you omit the FLDSEP option from your map, BMS uses control characters to calculate the position of the data on the “page” and maps it accordingly. The control characters that BMS recognizes are:

NL	new line	X'15'
IRS	interchange record separator	X'1E'
LF	line feed	X'25'
FF	form feed	X'0C'
HT	horizontal tab	X'05'
VT	vertical tab	X'0B'
CR	carriage return	X'0D'
RET	return on the TWX	X'26'
ETB	end text block	X'26'
ESC	escape, for 2780	X'27'

When you read data of this kind with a RECEIVE MAP command, there are some differences from true 3270 input:

- The flag byte (F subfield) is not set; it contains a null. You cannot determine whether the operator erased the field or whether the cursor remained in the field.
- You cannot preset a modified data tag on output to ensure that a field is returned on input.

Special options for non-3270 terminals
BMS provides some additional formatting options for non-3270 devices, to take advantage of device features that shorten the data stream.

These options include:

- Vertical and horizontal tabs. You can position your output with horizontal and vertical tab orders if the device supports them. The tab characters are defined by the HTAB and VTAB options in the map set definition. When you want to position to the next horizontal tab, you include the HTAB character in your data; you position to the next vertical tab by supplying the VTAB character in your data. BMS translates these characters to the tab sequence required by your particular principal facility.

  Before you use tabs in BMS output, your task or some earlier task at the same terminal must have set the tabs in the required positions. This is usually done with a terminal control SEND command, described in "Using data transmission commands" on page 260.
- Outboard formatting. Some logical units can store format information and participate in the formatting process. This allows BMS to send much less data (essentially the symbolic map contents) and delegate the work of merging the physical and symbolic maps to the logical unit. See "Outboard formatting" on page 437 for details.
- NLEOM (new line, end of message). Standard BMS also gives you the option of requesting that BMS format your output with blanks and new-line (NL) characters rather than 3270 buffer control orders. This technique gives you more flexibility in page width settings on printers, as explained in "NLEOM option" on page 349.

Device-dependent maps

Because the position, default attributes, and default contents of map fields appear only in the physical map and not in the symbolic map, you can use a single program to build maps that contain the same variable information but different constant information in different arrangements on the screen.

Device-dependent maps are very convenient if the program you are writing must support multiple devices with different characteristics. You do this by defining multiple maps with the same names but different attributes and layout, each with a different suffix.

Suppose, for example, that some of the clerks using the "quick update" transaction use 3270 Model 2s (as we have assumed until now), and the rest use a special-purpose terminal that has only 3 rows and 40 columns. The format we designed for the big screen will not do for the small one, but the information will fit if we rearrange it:

```
QUP Quick Account Update:
                  Current charge okay; enter next
                  Acct: _______ Charge: $ _______
```

Figure 112. "Quick update" for the small screen

We need the following map definition:

```
QUPSET DFHMSD
TYPE=MAP,STORAGE=AUTO,MODE=INOUT,LANG=COBOL,SUFFIX=9
QUPMAP DFHMDI SIZE=(3,40),LINE=1,COLUMN=1,CTRL=FREEKB
DFHMDF POS=(1,1),LENGTH=24,ATTRB=(ASKIP,BRT), X
INITIAL='QUP Quick Account Update'
MSG DFHMDF LENGTH=39,POS=(2,1),ATTRB=(ASKIP,NORM)
DFHMDF POS=(3,1),LENGTH=5,ATTRB=(ASKIP,NORM), X
INITIAL='Acct:'
ACCTNO DFHMDF POS=(3,11),LENGTH=6,ATTRB=(UNPROT,NUM,IC)
DFHMDF POS=(3,18),LENGTH=1,ATTRB=(ASKIP),INITIAL=' '
DFHMDF POS=(3,20),LENGTH=7,ATTRB=(ASKIP,NORM),INITIAL='Charge:'
CHG DFHMDF POS=(3,29),LENGTH=7,ATTRB=(UNPROT,NORM),PICIN='$$$$0.00'
DFHMDF POS=(3,37),LENGTH=1,ATTRB=(ASKIP),INITIAL=' '
DFHMSD TYPE=FINAL
```

Figure 113. Map definition

The symbolic map set produced by assembling this version of the map is identical to the one shown in “An input-output example” on page 396 , because the fields with names have the same names and same lengths, and they appear in the same order in the map definition. (They do not need to appear in the same order on the screen, incidentally; you can rearrange them, provided you keep the definitions of named fields in the same order in all maps.) You only need one copy of the symbolic map and you can use the same code to build the map.

CICS will select the physical map to use from the value coded in the ALTSUFFIX option of the TYPETERM resource definition for the terminal from which the transaction is being run. You also need to specify SCRNSZE(ALTERNATE) in the transaction PROFILE resource definition. For information about the TYPETERM and PROFILE resource definition, see RDO resources.

You might use this technique to distinguish among standard terminals used for special purposes. For example, if an application were used by both English and French speakers, you could create two sets of physical maps, one with the constants in French and the other in English. You would assign each a suffix, and specify the English suffix as the ALTSUFFIX value in the definitions of the English terminals and the French suffix for French terminals. Transactions using the map would point to a PROFILE that specified alternate screen size. Then when you sent the map, BMS would pick the version with the suffix that matched the terminal (that is, in the appropriate language).

Another way to provide device dependent maps is to allow BMS to generate a suffix based on the terminal type, and select the physical map to use to match the terminal in the current execution when you issue SEND MAP or RECEIVE MAP.

Device dependent support
Device dependent support (DDS) is an installation feature that enables device-dependent maps.

When you assemble your map sets, you specify the type of terminal the maps are for in the TERM option. This causes the assembler to store the physical map set under the MAPSET name suffixed by the character for that type of terminal. You also can use JCL or the link-edit NAME statement to control the member name under which a map set is stored. When you issue SEND MAP or RECEIVE MAP with DDS active, BMS adds a 1-character suffix to the name you supply in the MAPSET option. It chooses the suffix based on the definition of your terminal, and thus loads the physical map that corresponds to the terminal for any given execution.

BMS defines the suffixes used for the common terminal types. A 3270 Model 2 with a screen size of 24 rows and 80 columns is assigned the letter 'M,' for example. The type is determined from the TYPETERM definition if it is one of the standard types shown in Table 46 on page 383.

Table 46. Terminal codes for BMS

Code	**Terminal or logical unit**
A	CRLP (card reader input, line printer output)
B	Magnetic tape
C	Sequential disk
D	TWX Model 33/35
E	1050
F	2740-1, 2740-2 (without buffer receive)
G	2741
H	2740-2 (with buffer receive)
I	2770
J	2780
K	3780
L	3270-1 displays (40-character width)

Table 46. Terminal codes for BMS (continued)	
Code	**Terminal or logical unit**
M	3270-2 displays (80-character width), LU type 2s
N	3270-1 printers
O	3270-2 printers, LU type 3s
P	All interactive LUs, 3767/3770 Interpreter LU, 3790 full function LU, SCS printer LU
Q	2980 Models 1 and 2
R	2980 Model 4
U	3600 (3601) LU
V	3650 Host Conversational (3653) LU
W	3650 Interpreter LU
X	3650 Host Conversational (3270) LU
Y	3770 Batch LU, 3770 and 3790 batch data interchange LUs, LU type 4s
blank	3270-2 (default if TERM omitted)

An installation can also define additional terminal types, such as the miniature screen previously described. The system programmer does this by assigning an identifier to the terminal type and specifying it in the ALTSUFFIX option of the TYPETERM definition for the terminals. When you create a map for such a terminal, you specify this identifier in the SUFFIX option, instead of using the TERM option. Transactions using the map must also point to a PROFILE that specifies alternate screen size, so that ALTSUFFIX is used.

With DDS, the rules BMS uses for selecting a physical map are:

- BMS adds the ALTSUFFIX value in the terminal definition to your map set name, provided that definition specifies both ALTSUFFIX and ALTSCREEN, and provided that the screen size for the transaction is the alternate size (either because the transaction PROFILE calls for alternate size, or because the default and alternate sizes are the same).
- If these conditions are not met, or if BMS cannot find a map with that suffix, it attempts to find one with the suffix that corresponds to the terminal type in the terminal definition.
- If BMS cannot find that map either, it looks for one with no suffix. (A blank suffix indicates an all-purpose map, suitable for any terminal that might use it.)

Without DDS, BMS always looks first (and only) for an unsuffixed map.

Device-dependent support is an installation option for BMS, set by the system programmer in the system initialization table. Be sure that it is included in your system before taking advantage of it; you should know whether it is present, even if you are supporting only one device type.

With DDS in the system, there is an efficiency advantage in creating suffixed map sets, even if you are supporting only one device type, because you prevent BMS from attempting to load a map set that does not exist before defaulting to the universal one (the blank suffix).

Without DDS, on the other hand, it is unnecessary to suffix your maps, because BMS looks for the universal suffix (a blank) and fails to locate suffixed maps.

Finding out about your terminal
Because of the overall design of BMS, and device-dependent support in particular, you generally do not need to know much about your terminal to format for it. However, if you need to know the characteristics of your principal facility, you can use the ASSIGN and INQUIRE commands.

You can tell, for example, whether your terminal supports a particular extended attribute, what national language is in use, screen size, and so on. This type of information applies whether you are using BMS or

terminal control to communicate with your terminal. You need it more often for terminal control, and so the options that apply are described in “Finding out about your terminal” on page 265.

There are also ASSIGN options specific to BMS, but you need them most often when you use the ACCUM option, and so these are described later, in “ASSIGN options for cumulative processing” on page 413.

Sending BMS mapped output

Before you can send BMS mapped output, you must perform the following steps to move the output data into the map structure.

About this task

When you have assembled your symbolic map set, you are ready to code. The example used in “Creating the map” on page 371 describes how you get data from an application program to a map. This process is discussed in greater detail now, describing all the steps that must be performed, and telling you more about the options you have.

You must perform the following steps to produce mapped output:

Procedure

1. Acquire storage in which to build the map.
2. Copy the symbolic map set so that it defines the structure of this storage.
3. Initialize it.
4. Move the output data into the map structure.
5. Set the field attributes.
6. Write the map to the screen with a SEND MAP command, adding any device control information required.

What to do next

Acquiring and defining storage for the maps

The first step in creating mapped output is to provide storage in which to arrange the variable map data that your program passes to BMS.

About this task

If you place the map structure in working storage, CICS does the allocation for you. (CICS allocates a private copy of working storage for each execution of a program, so that data from one task does not get confused with that from another, as explained in “Program storage” on page 90 .) To use working storage, copy the symbolic map set there with the language statement provided for the purpose:

```
 COPY in COBOL and assembler
%INCLUDE in PL/I
#include in C and C++
```

Working storage is the WORKING-STORAGE SECTION in COBOL, automatic storage in PL/I, C, C++, and DFHEISTG in a CICS assembler program. For example:

```
 WORKING-STORAGE SECTION.
...
01 COPY QCKSET.
...
```

Alternatively, you can obtain and release map set storage as you need it, using CICS GETMAIN commands. (GETMAIN is discussed in “Storage control” on page 292 .) In this case you copy the map into storage addressed by a pointer variable (the LINKAGE SECTION in COBOL, based storage in PL/I, C, and C++, a DSECT in assembler). On return from the GETMAIN, you use the address returned in the SET option to associate the storage with the data structure, according to the facilities of the programming language.

We used working storage in the example in Figure 109 on page 370 , but we could have used a GETMAIN. If we had, the code would change to:

```
 LINKAGE SECTION.
...
01 COPY QCKSET.
...
PROCEDURE DIVISION.
...
MOVE LENGTH OF QCKMAPO TO LL.
EXEC CICS GETMAIN SET(ADDRESS OF QCKMAPO)
LENGTH(LL) END-EXEC.
...
```

The length you need on your GETMAIN command is the length of the variable whose name is the map name suffixed by the letter “O”. In COBOL, PL/I, C, and C++, you can use language facilities to determine this length, as in the previous example. In assembler, it is defined in an EQUate statement whose label is the map name suffixed by “L”.

BASE and STORAGE options
Two options on the DFHMSD map set definition macro affect how storage for maps is defined: BASE and STORAGE=AUTO (the STORAGE option always has the value AUTO). You can use either one or neither, so there are *three* possibilities.

If you specify neither for a map set containing several maps, the symbolic structures for the maps are defined so that they overlay one another. If you specify STORAGE=AUTO, they do not; each occupies separate space. Thus STORAGE=AUTO requires more storage.

However, when you use maps that overlay one another in a single program, you must use them serially or compensate for the reuse of storage by programming. Unless storage is a major issue, STORAGE=AUTO simplifies programming and reduces the risk of error.

In PL/I, C, and C++, STORAGE=AUTO has the additional effect of defining the map as automatic storage (storage that CICS allocates); the absence of STORAGE=AUTO causes these compilers to assume based storage, for which you generally incur the overhead of an additional GETMAIN. BMS assigns the name BMSMAPBR to the associated pointer variable, unless you specify another name with the BASE option.

The third possibility, BASE, lets you use the same storage for all the maps in multiple map sets. Its effect varies slightly with the programming language, but essentially, all the maps in map sets with the same BASE value overlay one another. In COBOL, BASE=xxxx causes the 01 levels (that is, each individual map) to contain a REDEFINES xxxx clause. In PL/I, C, and C++, it designates each map as storage based on the pointer variable xxxx. BASE cannot be used when the programming language is assembler.

Initializing the output map

Before you start building your output, make sure that the map storage is initialized to nulls, so that data that remains there by a previous process is not used inadvertently.

About this task

If you have read input data using this same map, or one that overlays it, you need to ensure that you have processed or saved this data first. The relationship between input and output maps is discussed in “The symbolic input map” on page 398 , and using the same map you used for input is discussed in “Sending mapped output after mapped input” on page 404.

You initialize by moving nulls (X'00') into the structure. The symbolic map structures are defined so that you can refer to the whole map area named the map suffixed by the letter *O* . You can see this in Figure 108 on page 370 , and, in fact, the statement:

```
 MOVE LOW-VALUES TO QCKMAPO.
```

would clear the area in which we built the map in the “quick check” example. If you are using the map for both input and output, it might be easier to clear the map one field at a time, as you edit the input (see “Handling input errors” on page 403).

When you obtain map storage with a CICS GETMAIN instruction, another way to initialize is to use the INITIMG option.

Moving the variable data to the map

Having obtained storage for your map, established the relationship of the map structure to the storage, and initialized, you are finally ready to create your output. There are two parts to it: the data itself and its display attributes. Firstly, we explain to you about the data, and then we explain to about the attributes.

In the typical case, an output display consists of some constant or default data (provided by the physical map) and some variable data (provided by the program). For each field that you want to supply by program, you move the data into the field in the symbolic map whose name is the name assigned in the map suffixed by the letter *O*. See the code in A BMS output example for an example.

If you do not supply a value for a field (that is, you leave it null, as initialized), BMS ordinarily uses the initial value assigned in the map, if any. Constants (that is, fields without names) also get the initial values specified in the map. However, the DATAONLY and MAPONLY options on the SEND MAP command modify the way in which program and map data are merged; we explain these options in “Merging the symbolic and physical maps” on page 390 and summarize the exact rules in “Building the output screen” on page 391.

Setting the display characteristics

Display attributes are the second component of the output data. For named fields, BMS generates subfields which can be used to override the map assigned value in your program.

In A BMS output example , we show how 3270 field attributes for a map field are defined with the ATTRB option, and how BMS generates the “A” subfield to let you override the map value by program if you name the field. (See “3270 field attributes” on page 277 for information about attributes).

BMS always provides the *A* subfield, because all 3270 devices support field attributes. Many 3270s also have some of the extended attributes shown in Table 47 on page 387 . BMS supports each of these attributes individually in much the same way that it does field attributes collectively. You can assign attribute values in your DFHMDF field definitions, and, if you name the field, BMS generates a subfield in the symbolic map, so that you can override the map-assigned value in your program. There is a separate subfield for each type of extended attribute.

You can request subfields for the extended attributes by specifying the required attribute in the DSATTS option of DFHMDI or DFHMSD. You must also include the list of extended attributes in the MAPATTS option (even if these attribute types do not appear in any DFHMDF macro).

Table 47. BMS attribute types. The columns show the types of attributes, the name of the associated MAPATTS and DSATTS value, and the suffix of the associated subfields in the symbolic map.

Attribute type	**MAPATTS, DSATTS value**	**Subfield suffix**
Field attributes	None (default)	A
Color	COLOR	C
Highlighting	HILIGHT	H
Outlining	OUTLINE	U
Background transparency	TRANSP	T
Validation	VALIDN	V
Double-byte character capability	SOSI	M
Programmed symbols	PS	P

Note: If you use programmed symbols, you need to ensure that a suitable symbol set has been sent to the device first, unless you choose one that is permanently loaded in the device. You can use a terminal control SEND command to do this (see “Using data transmission commands” on page 260 and IBM 3270 Data Stream Programmers Reference.

The types of attributes that apply depend on the features of your principal facility at the time of execution. If you specify a value for an attribute that the terminal does not possess, BMS ignores it. If you are supporting different terminal types, however, you may need to use different techniques to get the same visual clarity. You can find out what kind of terminal you are using with the ASSIGN and INQUIRE commands, explained in “Finding out about your terminal” on page 265 . There are also provisions in BMS for keeping your program independent of the terminal type; see “Device-dependent maps” on page 382.

Changing the attributes
Field attributes in map definitions can be changed using the options MAPATTS and DSATTS.

Here is an example of how this works. Suppose that the terminals in our “quick check” application have color and highlighting capabilities. We might decide to show the maximum charge allowed in a different color from the rest of the screen, because this field is of most interest to the clerk. We might also make the warning message red, because when it appears at all, it is important for the clerk to notice it. And when we really want to get the clerk's attention, because the card is stolen, we could change the attributes in the program to make the message flash. To add these features, we need to change our map definition as follows:

```
QCKMAP DFHMDI SIZE=(24,80),..., X
MAPATTS=(COLOR,HILIGHT),COLOR=GREEN,HILIGHT=OFF,DSATTS=HILIGHT
```

The MAPATTS option tells BMS that we specify color and highlighting in the map (or in the program, because any attribute listed in DSATTS must be included in MAPATTS as well). The COLOR and HILIGHT values indicate that fields with no color assigned should be green and that highlighting should be off if not specified.

The only field definitions that we need to change are the ones that *are not* green or *are* highlighted:

```
CHG DFHMDF
POS=(5,13),LENGTH=8,ATTRB=(ASKIP,NORM),PICOUT='$,$$0.00', X
COLOR=NEUTRAL
MSG DFHMDF LENGTH=20,POS=(7,1),ATTRB=(ASKIP,NORM),COLOR=RED
```

Specifying COLOR=NEUTRAL means that on a terminal, the field is displayed in white.

The DSATTS option tells BMS that we want to alter the highlighting of some fields at execution time, and therefore it should produce “H”-suffix subfields in the symbolic map to let us do that. Each named field gets the extra subfield; the message field, for example, expands from the current three lines in Figure 108 on page 370 to:

```
 02 FILLER PICTURE X(2).
02 MSGH PICTURE X.
02 MSGA PICTURE X.
02 MSGO PIC X(30).
```

The program statement we need to produce the flashing is:

```
 MOVE DFHBLINK to MSGH.
```

In general, BMS takes attribute values from the program if you supply them and from the map if you do not (that is, if you leave the program value null, as initialized). However, the MAPONLY and DATAONLY options on the SEND MAP command affect attribute values as well as field data, as explained in Where the values come from in “Building the output screen” on page 391.

Attribute value definitions: DFHBMSCA
The 1 byte values required to set attribute values are bit combinations defined by 3270 hardware. CICS provides source code, named DFHBMSCA, which defines the commonly used values for all attributes, and assigns meaningful names to each combination.

You can copy DFHBMSCA into your program. The program DFHBLINK in "Changing the attributes" on page 388 is an example. To define DFHBLINK, copy DFHBMSCA into the working storage, as follows:

```
 WORKING-STORAGE SECTION.
...
COPY DFHBMSCA.
```

A version of DFHBMSCA is provided for each programming language. The value names are the same in all versions. If you need an attribute combination that is not included in DFHBMSCA, to determine the value, see IBM 3270 Data Stream Programmers Reference. If you use the value often, you can modify DFHBMSCA to include it.

In assembler language only, the values are defined with EQUates, so you use MVI rather than MVC instructions.

Using the SEND MAP command

This list shows you what the SEND MAP command tells BMS.

The SEND MAP command tells BMS:

- Which map to use (MAP option), and where to find that map (the MAPSET option)
- Where to find the variable data for the map (FROM option) and how to merge it with the values from the map (MAPONLY and DATAONLY)
- Which device controls to include in the data stream, and other control options
- Where to put the cursor, if you want to override the position in the map definition (the CURSOR option)
- Whether the message is complete or is built cumulatively (the ACCUM option)
- What to do with the formatted output (TERMINAL, SET and PAGING options)

The MAP and MAPSET options are self-explanatory, and we cover most of the rest as we describe the programming steps that precede a simple SEND MAP. The last two topics require a knowledge of BMS logical message facilities, which we describe in "Output disposition options: TERMINAL, SET, and PAGING" on page 394.

Until we get to that point, we assume the defaults: that each SEND MAP creates one message, and we are sending that message to our own terminal.

SEND MAP control options

There are many control options for the BMS SEND commands. Some apply only to particular devices or special features of BMS, and we defer describing these until we get to the associated device support or feature. The following device control options, however, apply generally.

- **ERASE, ERASEAUP** , and **FRSET** all modify the contents of the device buffer, if the terminal has one, before writing your output into it. ERASE sets the entire buffer to nulls (X'00'). If the terminal has the alternate screen size feature, ERASE also sets the buffer size. Therefore, the first SEND MAP in a task normally specifies the ERASE option, both to clear the buffer and to select the buffer size. See 3270 write commands in "The output datastream" on page 275 for more information about alternate screen size.

  ERASEAUP (erase all unprotected fields) sets the contents of all fields in the buffer that are unprotected (that is, fields which the operator can change) to nulls. This is useful for data entry, as we explain in DATAONLY option in "Merging the symbolic and physical maps" on page 390.

  FRSET (field reset) turns off the modified data tag of all fields in the buffer; "3270 field attributes" on page 277 explains more about this option.

- **FREEKB** (free keyboard) unlocks the keyboard when the output is sent to the terminal. You typically want to do this on a display terminal.
- **ALARM** sounds the audible alarm, if the terminal has one.
- **FORMFEED, PRINT, L40, L64, L80,** and **HONEOM** are specific to printing and are explained in "CICS 3270 printer options" on page 348. **NLEOM** also is used mainly in printing, and is explained in the same section. NLEOM requires **standard** BMS.

Some of these options can also be specified in the map itself, in particular, the options that are expressed in the 3270 write control character and coded in the CTRL option of the DFHMDI or DFHMSD macros: PRINT, FREEKB, ALARM, FRSET, L40, L64, L80, HONEOM.

Note: CTRL options are always treated as a group, so if you include any of them on your SEND MAP command, BMS ignores the values for *all* of them in your map definition and uses only those in the command. As noted earlier, you can also send device control options separate from your map data, using a SEND CONTROL command. You can use any option on SEND CONTROL that you can use on SEND MAP, except those that relate expressly to data, such as NLEOM.

Other BMS SEND options: WAIT and LAST

When a task writes to a terminal with a BMS or terminal control SEND command CICS normally schedules the transmission and then makes the task ready for execution again. Actual transmission occurs later, depending on terminal type, access method and other activity in the system. If you want to ensure that transmission is complete before your program regains control, use the WAIT option.

WAIT can increase response time slightly, because it prevents overlap between processing and output transmission for a task. (Overlap occurs only until a subsequent SEND, RECEIVE, or end of task, however, because CICS finishes one terminal operation completely before starting another.)

You can improve response time slightly for some terminals by using the LAST option. LAST indicates that the output you are sending is the last output for the task. This knowledge allows CICS to combine transmission of the data with the z/OS Communications Server end-of-bracket flow that occurs at end of task.

Merging the symbolic and physical maps

So far, we have assumed that every display consists of some constant data (provided by the physical map) and some variable data (provided by the program and structured according to the symbolic map). Sometimes, however, one or more of these components is missing.

MAPONLY option

For example, a menu map may not need any data supplied by program. In such a case, you code the MAPONLY option in place of the FROM option on your SEND MAP command. BMS then takes all the information from the physical map, sending the initial values for both the constant (unnamed) and named fields. You do not need to copy the symbolic map set into a program that always sends the map with MAPONLY, and, in fact, you can skip the TYPE=DSECT map set assembly if all programs use all the maps in the set in this way.

MAPONLY is also the way you get an input-only map to the screen.

DATAONLY option

The opposite situation is also possible: the program can supply all the data and not need any constant or default values from the map. This happens on the second and subsequent displays of a map in many situations: data entry applications, inquiry applications where the operator browses through a series of records displayed in identical format, and screens which are redisplayed after detection of an error in the input.

BMS takes advantage of this situation if you indicate it with the DATAONLY option. You still need to tell BMS which map and map set you are using for positioning information, but BMS sends only those fields

which have non-null attribute or data values in the symbolic map. Other fields and attribute values remain unchanged.

The SEND CONTROL command

There are also occasions when you do not need to send data at all, but you do need to send device controls. For example, you might need to erase the screen or sound the alarm. You do this with a SEND CONTROL command listing the options you need,

Consider a program in a data entry application. When first initiated, it displays the data entry map to format the screen with the input fields, the associated labels, screen headings and instructions. This first SEND MAP command specifies MAPONLY, because the program sends no variable data. Thereafter, the program accepts one set of data input. If the input is correct, the program files it and requests another. It still does not need to send any variable data. What it needs to do is to erase the input from the screen and unlock the keyboard, to signal the operator to enter the next record.

```
EXEC CICS SEND CONTROL ERASEAUP FREEKB END-EXEC
```

does this. (See “SEND MAP control options ” on page 389 for a description of these and other device control options.)

If there are errors, the program does need to send variable data, to tell the operator how to fix the problem. This one changes the attributes of the fields in error to highlight them and sends a message in a field provided for the purpose. Here, our program uses the DATAONLY option, because the map is already on the screen. (We tell you more about handling input errors in “Handling input errors” on page 403 .)

You should use MAPONLY, DATAONLY, and SEND CONTROL when they apply, especially when response time is critical, as it is in a data entry situation. MAPONLY saves path length, DATAONLY reduces the length of the outbound data stream, and SEND CONTROL does both.

Building the output screen

The interaction of physical map definition options, SEND MAP options, program data and merge options is sufficiently complex that a summary of the rules for determining what appears on the screen after a SEND MAP is in order.

The contents of the screen (buffer) are determined by:

- What was there before your SEND MAP command
- The fields (field attributes, extended attributes, and display data) that get sent from your SEND MAP command
- Where the several values for these field elements come from

We discuss the possibilities in that order.

What you start with

The first thing that happens on a SEND MAP command is that the entire screen (buffer) is cleared to nulls if the ERASE option is present, regardless of the size or origin of your map. On terminals that have the alternate screen size feature, the screen size is set as well, as explained in 3270 write commands in “The output datastream” on page 275 . The screen is in unformatted state, with no fields defined and no display data. If ERASEAUP is present, all of the unprotected fields on the screen are erased, but the field structure and attributes of all fields and the contents of protected fields are unchanged.

ERASE and ERASEAUP are honored *before* your SEND MAP data is loaded into the buffer. If neither of these options appears on the SEND MAP, the screen buffer starts out in the same state it was in after the previous write operation, modified by whatever the operator did. In general, the positions of the fields (that is, of the attributes bytes) and their attributes are unchanged, but the data content of unprotected fields can be different. Furthermore, if the operator used the CLEAR key, the whole buffer is cleared to nulls and the screen is in unformatted state, just as if you had included the ERASE option.

What is sent

Secondly, BMS changes only those positions in the buffer within the confines of your map. Outside that area, the contents of the buffer are unchanged, although it is possible for areas outside your map to change in appearance, as explained in Outside the map.

Within the map area, what is sent depends on whether the DATAONLY option is present. In the typical case, where it is not present, BMS sends every component (field attributes, extended attributes, display data) of every field in your map. This creates a field at the position specified in the POS operand and overlays the number of bytes specified in the LENGTH field from POS. Buffer positions between the end of the display data and the next attributes byte (POS value) remain unchanged. (There might or might not be fields (attributes bytes) in these intervening spaces if you did not ERASE after the last write operation that used a different map.)

The values for these field elements come from the program, the map or defaults, as explained in the next section.

If DATAONLY is present, BMS sends only those fields, and only those components for them, that the program provides. Other screen data is unchanged.

Where the values come from

The values that determine screen contents can come from four sources: the program; the map; the hardware's defaults; or the previous contents of the screen.

BMS considers each component of each map field separately, and takes the value from the program, provided:

- The MAPONLY option has not been used.
- The field has a name in the map, so that the symbolic output map contains the corresponding set of subfields from which to get the data. The field attributes value comes from the program subfield whose name is the map field name suffixed by A. The display data comes from the subfield of the same name suffixed by O, and the extended attribute values come from the same-named subfields suffixed by the letter that identifies the attribute (see Table 47 on page 387). In the case of the extended attributes, the attribute must also appear among DSATTS in order for the symbolic map to contain the corresponding subfield.
- A value is present. The definition of “present” varies slightly with the field component:
  - For field attributes bytes, the value must not be null (X'00') or one of the values that remain from an input operation (X'80' , X'02' , or X'82').
  - For extended attribute bytes, the value must not be null.

    Note: BMS sends only those extended attribute values that the terminal is defined as supporting. Values for other extended attributes are omitted from the final data stream.
  - For display data, the first character of the data must not be null.

If any of these conditions is not met, the next step depends on whether DATAONLY is present. With DATAONLY, BMS stops the process here and sends only the data it got from the program. BMS does this in such a way that components not changed by program are not changed on the screen. In particular, extended attributes values are not changed unless you specify a new value or ask for the hardware default. (A value of X'FF' requests the hardware default for all extended attributes except background transparency, for which you specify X'F0' to get the hardware default.)

Without DATAONLY, if one of the previous conditions is not met, BMS takes the data from the map, as follows:

- For field attributes, it takes the value in the ATTRB option for the field. If none is present, BMS assumes an ATTRB value of (ASKIP,NORM).
- For extended attributes, BMS takes the value from:
  - The corresponding option in the DFHMDF field definition

  - If it is not specified there, the value is taken from the corresponding option in the DFHMDI map definition
  - If it is not there either, the value is taken from the corresponding option in the DFHMSD map set definition

  (If no value is specified anywhere, BMS does not send one, and this causes the 3270 to use its hardware default value.)
- For display data, from the initial value in the map (the INITIAL, XINIT, or GINIT option). If there is no initial value, the field is set to nulls.

Outside the map

Your map need not be the same size as your screen or printer page. Your application can use only a part of the screen area, or you might want to build your output incrementally, or both.

BMS logical messages allow you to build a screen from several maps, sending it with a single terminal write. You use the ACCUM option to do this, which we cover in "BMS logical messages" on page 406 . Even without using ACCUM, you can build a screen from several maps if the terminal is a 3270-like device with a buffer. You do this with multiple SEND MAP commands written to different areas of the screen (buffer), not erasing after the first command. Each SEND MAP causes output and might produce a momentary "flash" at a display device. For this reason, and to eliminate the path length of extra I/O, you might prefer to use logical messages for such composite screens.

Outside the map sent, the contents of the buffer are unchanged, except for the effects of ERASE and ERASEAUP cited earlier. In general, this means that the corresponding areas of the screen are unchanged. However, a screen position outside the map can get its attributes from a field within the map. This occurs if you do not define a field (using a different map) beyond the boundary of the map and before the position in question. If you change the attributes of the field inside your map governing this position outside, the appearance of the position might change, even though the contents do not.

Using GDDM and BMS

One use of the buffer overlay technique is the creation of screens containing a mixture of BMS and Graphical Data Display Manager (GDDM) output.

You generally write the BMS output first, followed by the GDDM. You can leave space in the BMS map for the GDDM output, or you can create a "graphic hole" in any display by writing a map with no fields in it to the position where you want the hole. Such a map is called a "null map," and its size (height and width) correspond to the size of the hole.

If you use GDDM to combine graphics with BMS output, you need to include a GDDM PSRSRV call to prevent GDDM from corrupting programmed symbol sets that BMS might be using.

Positioning the cursor

Positioning the cursor is important when you use a map for input. Usually, you set the initial position for the cursor in the map definition by including "insert cursor" (IC) in the ATTRB values of the field where you want it.

The CURSOR option on the SEND MAP command allows you to override this specification, if necessary, when the map is displayed. If you specify CURSOR(value), BMS places the cursor in that absolute position on the screen. Counting starts in the first row and column (the zero position), and proceeds across the rows. Thus, to place the cursor in the fourth column of the third row of an 80-column display, you code CURSOR(163).

Specifying CURSOR without a value signals BMS that you want "symbolic cursor positioning". You do this by setting the length subfield of the field where you want the cursor to minus one (-1). Length subfields are not defined on output-only maps, so you must define your map as INOUT to use symbolic cursor positioning. (We tell you about length subfields in "Formatted screen input" on page 399 , and about INOUT maps in "Receiving mapped data" on page 396 .) If you mark more than one field in this way, BMS uses the first one it finds.

Symbolic cursor positioning is particularly useful for input-output maps when the terminal operator enters incorrect data. If you validate the fields, setting the length of any in error to -1, BMS places the cursor under the first error when you redisplay. “Processing the mapped input” on page 402 shows this technique.

You can position the cursor with a SEND CONTROL command also, but only by specifying an absolute value for CURSOR; if you omit CURSOR on SEND CONTROL, the cursor is not moved.

Sending invalid data and other errors

Most of the exception conditions that can occur on SEND MAP and SEND CONTROL commands apply only to the advanced BMS options: logical messages, partitions, and special devices. However, it is also possible to send invalid data to a terminal.

BMS does not check the validity of attribute and data values in the symbolic map, although it does not attempt to send an extended attribute, such as color, to a terminal that is not defined to support that attribute.

For more information about the error conditions that can occur on SEND MAP and SEND CONTROL commands, see CICS API commands.

The effects of invalid data depend on both the specific terminal and the nature of the incorrect data:

- Invalid data might be interpreted as a control sequence, so that the device accepts the data but produces the wrong output.
- The screen might display an error indicator.
- An ATNI abend might occur. The point at which your task is notified of an ATNI depends on whether you specified the WAIT option (see “SEND MAP control options ” on page 389).

Output disposition options: TERMINAL, SET, and PAGING

The output disposition options specify what BMS should do with the formatted output stream. The only disposition option we have described up to this point is TERMINAL, which sends the output to the principal facility of your task. TERMINAL is the default value that you get if you do not specify another disposition. There are, however, two other possibilities; SET and PAGING.

1. BMS can return the formatted output stream to the task rather than sending it to the terminal. You use the SET disposition option to request this. You might do so to defer transmission or to modify the data stream to meet special requirements. “Acquiring and defining storage for the maps” on page 385 explains how and when to use SET.
2. You can ask BMS to store and manage your output in CICS temporary storage for subsequent delivery to your terminal. This option, PAGING, implies that your message may contain more than one screen or page, and is useful when you want to send a message to a display terminal that exceeds its screen capacity. BMS saves the entire message in temporary storage until you indicate that it is complete. Then it provides facilities for the operator to page through the output at the terminal. You can use PAGING for printers as well as displays, although you do not need the operator controls, and sometimes TERMINAL is just as satisfactory.

When you use PAGING, the output still goes to your principal facility, though indirectly, as just described. Full BMS also provides a feature, routing, that lets you send your message to another terminal, or several, in place of or in addition to your own. We tell you about routing in “Message routing” on page 418 , after we cover the prerequisites.

Note: Both PAGING and SET and related options require **full** BMS. TERMINAL is the only disposition available in minimum and standard BMS.

Using SET

When you specify a disposition of SET for a BMS message, BMS formats your output and returns it in the form of a device-dependent data stream. No terminal I/O occurs, although the returned data stream usually is sent to a terminal subsequently.

There are several reasons for asking BMS to format a data stream without sending it. You might want to do any of the following:

- Edit the data stream to meet the requirements of a device with special features or restrictions not explicitly supported by CICS.
- Compress the data stream, based on standard 3270 features or special device characteristics.
- Forward the data stream to a terminal not connected directly to CICS. For example, you might want to pass data to a 3270 terminal attached to a system connected to CICS by an APPC link. You can format the data with SET and send the resulting pages to a partner program across the link. If the terminal is of a different type from your principal facility, you can define a dummy terminal of the appropriate type and then ROUTE to that terminal with SET to get the proper formatting, as explained in "Routing with SET" on page 424.

BMS returns formatted output by setting the pointer variable named in the SET option to the address of a page *list* . This list consists of one or more 4-byte entries in the following format, each corresponding to one page of output.

Table 48. Page list entry format

Bytes	Contents
0	Terminal type (see Table 46 on page 383)
1-3	Address of TIOA containing the formatted page of output

An entry containing -1 (X'FF') in the terminal type signals the end of the page list. Notice that the addresses in this list are only 24 bits long. If your program uses 31-bit addressing, you must expand a 24-bit address to a full word by preceding it with binary zeros before using it as an address.

Each TIOA (terminal input-output area) is in the standard format for these areas:

Table 49. TIOA format

Field name	Position	Length	Contents
TIOASAA	0	8	CICS storage accounting information (8 bytes)
TIOATDL	8	2	Length of field TIOADBA in halfword binary format
(unnamed)	10	2	Reserved field
TIOADBA	12	TIOATDL	Formatted output page
(unnamed)	TIOATDL + 12	4	Page control area, required for the SEND TEXT MAPPED command (if used)

The reason that BMS uses a list to return pages is that some BMS commands produce multiple pages. SEND MAP does not, but SEND TEXT can. Furthermore, if you have established a routing environment, BMS builds a separate logical message for each of the terminal types among your destinations, and you may get pages for several different terminal types from a single BMS command. The terminal type tells you to which message a page belongs. (Pages for a given type are always presented in order.) If you are not routing, the terminal type is always that of your principal facility.

If you are not using the ACCUM option, pages are available on return from the BMS command that creates them. With ACCUM, however, BMS waits until the available space on the page is used. BMS turns on the RETPAGE condition to signal your program that pages are ready. You can detect RETPAGE with a HANDLE CONDITION command or by testing the response from the BMS command (in EIBRESP or the value returned in the RESP option).

You must capture the information in the page list whenever BMS returns one, because BMS reuses the list. You need save only the addresses of the pages, not the contents. BMS does not reuse the pages themselves, and, in fact, moves the storage for them from its control to that of your task. This allows you to free the storage for a page when you are through with it. If you do this, the DATA or DATAPOINTER option in your FREEMAIN command should point to the TIOATDL field, not to TIOASAA.

Receiving mapped data

Formatted screens are as important for input as for output. Data entry applications are an obvious example, but most other applications also use formatted input, at least in part.

On input, BMS does for you approximately the reverse of what it does on output: it removes device control characters from the data stream and moves the input fields into a data structure, so that you can address them by name.

Maps can be used exclusively for input, exclusively for output (the case we have already covered), or for both. Input-only maps are relatively rare, and we cover them as a special case of an input-output map, pointing out differences where they occur.

An input-output example

This example shows you the map definition for a simple data entry program called "quick update".

Before we explain the details of the input structure, let us reexamine the “quick check” example in “A BMS output example” on page 368. Suppose that it is against our policy to let a customer charge up to the limit over and over again between the nightly runs when new charges are posted to the accounts. We want a new transaction that augments “quick check” processing by keeping a running total for the day.

In addition, we want to use the same screen for both input and output, so that there is only one screen entry per customer. In the new transaction, “quick update,” the clerk enters both the account number and the charge at the same time. The normal response is:

```
QUP Quick Account Update
                  Current charge okay; enter next
                  Account: _______
                  Charge: $ _______
```

Figure 114. Normal “quick update” response

When we reject a transaction, we leave the input information about the screen, so that the clerk can see what was entered along with the description of the problem:

```
QUP Quick Account Update
                  Charge exceeds maximum; do not approve
                  Account: 482554
                  Charge: $ 1000.00
```

Figure 115. “Quick update” error response

(Here again, we are oversimplifying to keep our maps short for ease of explanation.)

The map definition we need for this exercise is:

```
QUPSET DFHMSD
TYPE=MAP,STORAGE=AUTO,MODE=INOUT,LANG=COBOL,TERM=3270-2
QUPMAP DFHMDI SIZE=(24,80),LINE=1,COLUMN=1,CTRL=FREEKB
DFHMDF POS=(1,1),LENGTH=3,ATTRB=(ASKIP,BRT),INITIAL='QUP'
DFHMDF POS=(1,26),LENGTH=20,ATTRB=(ASKIP,NORM), X
INITIAL='Quick Account Update'
MSG DFHMDF LENGTH=40,POS=(3,1),ATTRB=(ASKIP,NORM)
DFHMDF POS=(5,1),LENGTH=8,ATTRB=(ASKIP,NORM), X
INITIAL='Account:'
ACCTNO DFHMDF POS=(5,14),LENGTH=6,ATTRB=(UNPROT,NUM,IC)
DFHMDF POS=(5,21),LENGTH=1,ATTRB=(ASKIP),INITIAL=' '
DFHMDF POS=(6,1),LENGTH=7,ATTRB=(ASKIP,NORM),INITIAL='Charge:'
CHG DFHMDF POS=(6,13),ATTRB=(UNPROT,NORM),PICIN='$$$0.00'
DFHMDF POS=(6,21),LENGTH=1,ATTRB=(ASKIP),INITIAL=' '
DFHMSD TYPE=FINAL
```

Figure 116. Map definition for input-output map

You can see that the map field definitions for this input-output map are like those for the output-only “quick check” map, if we allow for changes to the content of the screen. The differences to note are:

- The MODE option in the DFHMSD map set definition is INOUT, indicating that the maps in this map set are used for both input and output. INOUT causes BMS to generate a symbolic structure for input as well as for output for every map in the map set. If this had been an input-only map, we would have said MODE=IN, and BMS would have generated only the input structures.
- We put names on the fields from which we want input (ACCTNO and CHG) as well as those to which we send output (MSG). As in an output-only map, we avoid naming constant fields to save space in the symbolic map.
- The input fields, ACCTNO and CHG, are unprotected (UNPROT), to allow the operator to key data into them.
- IC (insert cursor) is specified for ACCTNO. It positions the cursor at the start of the account number field when the map is first displayed, ready for the first item that the operator has to enter. (You can override this placement when you send the map; IC just provides the default position.)
- Just after the ACCTNO field, there is a constant field consisting of a single blank, and a similar one after the CHG field. These are called “stopper” fields. Normally, they are placed after each input field that is not followed immediately by some other field. They prevent the operator from keying data beyond the space you provided, into an unused area of the screen.

  If you define the stopper field as “autoskip”, the cursor jumps to the next unprotected field after the operator has filled the preceding input field. This is convenient if most of the input fields are of fixed length, because the operator does not have to advance the cursor to get from field to field.

  If you define the stopper field as “protected,” but not “autoskip,” the keyboard locks if the operator attempts to key beyond the end of the field. This choice may be preferable if most of the input fields are of variable length, where one usually has to use the cursor advance key anyway, because it alerts the operator to the overflow immediately. Whichever you choose, you should use the same choice throughout the application if possible, so that the operator sees a consistent interface.
- The CHG field has the option PICIN. PICIN produces an edit mask in the symbolic map, useful for COBOL and PL/I, and implies the field length. See BMS macro DFHMDF for details on using PICIN.

Figure 117 on page 397 shows the symbolic map set that results from this INOUT map definition.

```
01 QUPMAPI.
02 FILLER PIC X(12).
02 FILLER PICTURE X(2).
02 MSGL COMP PIC S9(4).
02 MSGF PICTURE X.
02 FILLER REDEFINES MSGF.
03 MSGA PICTURE X.
02 MSGI PIC X(40).
02 ACCTNOL COMP PIC S9(4).
02 ACCTNOF PICTURE X.
02 FILLER REDEFINES ACCTNOF.
03 ACCTNOA PICTURE X.
02 ACCTNOI PIC X(6).
02 CHGL COMP PIC S9(4).
02 CHGF PICTURE X.
02 FILLER REDEFINES CHGF.
03 CHGA PICTURE X.
02 CHGI PIC X(7) PICIN '$,$$0.00'.
01 QUPMAPO REDEFINES QUPMAPI.
02 FILLER PIC X(12).
02 FILLER PICTURE X(3).
02 MSGO PIC X(40).
02 FILLER PICTURE X(3).
02 ACCTNO PICTURE X(6).
02 FILLER PICTURE X(3).
02 CHGO PIC X.
```

Figure 117. Symbolic map for “quick update”

The second part of this structure, starting at QUPMAPO, is the symbolic *output* map—the structure required to send data back to the screen. Apart from the fields we redefined, it looks almost the same as the one you would have expected if we had specified MODE=OUT instead of MODE=INOUT. See Figure 106 on page 369 for a comparison. The main difference is that the field attributes (A) subfield appears to be missing, but we explain this in a moment.

The symbolic input map
The first part of the structure, under the label QUPMAPI, is new. This is the symbolic *input* map, the structure required for reading data from a screen formatted with map QUPMAP.

For each named field in the map, it contains three subfields. As in the symbolic output map, each subfield has the same name as the map field, suffixed by a letter indicating its purpose. The suffixes and subfields related to input are:

L
the length of the input in the map field.

F
the flag byte, which indicates whether the operator erased the field and whether the cursor was located there.

I
the input data itself.

The input and output structures are defined so that they overlay one another field by field. That is, the input (I) subfield for a given map field always occupies the same storage as the corresponding output (O) subfield. Similarly, the input flag (F) subfield overlays the output attributes (A) subfield. (For implementation reasons, the order of the subfield definitions varies somewhat among languages. In COBOL, the definition of the A subfield moves to the input structure in an INOUT map, but it still applies to output, just as it does in an output-only map. In assembler, the input and output subfield definitions are interleaved for each map field.)

BMS uses dummy fields to leave space in one part of the structure for subfields that do not occur in the other part. For example, there is always a 2-byte filler in the output map to correspond to the length (L) subfield in the input map, even in output-only maps. If there are output subfields for extended attributes, such as color or highlighting, BMS generates dummy fields in the input map to match them. You can see examples of these fields (FILLERs in COBOL) in both Figure 106 on page 369 and Figure 117 on page 397.

The correspondence of fields in the input and output map structures is very convenient for processes in which you use a map for input and then write back in the same format, as you do in data entry transactions or when you get erroneous input and have to request a correction from the operator.

Programming mapped input
The programming required for mapped input is similar to that for mapped output, except that the data is going in the opposite direction. You define your maps and assemble them first, as for mapped output.

In the program or programs reading from the terminal, you:

1. Acquire the storage to which the symbolic map set corresponds.
2. Copy the symbolic map set to define the structure of this storage.
3. Format the input data with a RECEIVE MAP command.
4. Process the input.

If the transaction also calls for mapped output, as “quick update” and most other transactions do, you continue with the steps outlined before, in “Sending BMS mapped output” on page 385 . Some considerations and shortcuts for mapped input are described in “Sending mapped output after mapped input” on page 404.

Using the RECEIVE MAP command
The RECEIVE MAP command causes BMS to format terminal input data and make it accessible to your application program.

The command specifies:

- Which map to use in formatting the input data stream—that is, what format is on the screen and what data structure the program expects (the MAP option)
- Where to find this map (MAPSET option)
- Where to get the input (TERMINAL or FROM option)

- Whether to suppress translation to uppercase (ASIS option)
- Where to put the formatted input data (the INTO and SET options)

The MAP and MAPSET options together tell BMS which map to use, and they work exactly as they do on a SEND MAP command.

BMS gets the input data to format from the terminal associated with your task (its principal facility), unless you use the FROM option. FROM is an alternative to TERMINAL, the default, used in relatively unusual circumstances (see “Formatting other input” on page 405).

BMS also translates lowercase input to uppercase automatically in some cases; we explain how to control translation in “Uppercase translation” on page 400.

You tell BMS where to put the formatted input with the INTO or SET option of RECEIVE MAP.

In addition to the data on the screen, the RECEIVE MAP command tells you where the operator last used the cursor and what key caused transmission. This information becomes available in the EIB on completion of the RECEIVE MAP command. EIBAID identifies the transmit key (the “attention identifier” or AID), and EIBCURSR tells you where the cursor was last located.

Getting storage for mapped input

When you issue a **RECEIVE MAP** command, BMS needs storage in which to build the input map structure. You can provide this space yourself, either in the working storage of your program or with a **GETMAIN** command.

These are the same choices you have for allocating storage in which to build an output map, and you use them the same way (see “Acquiring and defining storage for the maps” on page 385 for details and examples). For either, you code the INTO option on your RECEIVE command, naming the variable into which the formatted input is to be placed. For our “quick update”, for example, the required command is:

```
 EXEC CICS RECEIVE MAP('QUPMAP') MAPSET('QUPSET')
INTO(QUPMAPI) END-EXEC.
```

Usually, the receiving variable is the area defined by the symbolic input map, to which BMS assigns the map name suffixed by the letter “I”, as previously shown. You can specify some other variable if you want, however.

For input operations, you have a third choice for acquiring storage. If you code the SET option, BMS acquires the storage for you at the time of the RECEIVE command and returns the address in the pointer variable named in the SET option. So we could have coded the RECEIVE MAP command in “quick update” like this:

```
 LINKAGE SECTION.
...
01 QUPMAP COPY QUPMAP.
...
PROCEDURE DIVISION.
...
EXEC CICS RECEIVE MAP('QUPMAP') MAPSET('QUPSET')
SET(ADDRESS OF QUPMAPI) END-EXEC.
...
```

Storage obtained in this way remains until task end unless you issue a FREEMAIN to release it (see “Storage control” on page 292).

Formatted screen input

CICS normally reads a 3270 screen with a *read modified* command. CICS provides an option, BUFFER, for the terminal control RECEIVE command, with which you can capture the entire contents of a 3270 screen.

The data transmitted depends on what the operator did to cause transmission:

- The ENTER key or a function key
- CLEAR, CNCL or a PA key (the “short read” keys)
- Field selection: cursor select, light pen detect or a trigger field

You can tell which event occurred, if you need to know; how to do this is explained in “Using the attention identifier” on page 401. You can also find more detail on 3270 input operations in “Input from a 3270 terminal” on page 284.

The short read keys transmit only the attention identifier (the identity of the key itself). No field data comes in, and there is nothing to map. For this reason, short read keys can cause the MAPFAIL condition, as explained in “MAPFAIL and other exception conditions” on page 405 . Field selection features transmit field data, but in most cases not the same data as the ENTER and function keys, which we describe in the paragraphs that follow. See “Support for special hardware” on page 433 for the exceptions if you plan to use these features.

Most applications are designed for transmission by the ENTER key or a function key. When one of these is used to transmit, all of the fields on the screen that have been modified, and *only* those fields, are transmitted.

Modified data
A 3270 screen field is considered modified only if the “modified data tag” (MDT), one of the bits in the field attributes byte, is on.

Modification is explained in “3270 field attributes” on page 277. The terminal hardware turns on this bit if the operator changes the field in any way—entering data, changing data already there, or erasing. You can also turn it on by program when you send the map, by including MDT among the ATTRB values for the field. You do this when you want the data in a particular field to be returned even if the operator does not change it.

You can tell whether there was input from a particular map field by looking at the corresponding length (L) subfield. If the length is zero, no data was read from that field. The associated input (I) subfield contains all nulls (X'00'), because BMS sets the entire input structure to nulls before it performs the input mapping operation. The length is zero either if the modified data tag is off (that is, the field was sent with the tag off and the operator did not change it) or if the operator erased the field. You can distinguish between these two situations, if you care, by inspecting the flag (F) subfield. It has the high-order bit on if the field contains nulls but the MDT is on (that is, the operator changed the field by erasing it). See “Finding the cursor” on page 402 for more information about the flag subfield.

If the length is nonzero, data was read from the field. Either the operator entered some, or changed what was there, or the field was sent with the MDT on. You can find the data itself in the corresponding input (I) subfield. The length subfield tells how many characters were sent. A 3270 terminal sends only non-null characters, so BMS knows how much data was keyed into the field. Character fields are filled out with blanks on the right and numeric fields are filled on the left with zeros unless you specify otherwise in the JUSTIFY option of the field definition. BMS assumes that a field contains character data unless you indicate that it is numeric with ATTRB=NUM.

Related information
BMS macro DFHMDF

Uppercase translation
CICS converts lower case input characters to uppercase automatically under some circumstances. The definition of the terminal and the transaction together determine whether translation occurs.

See the UCTRAN option of the PROFILE and the TYPETERM resource definitions for how these specifications interact.

You can suppress this translation by using the ASIS option on your RECEIVE MAP command, *except* on the first RECEIVE in a task initiated by terminal input. (The first RECEIVE may be either a RECEIVE MAP (without FROM) or a terminal control RECEIVE.) CICS has already read and translated this input, and it is too late to suppress translation. (Its arrival caused the task to be invoked, as explained in “How tasks are started” on page 73 .) Consequently, ASIS is ignored entirely in pseudoconversational transaction sequences, where at most one RECEIVE MAP (without FROM) occurs per task, by definition. For the same reason, you cannot use ASIS with the FROM option (see “Formatting other input” on page 405).

Using the attention identifier

The attention identifier is part of the input in many applications, and you may also need it to interpret the input correctly.

For example, in the "quick update" transaction, we need some method for allowing the clerk to exit our transaction, and we have not yet provided for this. Suppose that we establish the convention that pressing PF12 causes you to leave control of the transaction. We would then code the following after our RECEIVE MAP command:

```
 IF EIBAID = DFHPF12,
EXEC CICS SEND CONTROL FREEKB ERASE END-EXEC
EXEC CICS RETURN END-EXEC.
```

This would end the transaction without specifying which one should be executed next, so that the operator would regain control. The SEND CONTROL command that precedes the RETURN unlocks the keyboard and clears the screen, so that the operator is ready to enter the next request.

The hexadecimal values that correspond to the various attention keys are defined in a copy book called DFHAID. To use these definitions, you copy DFHAID into your working storage, in the same way that you copy DFHBMSCA to use the predefined attributes byte combinations (see “Attribute value definitions: DFHBMSCA” on page 389). The contents of the DFHAID copy book are in Attention identifier constants, DFHAID.

Using the HANDLE AID command

You can use a HANDLE AID command to identify the attention key used.

HANDLE AID is used to specify the label to which control is to be passed when an attention identifier (AID) is received from a display device.

Restriction: This command is supported only in COBOL, PL/I, and assembler language applications (but not AMODE(64) assembler language applications). It is not supported in all other supported high level languages.

HANDLE AID works like other HANDLE commands. You issue it before the first RECEIVE command to which it applies, and it causes a program branch on completion of subsequent RECEIVEs if a key named in the HANDLE AID is used.

The following example shows an alternative to the "escape" code:

```
 EXEC CICS HANDLE AID PF12(ESCAPE) END-EXEC.
...
EXEC CICS RECEIVE MAP('QUPMAP') MAPSET('QUPSET') ...
...
ESCAPE.
EXEC CICS SEND CONTROL FREEKB ERASE END-EXEC
EXEC CICS RETURN END-EXEC.
```

HANDLE AID applies only to RECEIVE commands in the same program. The specification for a key remains in effect until another HANDLE AID in the same program supersedes it by naming a new label for the key, or terminates it by naming the key with no label. A RESP, RESP2, or NOHANDLE option on a RECEIVE command exempts that command from the effects of HANDLE AID specifications, but they remain in effect otherwise.

If you have a HANDLE active for an AID received during an input operation, control goes to the label specified in the HANDLE AID, regardless of any exception condition that occurs and whether a HANDLE CONDITION is active for that exception. Therefore, HANDLE AID can mask an exception condition if you check for it with HANDLE CONDITION. It can be preferable to use an alternative test for the AID, the exceptional conditions, or both. You can check EIBAID for the AID, and use the RESP option or check EIBRESP for exceptions. This situation is especially significant for the MAPFAIL condition; see “MAPFAIL and other exception conditions” on page 405.

Finding the cursor

In some applications, you need to know where the operator last located the cursor at the time of sending. There are two ways of finding out.

If your map specifies CURSLOC=YES, BMS turns on the seventh (X'02') bit in the flag subfield of the map field where the cursor was last located. This only works, of course, if the cursor is located in a map field to which you assigned a name.

Also, because the flag subfield is used to indicate both cursor presence and field erasure, you need to test the bits individually if you are looking for one in particular: the X'80' bit for field erasure and the X'02' bit for the cursor. If you are using a language in which it is awkward to test bits, you can test for combinations. A value of X'80' or X'82' signals erasure; either X'02' or X'82' indicates the cursor. The DFHBMSCA definitions described in the BMS macros manual include all of these combinations.

You can also determine the position of the cursor from the EIBCPOSN field in the EIB. This is the absolute position on the screen, counting from the upper left (position zero) and going across the rows. Thus a value of 41 on a screen 40 characters wide would put the cursor in the second row, second column. Avoid this method if possible, because it makes your program sensitive to the placement of fields on the screen and to the terminal type.

Processing the mapped input

This example shows you how the input subfields are used, using the "quick update" program.

To illustrate how the input subfields are used, we return to “quick update”. After we have the input, we need to do some checks on it before continuing. First, we require that the charge be entered (that is, that the input length be greater than zero), and be positive and numeric.

```
 IF CHGL = 0, MOVE -1 TO CHGL
MOVE 1 TO ERR-NO
ELSE IF CHGI NOT > ZERO OR CHGI NOT NUMERIC,
MOVE DFHUNIMD TO CHGA,
MOVE -1 TO CHGL
MOVE 2 TO ERR-NO.
```

The 'MOVE -1' statements here and following put the cursor in the first field in error when we redisplay the map, as explained in “Positioning the cursor” on page 393 . The message number tells us what message to put in the message area; 1 is “enter a charge”, and so on, through 6, for “charge is over limit”. We do these checks in roughly ascending order of importance, to ensure that the most basic error is the one that gets the message. At the end of the checking, we know that everything is okay if ERR-NO is zero.

An account number must be entered, as well as the charge. If we have one (whatever the condition of the charge), we can retrieve the customer's account record to ensure that the account exists:

```
 IF ACCTNOL = 0, MOVE -1 TO ACCTNOL
MOVE 3 TO ERR-NO
ELSE EXEC CICS READ FILE (ACCT) INTO (ACCTFILE-RECORD)
RIDFLD (ACCTNOI) UPDATE RESP(READRC) END-EXEC
IF READRC = DFHRESP(NOTFOUND), MOVE 4 TO ERR-NO,
MOVE DFHUNIMD TO ACCTNOA
MOVE -1 TO ACCTNOL
ELSE IF READRC NOT = DFHRESP(NORMAL) GO TO HARD-ERR-RTN.
```

If we get this far, we continue checking, until an error prevents us from going on. We need to ensure that the operator gave us a good account number (one that is not in trouble), and that the charge is not too much for the account:

```
 IF ERR-NO NOT > 2
IF ACCTFILE-WARNCODE = 'S', MOVE DFHBMBRY TO MSGA
MOVE 5 TO ERR-NO
MOVE -1 TO ACCTNOL
EXEC CICS LINK PROGRAM('NTFYCOPS') END-EXEC
ELSE IF CHGI > ACCTFILE-CREDIT-LIM - ACCTFILE-UNPAID-BAL
- ACCTFILE-CUR-CHGS
MOVE 6 TO ERR-NO
MOVE -1 TO ACCTNOL.
IF ERR-NO NOT = 0 GO TO REJECT-INPUT.
```

Handling input errors

Whenever you have operator input to process, there is almost always a possibility of incorrect data, and you must provide for this contingency in your code.

Usually, what you must do when the input is wrong is:

- Notify the operator of the errors. Try to diagnose all of them at once; it is annoying to the operator if you present them one at a time.
- Save the data already entered, so that the operator does not have to rekey anything except corrections.
- Arrange to recheck the input after the operator makes corrections.

Flagging errors

In the preceding code for the “quick update” transaction, we used the message field to describe the error (the first one, anyway). We highlighted all the fields in error, provided there was any data in them to highlight, and we set the length subfields to -1 so that BMS would place the cursor in the first bad field. We send this information using the same map, as follows:

```
 REJECT-INPUT.
MOVE LOW-VALUES TO ACCTNOO CHGO.
EXEC CICS SEND MAP('QUPMAP') MAPSET('QUPSET') FROM(QUPMAPO)
DATAONLY END-EXEC.
```

Notice that we specify the DATAONLY option. We can do this because the constant part of the map is still on the screen, and there is no point in rewriting it there. We cleared the output fields ACCTNOO and CHGO, to avoid sending back the input we had received, and we used a different attributes combination to make the ACCTNO field bright (DFHUNIMD instead of DFHBMBRY). DFHUNIMD highlights the field and leaves the modified data tag on, so that if the operator resends without changing the field, the account number is retransmitted.

Saving the good input

When the operator enters correct and incorrect data, the correct data should be saved, so that the operator does not have to rekey anything except corrections. One easy technique is to store the data on the screen. You do not have to do anything additional to accomplish this; once the MDT in a field is turned on, as it is the first time the operator touches the field, it remains on, no matter how many times the screen is read. Tags are not turned off until you erase the screen, turn them off explicitly with the FRSET option on your SEND, or set the attributes subfield to a value in which the tag is off.

The drawback to saving data on the screen is that all the data is lost if the operator uses the CLEAR key. If your task is conversational, you can avoid this hazard by moving the input to a safe area in the program before sending the error information and asking for corrections. In a pseudoconversational sequence, where the component tasks do not span interactions with the terminal, the equivalent is for the task that detects the error to pass the old input forward to the task that processes the corrected input. You can forward data through a COMMAREA on the RETURN command that ends a task, by writing to temporary storage, or in a number of other ways (see “Sharing data across transactions” on page 104 for possibilities).

In addition to avoiding the CLEAR key problem, storing data in your program or in a temporary storage queue reduces inbound transmission time, because you transmit only changed fields on the error correction cycles. (You must specify FRSET when you send the error information to prevent the fields already sent and not corrected from coming in again.) You can also avoid repeating field audits because, after the first time, you must audit only if the user has changed the field or a related one.

However, these gains are at the expense of extra programming and complexity, and therefore the savings in line time or audit path length must be considerable, and the probability of errors high, to justify this choice. You must add code to merge the new input with the old, and if you have turned off the MDTs, you must check both the length and the flag subfield to determine whether the operator has modified a map field. Fields with new data have a nonzero length; those which had data and were later erased have the high-order bit in the flag subfield on.

A good compromise is to save the data both ways. If the operator clears the screen, you use the saved data to refresh it; otherwise you use the data coming in from the screen. You do not need any merge logic, but you protect the operator from losing time over an unintended CLEAR.

For our “quick update” code, with its minimal audits and transmissions, we choose the “do nothing” approach and save the information about the screen.

Rechecking

The last requirement is to ensure that the input data is rechecked. If your task is conversational, this means repeating the audit section of your code after you have received (and merged, if necessary) the corrected input. In a pseudoconversational sequence, you typically repeat the transaction that failed. In the example, because we saved the data on the screen in such a way that corrected data is indistinguishable from new data, all we must do is arrange to execute the same transaction against the corrected data, thus:

```
  EXEC CICS RETURN TRANSID('QUPD') END-EXEC.
```

where 'QUPD' is the identifier of the “quick update” transaction.

Sending mapped output after mapped input

After a transaction has processed its initial input, the next step is to prepare and send the transaction output.

In general, if the output is to be mapped, you follow the steps outlined in “Sending BMS mapped output” on page 385 . However, the acquisition of storage for building the map may be affected by the input mapping you have already done. If the output and input maps are different, but in the same map set or in map sets defined to overlay one another, you have already done the storage acquisition during your input mapping process. If your output and input maps overlay one another, you need to ensure that you save any map input you still need and clear the output structure to nulls before you start building the output map. If this is awkward, you may want to define the maps so that they do not overlay one another. (See “BASE and STORAGE options” on page 386 for your choices in this regard.)

Your transaction may also call for using the same map for output as input. This is routine in code that handles input errors, as we have already seen, and also in simple transactions like "quick update". One-screen data-entry transactions are another common example.

When you are sending new data with a map already on the screen, you can reduce transmission with the DATAONLY option, and you may need only the SEND CONTROL command. See “Merging the symbolic and physical maps” on page 390 for a discussion of these options.

For the "quick update" transaction, however, we need to complete the message field with our "go" response (and update the file with the charge to finish our processing):

```
  MOVE 'CURRENT CHARGE OKAY; ENTER NEXT' TO MSGO
 ADD CHGI TO ACCTFILE-CUR-CHGS
 EXEC CICS REWRITE FILE('ACCT') FROM (ACCTFILE-RECORD)....
```

We also need to erase the input fields, so that the screen is ready for the next input. We have to do this both on the screen (the ERASEAUP option erases all unprotected fields) and in the output structure (because the output subfield overlays the input subfield and the input data is still there).

```
  MOVE LOW-VALUES TO ACCTNOO CHGO.
 EXEC CICS SEND MAP('QUPMAP') MAPSET('QUPSET') FROM(QUPMAPO)
 DATAONLY ERASEAUP END-EXEC.
```

Finally, we can return control to CICS , specifying that the same transaction is to be executed for the next input.

```
  EXEC CICS RETURN TRANSID('QUPD') END-EXEC.
```

MAPFAIL and other exception conditions

The MAPFAIL exception condition occurs when no usable data is transmitted from the terminal, or when the data transmitted is unformatted.

The MAPFAIL condition occurs on a RECEIVE MAP if the operator has used the CLEAR key or one of the PA keys. It also occurs if the operator uses ENTER or a PF key from a screen when both the following conditions apply:

- No fields defined in the map have the modified data tag set on (this means the operator did not key anything and you did not send any fields with the tags already set, so that no data is returned on the read).
- The cursor was not located in a field defined in the map and named, or the map did not specify CURSLOC=YES.

An operator might easily press ENTER too soon, or a "short read" key accidentally. To be user friendly, you might want to refresh the screen after a MAPFAIL condition, rather than ending the transaction in error.

A MAPFAIL condition also occurs if you issue a RECEIVE MAP without first formatting with a SEND MAP, or equivalent, in the current or a previous task, and can occur if you use a map that is different from the one you sent. This situation might signal an error in logic, or it might mean that your transaction is in its startup phase. For example, the quick update example does not allow for getting started; that is, for getting an empty map onto the screen so that the operator can start to use the transaction. You could use a separate transaction to do this, but you can take advantage of the code you need to refresh the screen after a MAPFAIL. The required code is as follows:

```
 IF RCV-RC = DFHRESP(MAPFAIL)
MOVE 'PRESS PF12 TO QUIT THIS TRANSACTION' TO MSGO
EXEC CICS SEND MAP('QUPMAP') MAPSET('QUPSET')
FROM(QUPMAPO) END-EXEC.
```

This code reminds the operator how to escape, because attempts to do this might be the original cause of the MAPFAIL. If you do not want to send this message, or if it was the default in the map, you can use the MAPONLY option:

```
 EXEC CICS SEND MAP('QUPMAP') MAPSET('QUPSET') MAPONLY
END-EXEC.
```

When MAPFAIL occurs, the input map structure is not cleared to nulls, as it is otherwise, so you must test for this condition if your program logic depends on this clearing.

You can issue a HANDLE CONDITION command to intercept MAPFAIL, as you can for other exception conditions. However, if you issue a HANDLE CONDITION command and you also have a HANDLE AID active for the AID you receive, control goes to the label specified for the AID and not that for MAPFAIL. For further explanation, see “Using the HANDLE AID command” on page 401 . In this situation, you will not be aware of the MAPFAIL, even though you issued a HANDLE for it, unless you also test EIBRESP.

EOC condition

EOC is another condition that you encounter frequently using BMS. It occurs when the end-of-chain (EOC) indicator is set in the request/response unit returned from z/OS Communications Server. EOC does not indicate an error, and the BMS default action is to ignore this condition.

Formatting other input

Although the data that you format with a RECEIVE MAP command normally comes from a terminal, you can also format data that did not come from a terminal, or that came indirectly.

For example, you might not know which map to use until you receive the input and inspect some part of it. This can happen when you use special hardware features like partitioning or logical device codes, and also in certain logic situations. You might also need to format data that was read from a formatted screen by an intermediate process (without mapping) and later passed to your transaction.

The FROM option of the RECEIVE MAP command addresses these situations. FROM tells BMS that the data has already been read, and only the translation from the native input stream to the input map structure is required.

Because the input has already been read, you need to specify its length if you use FROM, because BMS cannot get this information from the access method, as it does normally. If the data came originally from a RECEIVE command in another task, the length on the RECEIVE MAP FROM command should be the length produced by that original RECEIVE.

For the same reason, you cannot suppress translation to uppercase with the ASIS option when you use FROM. Moreover, BMS does not set EIBAID and EIBCURSR after a RECEIVE FROM command.

And finally, BMS does not know from what device the input came, and it assumes that it was your current principal facility. (You cannot even use RECEIVE FROM without a principal facility, even though no input/output occurs.) If the data came from a different type of device, you have to do the mapping in a transaction with a similar principal facility to get the proper conversion of the input data stream.

Note: You cannot map data read with a terminal control RECEIVE with the BUFFER option, because the input data is unformatted (in the 3270 sense). If you attempt to RECEIVE MAP FROM such input, MAPFAIL occurs.

BMS logical messages

The disposition options do not affect the correspondence between SEND MAP commands and pages of output.

You get one page for each SEND MAP command, unless you also use a second feature of full BMS, the ACCUM option. ACCUM allows you to build pages piecemeal, using more than one map, and like PAGING, it allows your message to exceed a page. You do not have to worry about page breaks or about tailoring your output to a specific page or screen capacity. BMS handles these automatically, giving you control at page breaks if you want. Details on cumulative page building are in “Cumulative output — the ACCUM option” on page 409.

As soon as you create an output message of more than one page, or a single page composed of several different maps, you are doing something BMS calls **cumulative** mapping. PAGING implies multiple pages, and ACCUM implies both multiple and composite pages, and so at the first appearance of either of these options, BMS goes into cumulative mapping mode and begins a **logical message** . The one-to-one correspondence between SEND commands and messages ends, and subsequent SEND MAPS add to the current logical message. Individual pages within the message are still disposed of as soon as they are complete, but they all belong to the same logical message, which continues until you tell BMS to end it.

Building logical messages

When you construct a logical message, you must observe a number of rules.

- You can build only one logical message at a time. If you are routing this message, BMS may create more than one logical message internally, but in terms of content, there is only one. After you complete the message and dispose of it, you can build another in the same task, using different options if you want.
- Options related to message management must be the same on all commands that build the message. These are:
  - the disposition option: PAGING, TERMINAL, or SET
  - the option governing page formation: ACCUM should be present on all commands or absent on all
  - the identifier for the message in CICS temporary storage: the REQID option value.

  Switching options mid-message results in the INVREQ condition or, in the case of REQID, the IGREQID condition.
- The ERASE, ERASEAUP, NLEOM, and FORMFEED options are honored if they are used on *any* of the BMS commands that contribute to the page.
- The values of the CURSOR, ACTPARTN, and MSR options for the page are taken from the most recent SEND MAP command, if they are specified there, and from the map if not.

- The 3270 write control character (WCC) from the most recent SEND MAP command is used. The WCC is assembled from the ALARM, FREEKB, PRINT, FRSET, L40, L64, L80, and HONEOM options in the command whenever *any* of them is specified. Otherwise, it is built from the same options in the map; options from the command are never mixed with those in the map.
- The FMHPARMs from all commands used to build the message are included.
- You can use both SEND MAP and SEND CONTROL commands to build a logical message, as long as the options previously noted are consistent. You can also build a logical message with a combination of SEND TEXT and SEND CONTROL commands. (SEND TEXT is an alternative to SEND MAP for formatting text output, covered in "The SEND TEXT command" on page 414 .) However, you cannot mix SEND MAP and SEND TEXT in the same message unless you are using partitions or logical device codes, subjects covered in "Partition support" on page 427 and "Logical device components" on page 433 respectively.

  There are also two special forms of SEND TEXT which allow combined mapping and text output, but to which other restrictions apply. See "SEND TEXT MAPPED and SEND TEXT NOEDIT" on page 417 for details.
- While you are building a logical message, you can still converse with your terminal. You cannot use BMS commands to write to the terminal unless you are also routing, but you can use BMS RECEIVE MAP commands and terminal control SEND and RECEIVE commands.

The SEND PAGE command

When you have completed a logical message, you notify BMS with a SEND PAGE command.

If you used the ACCUM option, SEND PAGE causes BMS to complete the current page and dispose of it according to the disposition option you established, as it has already done for any previous pages. If your disposition is TERMINAL, this last page is written to your principal facility; if SET, it is returned to the program; and if PAGING, it is written to temporary storage. If your disposition was PAGING, BMS also arranges delivery of the entire message to your principal facility. Options on the SEND PAGE command govern how this is done, as explained in "RETAIN and RELEASE" on page 407.

A SYNCPOINT command or the end of your task also ends a logical message, implicitly rather than explicitly. Where possible, BMS behaves as if you had issued SEND PAGE before your SYNCPOINT or RETURN, but you lose the last page of your output if you used the ACCUM option. Consequently, you should code SEND PAGE explicitly.

You also can delete an incomplete logical message if for some reason you decide not to send it. You use the PURGE MESSAGE command in place of SEND PAGE. PURGE MESSAGE causes BMS to delete the current logical message and associated control blocks, including any pages already written to CICS temporary storage. You can create other logical messages later in the same task, if you want.

RETAIN and RELEASE

Options on the **SEND PAGE** command tell BMS when and how to deliver the pages to the terminal.

When you complete a logical message with a disposition of PAGING, BMS arranges to deliver the entire logical message, which it has accumulated in temporary storage. The display or printing of pages can be done inline, immediately after the SEND PAGE command, but it is more common to schedule a separate task for the purpose. In either case, CICS supplies the programs required. These programs allow a terminal operator to control the display of the message, paging back and forth, displaying particular pages, and so on. When a separate task is used, it executes pseudoconversationally under transaction code CSPG. When the display is inline, the work is done (by the same CICS-supplied programs) within the task that created the message, which becomes conversational as a result.

You indicate how and when the message is sent by specifying RETAIN, RELEASE, or neither on your SEND PAGE command. The most common choice, and the default, is neither. It causes CICS to schedule the CICS-supplied transaction CSPG to display the message and then return control to the task. The CSPG transaction is queued with any others waiting to be executed at your terminal, which execute in priority sequence as the terminal becomes free. In the ordinary case, where no other tasks waiting, CSPG executes as soon as the creating task ends.

Note: The terminal must be defined as allowing automatic transaction initiation for CICS to start CSPG automatically (ATI(YES) in the associated TYPETERM definition). If it is not, the operator has to enter the

transaction code CSPG or one of the paging commands to get the process started when neither RELEASE nor RETAIN is specified.

The RELEASE option works similarly, but your task does not regain control after SEND PAGE RELEASE. Instead, BMS sends the first page of the message to the terminal immediately. It then ends your task, as if a CICS RETURN had been executed in the highest level program, and starts a CSPG transaction at your terminal so that the operator can display the rest of the pages. The CSPG code executes pseudoconversationally, just as it does if you specify neither RELEASE nor RETAIN, and the original task remains pseudoconversational if it was previously.

There are two other distinctions between RELEASE and using neither option:

- RELEASE allows you to specify the transaction identifier for the next input from the terminal, after the operator is through displaying the message with CSPG.
- RELEASE also permits the terminal operator to chain the output from multiple transactions (see "Terminal operator paging: the CSPG transaction" on page 408).

SEND PAGE RETAIN causes BMS to send the message immediately. When this process is complete, your task resumes control immediately after the SEND PAGE command. When the terminal is a display, BMS provides the same operator facilities for paging through the message as the CSPG transaction does, but within the framework of your task. The code that BMS uses for this purpose issues RECEIVE commands to get the operator's display requests, and this causes your task to become conversational.

Note: If an error occurs during the processing of a SEND PAGE command, the logical message is not considered complete and no attempt is made to display it. BMS discards the message in its clean up processing, unless you arrange to regain control after an error. If you do, you can either delete the logical message with a PURGE command or retry the SEND PAGE. You should not retry unless the error that caused the failure has been remedied.

The AUTOPAGE option

Options on the **SEND PAGE** command tell BMS how to deliver the pages to the terminal.

For display terminals, you want CSPG to send one page at a time, at the request of the terminal operator. For printers, you want to send one page after another. You control this with the AUTOPAGE or NOAUTOPAGE option on your SEND PAGE command. NOAUTOPAGE lets the terminal operator control the display of pages; AUTOPAGE sends the pages in ascending sequence, as quickly as the device can accept them. If you specify neither, BMS determine which is appropriate from the terminal definition.

Note: If your principal facility is a printer, you can sometimes use a disposition of TERMINAL rather than PAGING, because successive sends to a printer do not overlay one another as they do on a display. TERMINAL has less overhead, especially if you do not need ACCUM either, and thus avoid creating a logical message.

Terminal operator paging: the CSPG transaction

The CICS -supplied paging transaction, CSPG, allows a user at a terminal to display individual pages of a logical message by entering page retrieval requests. You define the transaction identifiers for retrieval and other requests supported by CSPG in the system initialization table; sometimes program function keys are used to minimize operator effort.

Retrieval can be sequential (next page or previous page) or random (a particular page, first page, last page). In addition to page retrieval, CSPG supports the following requests:

Page copy
Copy the page currently on display to another terminal. BMS reformats the page if the target terminal has a different page size or different formatting characteristics, provided the terminal is of a type supported by BMS.

Message query
List the messages waiting to be displayed at the terminal with CSPG. The list contains the BMS-assigned message identifier and, for a routed message, the message title, if the sender provided one.

Purge message
Delete the logical message.

Page chaining
Suspend the current CSPG transaction after starting to display a message, execute one or more other transactions, and then resume the original CSPG display. An intervening transaction may itself produce BMS or terminal output. If this output is a BMS logical message created with the PAGING and RELEASE or RETAIN options, this message is “chained” to the original one, and the operator can switch between one and the other.

Switch to autopage
Switch from NOAUTOPAGE display mode to AUTOPAGE mode. For terminals that combine a keyboard and hard copy output, this allows an operator to purge or print a message based on inspection of specific pages.

The process of examining pages continues until the operator signals that the message can be purged. CSPG provides a specific request for this purpose, as previously noted. If the SEND PAGE command contained the option OPERPURGE, this request is the only way to delete the message and get control back from CSPG.

If OPERPURGE is not present, however, any input from the terminal that is not a CSPG request is interpreted as a request to delete the message and end CSPG. If the message was displayed with the RETAIN option, the non-CSPG input that terminates the display can be accessed with a BMS or terminal control RECEIVE when the task resumes execution. See CSPG - page retrieval for detailed information about the CSPG transaction.

Logical message recovery

Logical messages created with a disposition of PAGING are kept in CICS temporary storage between creation and delivery.

BMS constructs the temporary storage queue name for a message from the 2-character REQID on the SEND commands, followed by a six-position number to maintain uniqueness. If you do not specify REQID, BMS uses a value of two asterisks (**).

Temporary storage can be a recoverable resource, and therefore logical messages with a disposition of PAGING can be recovered after a CICS abend. In fact, because CICS bases the recoverability of temporary storage on generic queue names, you can make some of your messages recoverable and others not, by your choice of REQID for the message. The conditions under which logical messages are recoverable are described in Troubleshooting for recovery processing.

Routed messages are eligible for recovery, as well as messages created for your principal facility. Routing is described in “Message routing” on page 418.

Cumulative output — the ACCUM option

The ACCUM option allows you to build your output cumulatively, from any number of SEND MAP commands and less-than-page-size maps.

Without it, each SEND MAP command corresponds to one page (if the disposition is PAGING), or a whole message (if TERMINAL or SET). With ACCUM, however, BMS formats your output but does not dispose of it until either it has accumulated more than fits on a page or you end the logical message. You can intercept page breaks if you want, or you can let BMS handle them automatically.

Page size is determined by the PAGESIZE or ALTPAGE value in the terminal definition. PAGESIZE is used if the PROFILE under which your transaction is running specifies the default screen size, and ALTPAGE is used if it indicates alternate screen size. (Unlike screen size, page size is not affected by the DEFAULT and ALTERNATE options that you can include with the ERASE command.)

Floating maps: how BMS places maps using ACCUM

Maps can *float* - that is, they can be positioned relative to maps already written to the same page and to any edge of the page.

Floating maps save program logic when you need to support multiple screen sizes or build pages piecemeal out of headers, detail lines and trailers, where the number of detail lines depends on the data.

The BMS options that allow you to do this are:

- JUSTIFY
- HEADER and TRAILER
- Relative values (NEXT and SAME) for the LINE and COLUMN options

When you are building a composite screen with the ACCUM option, the position on the screen of any particular map is determined by:

- The space remaining on the screen at the time it is sent
- The JUSTIFY, LINE and COLUMN option values in the map definition

The space remaining on the page, in turn, depends on:

- Maps already placed on the current page.
- Whether you are participating in “overflow processing”, that is, the processing that occurs at page breaks. If you are, the sizes of the trailer maps in your map sets are also a factor.

The placement rules we are about to list apply even if you do not specify ACCUM, although JUSTIFY values of FIRST and LAST are ignored. However, without ACCUM, each SEND MAP corresponds to a separate page, and thus the space remaining on the page is always the whole page.

Page breaks: BMS overflow processing

When you build a mapped logical message, you can ask BMS to notify you when a page break is about to occur; that is, when the map you just sent does not fit on the current page.

This feature is useful when you are forming composite pages with ACCUM. It allows you to put trailer maps at the bottom of the current page and header maps at the top of the next one, number your pages, and so on.

BMS gives your program control at page breaks if either:

- You have issued a HANDLE CONDITION command naming a label for the OVERFLOW condition, or
- You specify the NOFLUSH option with either the RESP or the NOHANDLE option on your SEND MAP commands.

Either of these actions has two effects:

- The calculation of the space remaining on the page changes. Unless the map you are sending is itself a trailer map, BMS assumes that you eventually want one on the current page. It therefore reserves space for the largest trailer in the same map set. (The largest trailer map is the one containing the TRAILER option that has the most lines.) If you do not intercept page breaks (or if you send a trailer map), BMS uses the true end of the page to determine whether the current map fits.
- The flow of control changes if the map does not fit on the current page. On detecting this situation, BMS raises the OVERFLOW condition. Then it returns control to your task *without* processing the SEND MAP command that caused the overflow. Control goes to the location named in the HANDLE CONDITION command if you used one. With NOFLUSH, control goes to the statement after the SEND MAP as usual, and you need to test the RESP value or EIBRESP in the EIB to determine whether overflow occurred.

When your program gets control after overflow, it should:

- Add any trailer maps that you want on the current page. BMS leaves room for the one with the most lines in the map set you just used. If this is not the right number of lines to reserve, or if you are using several map sets, you can ensure the proper amount by including a dummy map in any map set that

may apply. The dummy map must specify TRAILER and contain the number of lines you want to reserve; you do not need to use it in any SEND MAP commands.

- Write any header maps that you want at the top of the next page.
- Resend the map that caused the overflow in the first place. You must keep track of the data and map name at the time the overflow occurs; BMS does not save this information for you. Note that if you have several SEND MAP commands which might cause overflow, the program logic required to determine which one you need to reissue is more complex if you use HANDLE CONDITION OVERFLOW than if you use NOFLUSH.

Once OVERFLOW is turned on, BMS suspends returning control to your program when the output does not fit on the current page, although it still uses overflow rules for calculating the remaining space. OVERFLOW remains on until BMS processes the first SEND MAP naming a map which is *not* a header or a trailer. This allows you to send your trailers and headers without disabling your HANDLE CONDITION for OVERFLOW or changing your response code tests, and reinstates your overflow logic as soon as you return to regular output. (Resending the map that originally caused overflow is usually the event that turns off the overflow condition.)

If you do not intercept overflows, BMS does not notify your program when a page break occurs. Instead, it disposes of the current page according to the disposition option you have established and starts a new page for the map that caused the overflow.

Map placement rules

The primary placement of maps on the screen is from top to bottom. You can place maps side-by-side where space permits, provided you maintain the overall flow from top to bottom.

The precise rules for a given SEND MAP ACCUM command are as follows:

1. The highest line on which the map might start is determined as follows:
   a. If the map definition contains JUSTIFY=FIRST, BMS goes immediately to a new page (at Step “5” on page 412), unless the only maps already on the page are headers placed there during overflow processing. In this case, BMS continues at Step “1.c” on page 411.
   b. If the map specifies JUSTIFY=LAST, BMS starts the map on the lowest line that allows it to fit on the page. If the map is a trailer map or you are not intercepting overflows or you are already in overflow processing, BMS uses all the space on the page. Otherwise, BMS places the map as low on the page as it can while still retaining room for the largest trailer map. If the map fits vertically using this starting line, processing continues at Step “3” on page 411 (the LINE option is ignored if JUSTIFY=LAST); if not, overflow occurs (Step “5” on page 412).

      Note: JUSTIFY=BOTTOM is the same as JUSTIFY=LAST for output operations with ACCUM. (There are differences without ACCUM and for input mapping; see SEND MAP).
   c. If there is no vertical JUSTIFY value (or after any overflow processing caused by JUSTIFY=FIRST has been completed), the LINE operand is checked. If an absolute value for LINE is given, that line is used, provided it is at or below the starting line of the map most recently placed on the page. If the value is above that point, BMS goes to a new page at Step “5” on page 412.

      If LINE=NEXT, the first completely unused line (below all maps currently on the page) is used. If LINE=SAME, the starting line of the map sent most recently is used.
2. BMS now checks that the map fits vertically on the screen, given its tentative starting line. Here again, BMS uses all of the space remaining if the map is a trailer map, if you are not intercepting overflows or if you are already in overflow processing. Otherwise, BMS requires that the map fit and still leave space for the largest trailer map. If the map does not fit vertically, BMS starts a new page (Step “5” on page 412).
3. Next, BMS checks whether the map fits horizontally, assuming the proposed starting line. In horizontal positioning, the JUSTIFY option values of LEFT and RIGHT come into play. LEFT is the default, and means that the COLUMN value refers to the left-hand side of the map. A numeric value for COLUMN tells where the left edge of the map should start, counting from the left side of the page. COLUMN=NEXT starts the map in the first unused column from the left on the starting line.

COLUMN=SAME means the left-hand column of the map most recently placed on the screen which also specified JUSTIFY=LEFT and which was not a header or trailer map.

JUSTIFY=RIGHT means that the COLUMN value refers to the right-hand edge of the map. A numeric value tells where the right edge of the map should start, *counting from the right* . COLUMN=NEXT means the first available column from the right, and COLUMN=SAME is the right-hand column of the map most recently placed which had JUSTIFY=RIGHT and was not a header or trailer.

If the map does not fit horizontally, BMS adjusts the starting line downward, one line at a time, until it reaches a line where the map does fit or overflow occurs. Processing resumes with the vertical check (Step “2” on page 411) after each adjustment of the starting line.

4. If the map fits, BMS adds it to the current page and updates the available space, using the following rules:
   - Lines above the first line of the map are completely unavailable.
   - If the map specifies JUSTIFY=LEFT, the columns from the left edge of the page through the right-most column of the map are unavailable on the lines from the top of the map through the last line on the page that has anything on it (whether from this map or an earlier one).
   - If the map specifies JUSTIFY=RIGHT, the columns between the right-hand edge of the page and the left-hand edge of the map are unavailable on the lines from the top of the map through the last line of the page that has anything on it.

   Figure 118 on page 412 shows how the remaining space is reduced with each new map placed.

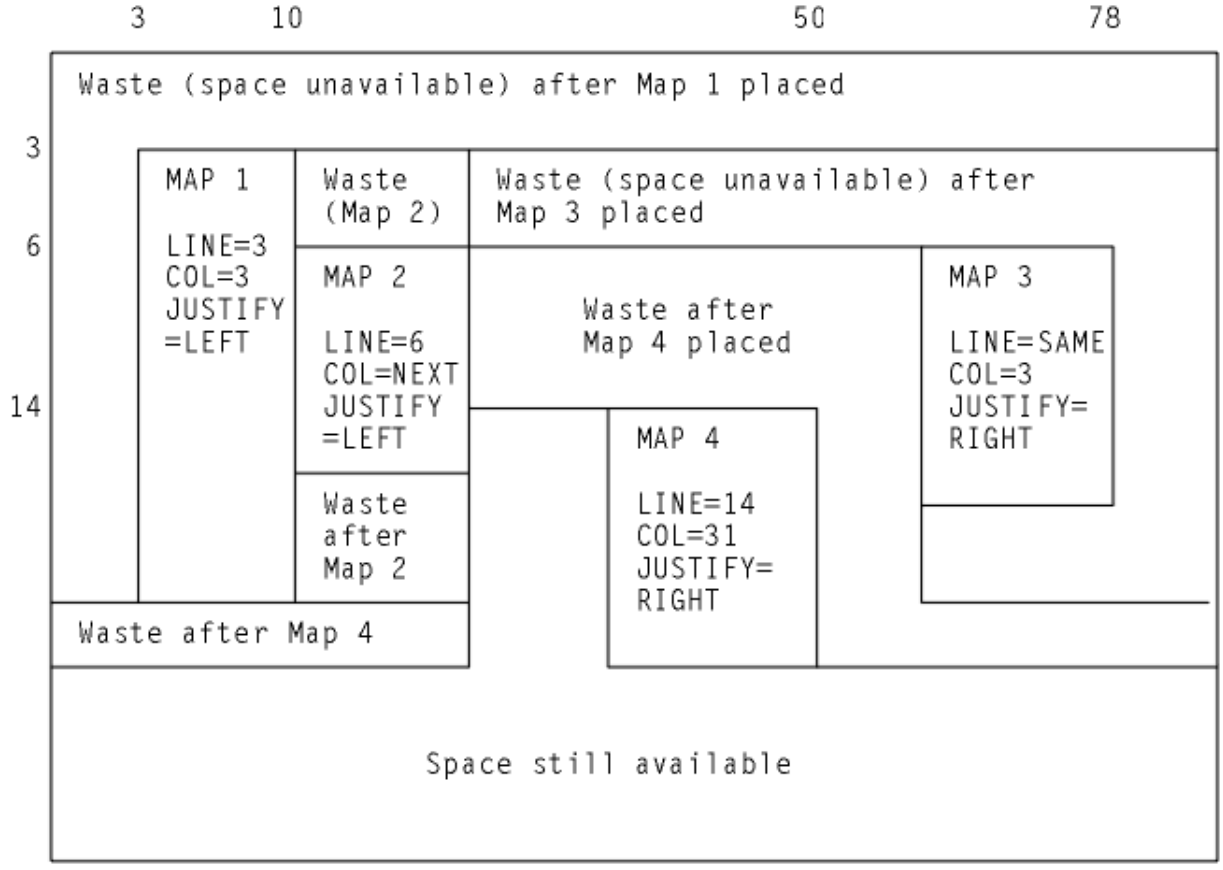


Figure 118. Successive placement of maps on a page, showing space made unavailable by each

5. When the current map does not fit on a page, BMS determines whether it should return control to your program. If you have asked for control at overflow and you are not already in overflow processing, BMS returns control as described in “Page breaks: BMS overflow processing” on page 410 . Otherwise, BMS disposes of the current page according to the disposition option you have established, starts a new page, and resumes processing for the map that would not fit at Step “1” on page 411.

ASSIGN options for cumulative processing
To help you manage the complexities of building a composite screen, CICS provides ASSIGN command options that relate specifically to cumulative processing.

The following options apply to cumulative processing:

- MAPCOLUMN
- MAPHEIGHT
- MAPLINE
- MAPWIDTH

All apply to the map most recently sent. MAPHEIGHT and MAPWIDTH are the size (number of rows and columns) and MAPLINE and MAPCOLUMN are the origin of the map (the position of the upper left corner).

Input from a composite screen
You can read mapped input from a screen built from multiple maps, but there are restrictions.

First, you can specify only one map in your RECEIVE MAP command, whereas the screen may have been written with several.

Second, BMS cannot know how to position a floating map for input and assumes that the map in your RECEIVE MAP command was written to an empty screen. Thus LINE or COLUMN values of NEXT or SAME are interpreted differently on input than on output. JUSTIFY=LAST is ignored altogether; you should use JUSTIFY=BOTTOM if you want to place a map at the bottom of the screen and read data back from it.

Performance considerations
There are three components to the overall efficiency of the part of your application that the user sees: processor path length, communications line utilization, and user time.

As online systems have evolved, the emphasis has shifted steadily to making things as easy and quick for the user as possible, at the expense of the other factors if necessary. Also, as processors have become cheaper, designers have been willing to expend cycles to reduce programming and maintenance effort as well as to make the user interface better.

Follow the suggestions to reduce path length and line time. Judge for yourself whether and to what extent the savings justify extra programming effort or a less general design.

Minimizing path length
Ordinarily, the number of instructions executed in a single CICS command is large in comparison to the number of instructions in the application program that invoked it. The path length for a given task depends more on the number and type of CICS commands than on anything else, and commands are the most fertile area for tuning.

Commands vary by type, and path length for any given command can vary considerably with circumstances.

For BMS, some recommendations are:

- Build screens (pages) with a single command when practical. Avoid building a composite screen with the ACCUM feature when a modest amount of additional programming accomplishes the same function, and avoid building a composite screen by multiple physical writes, as described in Outside the map in “Building the output screen” on page 391 , except in unusual circumstances.
- Avoid producing more output at one time than the user is likely to inspect. Some transactions—inquiries, especially—produce many pages of output for certain input values. When this happens, the user typically narrows the search and repeats the inquiry, rather than page through the initial output. To avoid the path length of producing output that is never viewed, you can limit it to some reasonable number of pages, inform the user on the last page that there is more, and save the information required to restart the search from that point if the user requests it. The extra programming is minimal; see “Sharing data across transactions” on page 104 for ways to save the restart data.

- Use commands that are on the BMS "fast path" if possible. (See Minimum BMS in "BMS support levels" on page 367 for the commands and terminal types that qualify.)
- Use terminal control commands for very simple inputs an outputs, where you do not need BMS formatting or other function. If path length is critical, you might want to use terminal control entirely. However, the advantages of BMS over terminal control in terms of flexibility, initial programming effort and maintainability are significant, and typically outweigh the path length penalty.

Reducing message lengths
You can take advantage of 3270 hardware to reduce the length of both inbound and outbound messages. If the bandwidth in any link between the terminal and the processor is constrained, you get better response overall with shorter messages.

However, the time for any given transmission depends on the behavior of other users of those links at the time, and so you might not see improvement directly. Here are some of the possibilities for reducing the length of a 3270 datastream:

- Avoid turning on MDTs unnecessarily when you send a screen, because they cause the associated input fields to be transmitted on input. Ordinarily, you do not need to set the tag on, because the hardware does this when the user enters input into the field. The tag remains on, no matter how many times the screen is transmitted, until explicitly turned off by program (by FRSET, ERASEAUP, or ERASE, or by an override attribute byte with the tag off). The only time you need to set it on by program is when you want to store data on the screen in a field that the user does not modify, or when you highlight a field in error and you want the field returned whether the user changes it. In this case you need to turn on the MDT as well as the highlighting.
- Use FRSET to reset the MDTs when you do not want input on a screen retransmitted (that is, when you have saved it and the user does not need to change it on a subsequent transmission of the same screen).
- Do not initialize input fields to blanks when you send the screen because, on input, blanks are transmitted and nulls are not. Hence the data stream is shortened by the unused positions in each modified field if you initialize with nulls. The appearance on the screen is the same, and the data returned to the program is also the same, if you map the input.
- For single-screen data entry operations, use ERASEAUP to clear data from the screen, rather than resending the screen.
- If you are updating a screen, send only the changed fields, especially if the changes are modest, as when you highlight fields in error or add a message to the screen. In BMS, you can use the DATAONLY option, both to shorten the data stream and reduce the path length (see the DATAONLY option in "Merging the symbolic and physical maps" on page 390). To highlight a field, in fact, you send only the new attribute byte; the field data remains undisturbed on the screen.
- If you are using terminal control commands, format with set buffer address (SBA) and repeat-to-address (RA) orders, rather than spacing with blanks and nulls. (BMS does this for you.)

Text output

If you are sending text output to a terminal, and you do not need to format the screen for subsequent input, you do not need to create a map. BMS provides the **SEND TEXT** command expressly for this purpose.

When you use SEND TEXT, BMS breaks your output into pages of the proper width and depth for the terminal to which it is directed. Lines are broken at word boundaries, and you can add header and trailer text to each page if you want. Page size is determined as it is for other BMS output (see "The SEND PAGE command" on page 407).

The SEND TEXT command

Except for the different type of formatting performed, the SEND TEXT command is very similar to SEND MAP. You specify the location of the text to be formatted in the FROM option and its length in the LENGTH option.

Nearly all the options that apply to mapped output apply to text output as well, including:

Device controls
FORMFEED, ERASE, PRINT, FREEKB, ALARM, CURSOR.

Formatting options
NLEOM, L40, L64, L80, HONEOM.

Disposition options
TERMINAL, PAGING, SET.

Page formation option
ACCUM.

In general, these options have the same meaning on a SEND TEXT command as they do on a SEND MAP command. The SEND TEXT command itself requires **standard** BMS; options like ACCUM, PAGING and SET that require **full** BMS in a mapped environment also require full BMS in a text environment.

There are also options for SEND TEXT that correspond to functions associated with the map in a SEND MAP context. These are HEADER, TRAILER, JUSTIFY, JUSFIRST and JUSLAST. We explain how they work in "Text pages" on page 415.

Two SEND MAP options that do not carry over to SEND TEXT are ERASEAUP and NOFLUSH. ERASEAUP does not apply because text uses fields only minimally, and NOFLUSH does not apply because BMS does not raise the OVERFLOW condition on text output.

Text logical messages
The presence of either the ACCUM or PAGING option on a SEND TEXT command signals BMS that you are building a logical message, just as it does in a SEND MAP command.

Text logical messages are subject to the same rules as mapped logical messages (see "Building logical messages" on page 406). In particular, you can use both SEND TEXT and SEND CONTROL commands to build your message, but you cannot mix in SEND MAPs, except as noted there. You also end your message in the same way as a mapped message (see "BMS logical messages" on page 406).

Text pages
Page formation with SEND TEXT is somewhat different from page formation with SEND MAP.

First, a single SEND TEXT command can produce more output than fits on a screen or a printer page (SEND MAP never does this). BMS sends the whole message, which means that you can deliver a multi-page message to a printer without using logical facilities. You cannot use the same technique for displays, however, because even though BMS delivers the whole message, the component screens overlay one another, generally too quickly for anyone to read.

If you specify ACCUM, BMS breaks the output into pages for you, and the second difference is that unless you specify a disposition of SET, your task does not get control at page breaks. Instead, when the current page has no more room, BMS starts a new one. It adds your header and trailer, if any, automatically, and does not raise the OVERFLOW condition. This is true whether you produced the pages with a single SEND TEXT command or you built the message piecemeal, with several. The only situation in which your task is informed of a page break is when the disposition is SET. In this case, BMS raises the RETPAGE condition to tell you that one or more pages are complete, as explained in "Using SET" on page 394.

Here are the details of how BMS builds pages of text with ACCUM:

1. Every message starts on page 1, which is initially empty.
2. If you specify the HEADER option, BMS starts every page with your header text. BMS numbers your pages in the header or trailer if you want. (Header format and page numbering are explained in "Header and trailer format " on page 416 .)
3. If you specify one of the justification options (JUSTIFY, JUSFIRST, JUSLAST), BMS starts your text on the indicated line. JUSFIRST begins your text on the first line after the header, or the top line if there is no header. JUSTIFY=n starts your text on line n, and JUSLAST starts it on the lowest line that allows both it and your trailer (if any) to fit on the current page. If the contents of the current page prevent BMS from honoring the justification option there, BMS goes to a new page first, at step "6" on page 416.

Justification applies only to the start of the data for each SEND TEXT command; when the length of your data requires an additional page, BMS continues your text on it in the first available position there.

4. If you do not specify justification, BMS starts your text in the first position available. On the first SEND TEXT of the message, this works out the same as JUSFIRST. Thereafter, your text follows one character after the text from the previous SEND TEXT of the current logical message. (The intervening character is an attributes byte on 3270 terminals, a blank on others.)
5. Having determined the starting position, BMS continues down the page, breaking your data into lines as explained in "Text lines" on page 416 , until it runs out of space or data. If you have specified a trailer, the available space is reduced by the requirement for the trailer. If the data is exhausted before the space, processing ends at this point. The message is completed when you indicate that you are finished with a SEND PAGE or PURGE MESSAGE command.
6. If you text does not fit on the current page, BMS completes it by adding your trailer text, if any, at the bottom and disposes of it according to the disposition option you have established (TERMINAL, PAGING, or SET), just as it does for a mapped logical message. The trailer is optional, like the header; you use the TRAILER option to specify it (see "Header and trailer format " on page 416).
7. BMS then goes to a new page and repeats from step "2" on page 415 with the remaining data.

Text lines

BMS uses well-defined rules to break text into lines for display on a terminal.

In breaking the text into lines, BMS uses the following rules:

1. Ordinarily, each line starts with what appears to be a blank. On a 3270 device, this is the attributes byte of a field that occupies the rest of the line on the screen or printed page. For other devices, it is a blank or a carriage control character.

   An exception occurs if the task creating the output is running under a PROFILE that specifies PRINTERCOMP(YES) and the output device is a 3270 printer. In this case, no character is reserved at the beginning of each line. See "PRINTERCOMP option" on page 350.
2. BMS copies your text character for character, including all blanks, with two exceptions that occur at line end:

   - If a line ends in the middle of a word, BMS fills out the current line with blanks and places the word that would not fit in the first available position of the next line. For this purpose, a "word" is any string of consecutive nonblank characters.
   - If two words are separated by a single blank, and first one fits on the current line without leaving room for the blank, the blank is removed and the next line starts at the beginning of the second word.
3. You can embed new-line (NL) characters and other print format orders as well as blanks to control the format, if the destination terminal is a printer. NLs and tabs are particularly useful with columnar data, and BMS does not filter or even interpret these characters. However, print format orders do not format displays; see "CICS 3270 printers" on page 347 for more information about using them.
4. You can also include set attribute (SA) order sequences in your output. (Each one sets the attributes of a single character in the data stream, as explained in "The set attribute order" on page 281 .) BMS abends the task unless SA sequences are exactly three bytes long and represent a valid attribute type. However, if you use a valid SA sequence to a terminal that does not support the attribute, BMS removes the SA sequence and then sends the message. Attributes set with SA orders remain until overridden by subsequent orders or until another SEND TEXT command, which resets them to their default values.

   You should not include 3270 orders other than SA in your text. BMS treats them as display data and they do not format as intended; they may even cause a terminal error.

Header and trailer format

To place a header or trailer on the pages of a text message, you point to a block of data with a set format.

To place a header on the pages of a text message, you point to a block of data in the following format in the HEADER option:

You use the same format for trailer text, but you point to it with the TRAILER option. Here:

LL
is the length of the header (trailer) data, not including the four bytes of LL, P, and C characters. LL should be expressed in halfword binary form.

P
is the page-number substitution character (see PNFLD below). Use a blank if you do not want page numbers.

C
is a reserved 1-byte field.

TEXT
is the header (trailer) text to be placed at the top (bottom) of each page of output. Use new-line characters (X'15') to indicate where line breaks should occur if you want multiple lines.

PNFLD
is the page number field within your header (trailer) text. If you want to number the pages of your output, choose a character that does not otherwise appear in your header (trailer) text. Place this character in the positions where the page number is to appear. You can use from one to five adjacent positions, depending on how large you expect your page numbers to get (32,767 is the maximum BMS allows). Place the same character in the P field above, to tell BMS where to make the substitution. Do not use X'0C' , X'15' , X'17' , X'26' , or X'FF' for P; these values are reserved for other purposes. If you do not want page numbering, place a blank (X'40') in P.

When you are building a logical message, you should repeat your HEADER and TRAILER options on each SEND TEXT command, so that they are present when the page breaks occur, and you need to specify the trailer again on the SEND PAGE command that terminates the message.

Here is an example of a COBOL definition for a header that numbers the pages, leaving room for a number up to 99.

```
 EXEC CICS SEND TEXT FROM (OUTPUT-AREA)
HEADER(HEADER-TEXT) PAGING ACCUM END-EXEC.
```

where:

```
 01 HEADER-TEXT
02 HEADER-LL PIC S9(4) COMP VALUE +11.
02 HEADP PIC X VALUE '@'.
02 FILLER PIC X VALUE LOW-VALUE.
02 HEADING PIC X(11) VALUE 'PAGE NO. @@'.
```

Screens built with SEND TEXT are not designed for extensive input from the terminal operator. However, you can interpret the attention identifier and read simple inputs—such as those used in the CSPG transaction to control the page display—if the field structure on the screen is suitable and the operator knows or can see what is expected. (A new field starts at each line, as well as at the first character of the text sent with each SEND TEXT command that made up the message. The fields defined are unprotected, alphameric and normal intensity, so that the operator can key into them.) Normally a terminal control RECEIVE is used in this situation; you can use RECEIVE MAP only if you can build a map with a field structure matching that of the screen.

SEND TEXT MAPPED and SEND TEXT NOEDIT

BMS provides two special forms of the SEND TEXT command that allow you to use some of the message delivery facilities of BMS for output that is already formatted. SEND TEXT MAPPED sends a page of device-dependent data previously built by BMS and captured with the SET option. You might have used either SEND MAP or SEND TEXT commands to build the page originally.

SEND TEXT NOEDIT is similar, but is used to send a page of device-dependent output built by the program or some method other than BMS.

You can deliver such pages to your own principal facility individually, using a disposition of TERMINAL, or you can include them in a logical message built with the PAGING option. In a logical message, these forms can be mixed with ordinary SEND TEXT commands or with SEND MAP commands, as long as each BMS SEND represents a separate page (that is, the ACCUM option is not used). The usual restriction against mixing text with mapped output in the same logical method does not apply here, because the page is already formed.

You can also use these commands in a routing environment. Whether you are routing or sending to your own terminal, you must ensure that the data stream is appropriate to the destinations; BMS does not check before transmission, other than to remove 3270 attributes that the destination does not support.

None of the page-formatting options, ACCUM, JUSTIFY, JUSFIRST, JUSLAST, HEADER, and TRAILER, apply to either of these commands, because the page is already formatted and built, by definition.

The primary difference between the MAPPED and NOEDIT forms is that SEND TEXT MAPPED uses the 4-byte page control area (PGA) that BMS appends to the end of pages returned by the SET option. This area tells BMS the write command and write control character to be used, which extended attributes were used on the page, and whether the page contains formfeeds, data in SCS format, 14- or 16-bit buffer addresses, structured fields and FMHs. It allows BMS to deliver a page built for one device to one with different hardware characteristics, as might be required for a page copy or a routing operation. With SEND TEXT NOEDIT, you specify this type of information on the command itself. You should use SEND TEXT MAPPED for output created with BMS, and NOEDIT for output formatted by other means. You cannot include structured fields in the output with either SEND TEXT MAPPED or SEND TEXT NOEDIT, incidentally; you must use a terminal control SEND for such output.

The LENGTH option for a SEND TEXT MAPPED command should be set from the TIOATDL value returned when the page was built, which does not include the PGA. If you copy the page for later use with SEND TEXT MAPPED, however, you must be sure to copy the PGA as well as the page itself (TIOATDL + 4 bytes in all).

Message routing

The message routing facilities of BMS allow you to send messages to terminals other than the principal facility of your task. Your task does not even need to have a principal facility.

Routing does not give your task direct control of these terminals, but instead causes the scheduling of a task for each destination to deliver your message. These tasks execute the CICS -supplied transaction CSPG, the same one used for delivery of messages with a disposition of PAGING to your own terminal. Thus the operator at a display terminal who receives a routed message uses CSPG requests to view the message. (See “Terminal operator paging: the CSPG transaction” on page 408 for more information about CSPG.)

Message routing is useful for message-switching and broadcasting applications, and also for printing (see “Using CICS printers” on page 353). It is the basis for the CICS-supplied transaction CMSG, with which terminal users can send messages to other terminals and users. See CMSG - message switching for an explanation of CMSG and what you can do with it.

To route a message, you start by issuing a ROUTE command. This command tells BMS where to send the message, when to deliver it, what to do about errors, and other details. Then you build your message. It can be a mapped or text message, but it must be a logical message (that is, either ACCUM or PAGING present), and the disposition must be either PAGING or SET, not TERMINAL. PAGING is the more common choice and is assumed in the discussion that follows. We explain SET in a routing context in “Routing with SET” on page 424.

Your ROUTE command is in effect until you end your message with a SEND PAGE command, and you must not issue another one until this occurs. (If you issue ROUTE while building your message you get an invalid request response; if you issue a second ROUTE before you start your logical message, it replaces the first one.) You can also terminate a message with PURGE MESSAGE, if you decide you do not want to

send it. PURGE MESSAGE causes the routing environment established with your ROUTE command to be discarded, along with your logical message.

Message destinations

Destinations can be specified for your routed message in several ways: by requesting certain classes of operator, naming a particular operator or naming a particular terminal.

- You can request that certain classes of operators receive the message, by using the OPCLASS option of the ROUTE command. Classes are associated with an operator in the RACF user definition or a CICS sign-on table entry.
- You can name particular operators who are to receive the message by using a route list, to which you point with the LIST option of the ROUTE command. Operators are identified by a three-character OPIDENT value, which is also assigned in the RACF definition or a sign-on table entry.
- You can name particular terminals which are to receive the message; this is also done with a route list. Terminals are identified by their 4-character TERMID value, and, for terminal types to which they apply, a 2-character logical device code.

Note: If you need to know the identifier or operator class values for the operator signed on at your principal facility to specify the destination of your routed message, you can use the ASSIGN command with the OPID or OPCLASS options to find out.

Eligible terminals

To format a message properly for a particular destination, BMS needs to know the characteristics of the terminal for which it is formatting. This is true even for destinations that you designate by operator or operator class.

The first step in processing a route list, therefore, is to translate your destinations to a list of *terminals* to which the message *can* be delivered. This “eligible terminal” list combines the information in your route list and your OPCLASS specification with the state of the terminal network *at the time of the ROUTE command*.

Later, when your message is ready for delivery, BMS uses this list to decide which terminals get your message. A terminal must be on the list to receive the message, but its presence there does not guarantee delivery. There might be operator restrictions associated with the terminal and, because delivery occurs later in time, the status or even the nature of the terminal can have changed.

Both at the time the list is built and at the time of delivery, BMS is restricted to the terminal definitions installed in its own CICS region (where the routing task is running, or ran) and might not have all the terminal definitions you expect. First, terminals that are autoinstalled might not be logged on either at the time of the ROUTE, excluding them from inclusion on the list, or at the times sending is attempted, preventing delivery.

In a multiple-region environment, there is the additional possibility that terminals known to one region might not be known to another. It depends on how they are defined, as explained in TYPETERM attributes . In particular, if a terminal definition is shared among regions by designating it as SHIPPABLE in the region that owns it, the terminal is not defined in any other region until something occurs to cause shipment there. This typically happens the first time the terminal routes a transaction to the region in question. Consequently, a ROUTE in this region cannot include the terminal before the first such event occurs.

The following sections describe how BMS builds the list of eligible terminals. This occurs at the time of the ROUTE command:

Destinations specified with OPCLASS only

If you specified operator classes (the OPCLASS option) but no route list, BMS scans all the terminal definitions in the local system. Any terminal that meets all these conditions gets on the eligible terminal list:

- The terminal is of a type supported by BMS.

- The terminal can receive routed messages not addressed to it (ROUTEDMSGS (ALL) in the terminal definition).
- An operator is signed on at the terminal.
- The operator belongs to one of the operator classes in your OPCLASS list.

The resulting entry is marked so that delivery occurs only when and if an operator belonging to at least one of the operator classes in your OPCLASS list is signed on. (This operator does not have to be the one that was signed on at ROUTE time.)

OPCLASS and LIST omitted

If you do not specify the operator class or a route list, BMS puts every terminal that meets the first two conditions above on the list, and sets no operator restrictions on delivery. In a network where many terminals are eligible to receive all routed messages, this is a choice you almost certainly want to avoid.

Route list provided

If you provide a route list, BMS builds its list from yours instead of scanning the terminal definitions. Each of your entries is processed as follows. Processing includes setting a status flag in the list entry to tell you whether the entry was used or skipped and why.

- If the entry contains a terminal identifier but no operator identifier, the terminal goes on the eligible list, provided it is defined, of a type supported by BMS, and eligible to receive routed messages. If BMS cannot find the terminal definition, it sets the “entry skipped” and “invalid terminal identifier” bits (X'C0') in the status flag of the route list entry; if the terminal exists but is not supported by BMS or is not allowed to receive any routed messages, the “entry skipped” and “terminal not supported under BMS” bits get set (X'A0').

  Note: The eligibility of a terminal to receive routed messages is governed by the ROUTEDMSGS option in the terminal definition. Three values are possible: a terminal can be allowed to receive all routed messages, only messages routed to it by terminal or operator name, or no routed messages at all. If you specified OPCLASS as well as a route list, BMS checks whether an operator belonging to one of the classes you listed is signed on at the terminal. If not, BMS sets the “operator not signed on” bit (X'10') in the status flag for the entry to inform you, but includes the terminal anyway. There are no operator restrictions associated with the list entry, even when you specify operator classes.
- If the entry contains both a terminal and an operator identifier, the terminal identifier is checked in the same way as it is without an operator identifier, and the same errors can occur. If the terminal passes these tests, it goes on the eligible list. However, the entry is marked such that the message can be delivered only when the operator named is signed on at the same terminal.

  If this operator is not signed on to the terminal at the time of the ROUTE command, BMS notifies you by turning on the “operator not signed on” bit (X'10') in the status flag, but the terminal goes on the delivery list regardless of sign-on status. (OPCLASS is ignored entirely when an operator identifier is present.)
- If the entry contains only an operator identifier, BMS searches the terminal definitions until it finds one where the operator is signed on. (The operator might be signed on at additional terminals, but BMS ignores these.) If this terminal is of a type not supported by BMS, or if the terminal cannot receive routed messages, BMS sets the “entry skipped” and “operator signed on at unsupported terminal” bits (X'88') in the status flag. It also enters the terminal identifier in your route list. If the terminal is suitable, BMS treats the entry as if you had specified both that terminal and operator identifier, as previously described.

  If the operator is not signed on anywhere, BMS sets the “entry skipped” and “operator not signed on” bits (X'90') in the status flag.

Route list format

BMS requires a fixed format for route lists. Each entry in the list is 16 bytes long.

Table 50. Standard route list entry format

Bytes	Contents
0-3	Terminal or logical unit identifier (four characters, including trailing blanks), or blanks
4,5	LDC mnemonic (two characters) for logical units with LDC support, or blanks
6-8	Operator identifier, or blanks
9	Status flag for the route entry
10-15	Reserved; must contain blanks

Either a terminal or an operator identifier must be present in each entry. A Logical Device Component(LDC) may accompany either; see LDCs and routing in "Logical device components" on page 433 for more information about LDCs.

The entries in the route list normally follow one another in sequence. However, they do not all have to be adjacent. If you have a discontinuity in your list, you end each group of successive entries except the last group with an 8-byte chain entry that points to the first entry in the next group. This entry looks like this:

Table 51. Route list chain entry format

Bytes	Contents
0,1	-2 in binary halfword format (X'FFFE')
2,3	Reserved
4-7	Address of the first entry in the next group of contiguous entries

The end of the entire list is signalled by a 2-byte entry containing a halfword value of -1 (X'FFFF').

Your list may consist of as many groups as you want. There is an upper limit on the total number of destinations, but it depends on many variables; if you exceed it, BMS abends your task with abend code ABMC.

On return from a ROUTE command, BMS raises condition codes to signal errors in your list:

RTESOME
: means that at least one of the entries in your route list could not be used and was skipped. The default action is to continue the routing operation, using the destinations that were processed successfully.

RTEFAIL
: means that none of the destinations in your list could be used, and therefore no routing environment has been set up. The default action is to return control to your task. You should test for this condition, consequently, because with no routing environment, a subsequent BMS SEND command goes to the principal facility, which is probably not your intention.

In addition to the general information reflected by RTESOME and RTEFAIL, BMS tells you what it did with each entry in your list by setting the status flag (byte 9). A null value (X'00') means that the entry was entirely correct. The high-order bit tells you whether the entry was used or skipped, and the other bits tell you exactly what happened. Here are the meanings of each bit being on:

ENTRY SKIPPED (X'80')
: The entry was not used. When this bit is on, another bit is also on to indicate the reason.

INVALID TERMINAL IDENTIFIER (X'40')
: There is no terminal definition for the terminal named in the entry. The entry is skipped.

TERMINAL NOT SUPPORTED UNDER BMS (X'20')
The terminal named in the route list entry is of a type not supported by BMS, or it is restricted from receiving routed messages. The entry is skipped.

OPERATOR NOT SIGNED ON (X'10')
The operator named in the entry is not signed on. Any of these conditions causes this flag to be set:

- Both an operator identifier and a terminal identifier were specified, and the operator was not signed on at the terminal. The entry is not skipped.
- An operator identifier was specified without a terminal identifier, and the operator was not signed on at any terminal. The entry is skipped.
- OPCLASS was specified on the ROUTE command, a terminal identifier was specified in the route list entry, and the operator signed on at the terminal did not have any of the specified operator classes. The entry is not skipped.

OPERATOR SIGNED ON AT UNSUPPORTED TERMINAL (X'08')
Only an operator identifier was specified in the route list entry, and that operator was signed on at a terminal not supported by BMS or not eligible to receive routed messages. The entry is skipped. The name of the terminal is returned in the terminal identifier field of the entry.

INVALID LDC MNEMONIC (X'04')
Either of these conditions causes this flag to be set:

- The LDC mnemonic specified in the route list is not defined for this terminal. That is, the terminal supports LDCs but it has no LDC list, or its LDC list is extended but does not contain this entry.
- The device type for this LDC entry is different from that of the first entry in the route list with an LDC (only one LDC device type is allowed, as explained in LDCs and routing in "Logical device components" on page 433).

The entry is skipped.

Note: CICS provides source code which defines a standard route list entry and the values you need to test status flag bit combinations. You can insert this code into your program with a COPY or INCLUDE of the member DFHURLDS, in the same way you can include the BMS attention identifier or attribute byte definitions.

Message delivery
A message can be delivered some time after the **ROUTE** command is issued, depending on the scheduling options in the command (INTERVAL, TIME, AFTER and AT). You can request delivery immediately, after an interval of time has elapsed, or at a particular time of day.

When the appointed time arrives, BMS attempts to deliver the message to every terminal on the eligible terminal list. All the following conditions must be met for the message to be delivered to any particular terminal:

- The terminal must be defined as a type supported by BMS, and the same type as when the ROUTE command was processed. Where there is a long delay between creation and delivery of a message, it is possible for the terminal defined with a particular TERMID to change characteristics or disappear, especially in an autoinstall environment. A 3270 terminal need not have exactly the same extended attributes that it had at the time the ROUTE command was issued, because BMS removes unsupported attributes from the data stream at the time of delivery.
- The terminal must be in service and available (that is, there cannot be a task running with the terminal as its principal facility).
- The terminal must be eligible for automatic transaction initiation, or the terminal operator must request delivery of the message with the CSPG transaction.

  Note: If several messages accumulate for delivery to a particular terminal, there is no guarantee that the operator will view them in any particular order. In fact, the CSPG transaction allows the operator to control delivery order in some situations. If a specific sequence of pages is required, you must send them as one message.

- If the delivery list entry restricts delivery to a particular operator or to operators in certain classes, the operator signed on at the terminal must qualify. (See “Message destinations” on page 419 for the OPCLASS and LIST specifications that produce these restrictions.)
- The purge delay must not have expired, as explained in the next section.

Undeliverable messages
If BMS cannot deliver a message to an eligible terminal, it continues to try periodically.

Delivery is retried until one of the following conditions occurs:

- A change in terminal status allows the message to be sent.
- The message is deleted by the destination terminal operator.
- The purge delay elapses.

The purge delay is the period of time allowed for delivery of a message once it is scheduled for delivery. After this interval elapses, the message is discarded. The purge delay is a system-wide value, set by the PRGDLY option in the system initialization table. Its use is optional; if the systems programmer sets PRGDLY to zero, messages are kept indefinitely.

When BMS purges a message in this fashion, it sends an error message to the terminal you specify in ERRTERM. (If you use ERRTERM without a specific terminal name, it sends the message to the principal facility of the task that originally created the message. If you omit ERRTERM altogether, no message is sent.)

Recoverable messages

Between creation and delivery of a routed message with a disposition of PAGING, BMS stores the message in CICS temporary storage, just as it does in the case of an ordinary PAGING message. Consequently, you can make your routed messages recoverable by your choice of the REQID option value, just as in the case of a nonrouted message.

If you are routing to more than one type of terminal, BMS builds a separate logical message for each type, with the appropriate device-dependent data stream, and uses a separate temporary storage queue for each type.

Note: For terminal destinations that have the alternate screen size feature, where two message formats are possible, BMS chooses the default size if the profile under which the task creating the message specifies default size, and alternate size if the profile specifies alternate size.

All of the logical messages use the same REQID value, however, so that you can still choose whether they are recoverable or not.

BMS also uses temporary storage to store the list of terminals eligible to receive your message and to keep track of whether delivery has occurred. When all of the eligible terminals of a particular type have received a message, BMS deletes the associated logical message. When all of the destinations have received delivery, or the purge delay expires, BMS erases all of the information for the message, reporting the number of undeliverable messages by destination to the main terminal operator message queue.

Message identification
You can assign a title to your routed message if you want. The title is not part of the message itself, but is included with the other information that BMS maintains about your message in CICS temporary storage.

Titles are helpful in situations where a number of messages might accumulate for an operator or a terminal, because they allow the operator to control the order in which they are displayed. (See the “query” option of the CSPG command in CSPG - page retrieval .)

To assign a title, use the TITLE option of the ROUTE command to point to a data area that consists of the title preceded by a halfword binary length field. The length includes the 2 byte length field and has a maximum of 64, so the title itself can be up to 62 characters long. For example:

```
 01 MSG-TITLE.
02 TITLE-LENGTH PIC S9(4) COMP VALUE +19.
02 TITLE-TEXT PIC X(17) VALUE 'MONTHLY INVENTORY'.
...
EXEC CICS ROUTE TITLE(MSG-TITLE)....
```

Figure 119. Assigning a title

Programming considerations with routing

You build a routed message in a similar way to the way you build a non-routed message. However, some differences exist, because BMS builds a separate logical message for each terminal type among your destinations.

Routing and page overflow

Because different types of terminals have different page capacities, page overflow may occur at different times for different types. If you are using SEND MAP commands and intercepting overflows, your program gets control when overflow occurs for each page of each logical message that BMS is creating in response to your ROUTE.

If you want to number your pages or do page-dependent processing at overflow time, you may need to keep track of information for each terminal type separately. Data areas kept for this purpose are called **overflow control areas** . You can tell how many such areas you need (that is, how many different terminal types appeared in your ROUTE command) by issuing an ASSIGN command with the DESTCOUNT option after your ROUTE and before any BMS command that could cause overflow. Issued at this time, ASSIGN DESTCOUNT returns a count of the logical messages that BMS builds.

When overflow occurs, you can use the same command to determine for which logical message overflow occurred. At this time ASSIGN DESTCOUNT returns the relative number of that message among those BMS is building for this ROUTE command. If you are using overflow control areas, this number tells you which one to use. If you use ASSIGN PAGENUM at this time, BMS returns the number of the page that overflowed as well.

To handle the complication of different overflow points for different terminal types, the processing you need to do on overflow in a routing environment is:

- Determine which logical message overflowed with ASSIGN DESTCOUNT (unless you are doing very simple overflow processing).
- Send your trailer maps for the current page, followed by headers for the next page, as you do in a non-routing environment (see “Page breaks: BMS overflow processing” on page 410). While the OVERFLOW condition is in force, these SEND MAP commands apply only to the logical message that overflowed (you would not want them in a logical message where you were mid-page, and BMS makes no assumptions about different terminal types that happen to have the same page capacity).
- Reissue the command that caused the overflow, as you do in a non-routing environment. After you do, however, you must retest for overflow and repeat the whole process, until overflow does not occur. This procedure ensures that you get the trailers and headers and the map that caused the overflow into each of the logical messages that you are building.

Routing with SET

When you specify a disposition of SET in a routing environment, no messages are sent to the destinations in your route list, because the pages are returned to your program as they are completed.

However, the ROUTE command is processed in the usual way to determine these destinations and the terminal types among them. BMS builds a separate logical message for each type, as usual, and returns a page to the program each time one is completed for any of the terminal types. BMS raises the OVERFLOW and RETPAGE conditions as it does with a disposition of PAGING. Consequently, ROUTING with SET allows you to format messages for terminal types other than that of your principal facility.

Interleaving a conversation with message routing
While you are building a message to be routed, you can use BMS SEND commands as well as RECEIVE MAP and terminal control commands to converse with your principal facility.

Such SEND commands must have a disposition option of TERMINAL rather than PAGING or SET, and must not specify ACCUM. The associated inputs and outputs are processed directly and do not interfere with your logical message, even if your terminal is one of the destinations of the message.

Without routing, you cannot use BMS SENDs, as noted in "Building logical messages" on page 406.

The MAPPINGDEV facility

Minimum BMS function assumes that the principal facility of your task is the mapping device that performs input and output mapping operations for the features and status that is defined in the TCTTE (Terminal Control Table entry).

The principal facility for transactions using BMS function should have a device type supported by BMS. However, the MAPPINGDEV facility is an extension of minimum BMS that allows you to perform mapping operations for a device that is not the principal facility. When the MAPPINGDEV request completes, the mapped data is returned to the application. BMS does not have any communication with the MAPPINGDEV device.

You can specify the MAPPINGDEV option on the RECEIVE MAP command and the SEND MAP command, but not on any other BMS command.

The TERMID specified in the MAPPINGDEV option must represent a device in the 3270 family supported by BMS. If the device is partitioned, it is assumed to be in base state. Outboard formatting is ignored.

Data is mapped in the same way as for minimum BMS, and there is no need to change mapset definitions or to regenerate mapsets.

SEND MAP with the MAPPINGDEV option

Your SEND MAP commands that have the MAPPINGDEV option must also specify the SET option. (The SET option provides BMS with a pointer that sets the address of the storage area that contains the mapped output datastream.)

If you have storage protection active, the data is returned in storage in the key specified in the TASKDATAKEY option of the transaction definition. The storage is located above or below the line depending on which TASKDATALOC option of the transaction definition you have specified.

The storage area is in task-related user storage but in the format of a TIOA (Terminal Input/Output Area). The application can reference the storage area using the DFHTIOA copybook. The TIOATDL field, at offset 8, contains the length of the datastream that starts at TIOADBA, at offset 12, in the storage area. The length value placed in TIOATDL does not include the length of the 4 byte page control area, which contains information such as the extended attributes that have been used in the datastream and can be referenced using the DFHPGADS copybook.

The storage area typically has a length greater than the datastream because the storage area is allocated before the exact length of the output datastream is determined. This storage area is in a form that can be used in a SEND TEXT MAPPED command.

If you are familiar with using the SET option without the MAPPINGDEV option, you know that the datastream is returned to the application indirectly by a list of pages. However, when MAPPINGDEV is specified, a direct pointer to the storage area containing the datastream is returned to your application.

When the SEND MAP MAPPINGDEV command completes its processing, the storage area is under the control of the application and remains allocated until the end of the transaction unless your application releases it by using a FREEMAIN request. It is advisable to release these storage areas, by using a FREEMAIN request, for long-running transactions, but CICS frees these areas when the task terminates.

RECEIVE MAP with the MAPPINGDEV option

You must specify the FROM option when using the MAPPINGDEV option on the RECEIVE MAP command. BMS needs the FROM option to supply a formatted 3270 input datastream that is consistent with the datastream returned via a Terminal Control RECEIVE command (that is, a normal input 3270 datastream). The only difference is that it does not start with an AID and input cursor address because this information is removed from the input datastream by terminal control, but there are options on the RECEIVE MAP command that allow you to specify an AID value and input cursor position when the MAPPINGDEV option is specified. If the datastream contains an AID and input cursor address, they are ignored by BMS.

When neither option is specified, BMS assumes that the input data operation was terminated with the ENTER key, and returns the appropriate AID value to the application from the EIBAID field. BMS also assumes that the input cursor was positioned at the home address and returns a value of zero to the application from the EIBCPOSN field.

The new AID option of the RECEIVE MAP command allows your application to specify an AID value which, if specified, overrides the default value of ENTER. Whether provided by the application, or defaulted by BMS, the AID value that you established causes control to be passed, when applicable, to the routine registered by a previous HANDLE AID request issued by the application.

The new CURSOR option of the RECEIVE MAP command allows your application to specify an input cursor position which, if specified, overrides the default value of zero. Whether provided by the application, or defaulted by BMS, the input cursor value is used in cursor location processing when you define the map with CURSLOC=YES.

As with the minimum BMS RECEIVE MAP command, the mapped data is returned to your application by the INTO or SET option. If neither option is specified, the CICS translator attempts to apply a default INTO option by appending the character 'I' to the map name.

When you use the SET option with the MAPPINGDEV option, it must provide a pointer variable that BMS sets with the address of the storage area containing the mapped input datastream. The data is returned in task-related user storage. If storage protection is active, the data is returned in storage in the key specified by the TASKDATAKEY option of the transaction definition. The storage is located above or below the line depending on the TASKDATALOC option of your transaction definition.

When the RECEIVE MAP MAPPINGDEV command completes its processing successfully, the storage area is returned by the SET option and is under the control of the application and remains allocated until the end of the transaction unless your application releases it by using a FREEMAIN request. It is advisable to release these storage areas, by using a FREEMAIN request, for long-running transactions, but CICS frees these areas when the task terminates.

Sample assembler MAPPINGDEV application

This example uses a modified version of the FILEA operator instruction sample to show you how to code the keywords associated with the MAPPINGDEV facility.

Figure 120 on page 427 is a modification of the FILEA operator instruction sample program, and uses the same mapset named DFH$AGA.

This application is only intended to demonstrate how to code the keywords associated with the MAPPINGDEV facility, and as a means of testing this function. It is not offered as a recommended design for applications that use the MAPPINGDEV facility.

```
DFH$AMNX CSECT
*
DFHREGS
DFHEISTG DSECT
OUTAREA DS 0CL512
DS CL8
OUTLEN DS H
DS H
OUTDATA DS CL500
INLEN DS H
INAREA DS CL256
PROOF DS CL60
COPY DFH$AGA
COPY DFHBMSCA
DFH$AMNU CSECT
EXEC CICS HANDLE AID PF3(PF3_ROUTINE)
*
XC DFH$AGAS(DFH$AGAL),DFH$AGAS
MVC MSGO(L'APPLMSG),APPLMSG
EXEC CICS SEND MAP('DFH$$AGA') FROM(DFH$AGAO) ERASE
MAPPINGDEV(EIBTRMID) SET(R6)
MVC OUTAREA(256),0(R6)
MVC OUTAREA+256(256),256(R6)
EXEC CICS SEND TEXT MAPPED FROM(OUTDATA) LENGTH(OUTLEN)
*
EXEC CICS RECEIVE INTO(INAREA) LENGTH(INLEN)
MAXLENGTH(MAXLEN)
*
EXEC CICS RECEIVE MAP('DFH$AGA') SET(R7) LENGTH(INLEN)
MAPPINGDEV(EIBTRMID) FROM(INAREA)
CURSOR(820) AID(=C'3')
*
XC PROOF,PROOF
MVC PROOF(25),=C'You just keyed in number '
MVC PROOF+25(6),KEYI-DFH$$AGAI(R7)
```

```
FINISH DS 0H
EXEC CICS SEND TEXT FROM(PROOF) LENGTH(60) ERASE FREEKB
TM MSGF-DFH$AGAI(R7),X'02'
BNO RETURN
XC PROOF,PROOF
MVC PROOF(33),=C'Input cursor located in MSG field'
EXEC CICS SEND TEXT FROM(PROOF) LENGTH(60) ERASE FREEKB
*
* THE RETURN COMMAND ENDS THE PROGRAM.
*
RETURN DS 0H
EXEC CICS RETURN
*
PF3_ROUTINE DS 0H
XC PROOF,PROOF
MVC PROOF(30),=C'RECEIVE MAP specified AID(PF3)'
B FINISH
MAXLEN DC H'256'
APPLMSG DC C'This is a MAPPINGDEV application'
END
```

Figure 120. ASM example of a MAPPINGDEV application

Partition support

Some IBM displays allow you to divide the screen into areas which you can write to and read from separately, as if they were independent screens. The areas are called partitions, and features of BMS that allow you to take advantage of the special hardware are collectively called *partition support*.

Standard BMS is required for partitions.

The IBM 3290 display, which is a member of the 3270 family, and the IBM 8775 are the primary examples of devices that support partitioning. You should consult the device manuals (*IBM 3290 Information Display Panel Description and Reference* for the 3290 and *IBM 8775 Display Terminal Component Description*) to understand the full capabilities of a partitioned device, but the essential features are these:

- You can divide the physical screen into any arrangement of one to eight non-overlapping rectangular areas. The areas are independent from one other, in the sense that the operator can clear them

separately, the state of the keyboard (locked or unlocked) is maintained separately for each, and you write to and read from them one at a time.

- Only one partition is **active** at any given time. This is the one containing the cursor. The operator is restricted to keying into this partition, and the cursor wraps at partition boundaries. When a key that transmits data is depressed (the ENTER key or one of the program function keys), data is transmitted only from the active partition.
- The operator can change the active partition at any time by using the "jump" key; your program can also, as explained in "Determining the active partition" on page 431.
- BMS also writes to only one partition on a given SEND, but you can issue multiple SENDs and you do not have to write to the active partition.
- The partition configuration is sent to the device as a data stream, so that you can change the partitions for each new task or even within a task. The BMS construct that defines the partitions is called a **partition set** and is described in "Partition definition" on page 429.
- You also can use the terminal in **base state** (without partitions) and you can switch from partitioned to base state with the same command that you use to change partition arrangements.
- When you specify how to partition the screen area, you also divide up the hardware buffer space from which the screen is driven. In partitioned devices, the capacity of the buffer is generally greater than that of the screen, so that some partitions can be assigned extra buffer space. The screen area allocated to a partition is called its **viewport** and the buffer storage is called its **presentation space**.

  BMS uses the presentation space as its page size for the partition, so that you can send as much data as fits there, even though not all of it can be on display at once. Keys on the device allow the operator to scroll the viewport of the partition vertically to view the entire presentation space. Scrolling occurs without any intervention from the host.
- Some partitioned devices allow you to choose among character sets of different sizes. We talk about this in "3290 character size" on page 430.

In spite of the independence of the partitions, the display is still a single terminal to CICS . You cannot have more than one task at a time with the terminal as its principal facility, although you can use the screen space cooperatively among several pseudoconversational transaction sequences if they use the same partition set (see "Terminal sharing" on page 433).

Note: The 3290 can be configured internally so that it behaves as more than one logical unit (to CICS or any other system); this definition is separate from the partitioning that may occur at any one of those logical terminals.

Uses for partitioned screens

Partitioned screens are particularly useful in certain types of application.

Scrolling

For transactions that produce more output than fits on a single screen, scrolling is an alternative to BMS terminal paging.

For example, you can define a partition set that consists of just one partition, where the viewport is the whole screen and the presentation space is the entire buffer. You can write to the entire buffer as a single page, and the operator can scroll through the data using the terminal facilities. Response time is to scrolling requests is very short, because there is no interaction with the host. You are limited to the capacity of the buffer, of course.

You may also want to scroll just part of the screen and use some partitions for fixed data.

Data entry

Another good use for a partitioned screen is "heads down" data entry, where the operator's productivity depends on how fast the application can process an input and reopen the keyboard for the next.

With a partitioned screen, you can divide the screen into two identical entry screens. The operator fills one, presses Enter, and then fills the second one while the data entry transaction is processing the first

input. If the input is good, the program erases it in preparation for the next entry; if not, there is still an opportunity for the operator to make corrections without losing subsequent work.

Lookaside
In many online operations, the operator sometimes needs to execute a second transaction in order to finish one in progress. Order entry is an example, where the operator may have to look up codes or prices to complete an entry.

Many inquiries are similar. The initial inquiry brings back a summary list of hits. The operator selects one and asks for further detail, then may need to select another for detail, and so on. In such cases, a partitioned screen allows the operator to do the second task while keeping the output of the first, which is needed later, on the screen.

"Help" text is still another example of "lookaside". If you allocate one partition of the screen to this text, the operator can get the required tutorial information without losing the main screen.

Data comparison
Applications in which the operator needs to compare two or more sets of data simultaneously are also excellent candidates for a partitioned screen. Partitioning allows a side-by-side comparison, and the scrolling feature makes it possible to compare relatively large documents or records.

Error messages
You can enhance usability if you partition a screen and allocate one area to error messages and other explanatory text. If you do this, the operator always knows where to look for messages, and the main screen areas are never overwritten with such information.

CICS sends its own messages to a such a partition if you designate one in your partition set, as we explain in "Partition definition" on page 429.

Partition definition
Each partitioning of a screen is defined by a partition set, which is a collection of screen areas (partitions) intended for display together on a screen.

Y ou define a partition set with assembler macros, just as you do map sets. There are two of them: DFHPSD and DFHPDI.

The partition set definition begins with a DFHPSD (partition set definition) macro, which defines:

- The name of the partition set
- Screen size (BMS makes sure that the partition viewports do not exceed the total space available)
- Default character cell size (we talk about cell size in "3290 character size" on page 430)
- The partition set suffix, used to associate the partition set with a particular screen size (see "Establishing partitioning" on page 430)

After the initial DFHPSD macro, you define each partition (screen area) with a DFHPDI macro. DFHPDI specifies:

- The identifier of the partition within the partition set.
- Where the partition is located on the screen.
- Its viewport size (in lines and columns).
- The presentation space associated with the viewport (that is, the amount of buffer space allocated), also in lines and columns. Because scrolling is strictly vertical, BMS requires that the width of the presentation space match the width of the viewport.
- The character size to be used.
- The map set suffix associated with the partition, used to select the map set appropriate for the partition size.
- Whether the partition may receive CICS error messages (BMS sends certain error messages that it generates to a partition so designated, if there is one).

You end the partition set with a second DFHPSD macro, containing only the option TYPE=FINAL.

Because these are assembler macros, you need to follow assembler format rules in creating them. See "Writing BMS macros" on page 374 if you are not familiar with assembler language. After you do, you need to assemble and link-edit your partition set. The resulting load module can be placed in the same library as your map sets, or in a separate library if your installation prefers. Your systems staff also need to define each partition set to the system with a PARTITION definition.

3290 character size
The 3290 hardware allows you to use up to eight different character sets, of different sizes. Two sets come with the hardware; the others can be loaded with a terminal control SEND command.

Each character occupies a rectangular **cell** on the screen. Cell size determines how many lines and columns fit on the screen, or in a particular partition of the screen, because you can specify cell size by partition. Cells are measured in pels (picture elements), both vertically and horizontally. The smallest cell allowed is 12 vertical pels by 6 horizontal pels. The 3290 screen is 750 pels high and 960 pels wide. Using the minimum cell size, therefore, you can fit 62 characters vertically (that is, have 62 lines), and 160 characters horizontally (for 160 columns). (The 3290 always selects the character set that best fits your cell size, and places the character at the top left corner of the cell.)

Partition sizes are expressed in lines and columns, based on the cell size you specify for the partition, which is expressed in pels. (The name of the option is CHARSZE, but it is really cell size.) To make sure your partitions fit on the screen, you need to work out your allocation in pels, although BMS tells you when you assemble if your partitions overlap or does not fit on the screen. The partition height is the product of the number of rows in the partition and the vertical CHARSZE dimension; the partition width is the product of the number of columns and the horizontal CHARSZE value.

If you do not specify a CHARSZE size in your DFHPDI partition definition, BMS uses the default given in the DFHPSD partition set definition. If DFHPSD does not specify CHARSZE either, BMS uses the default established for the terminal when it was installed. If you specify cell size for some but not all partitions, you must specify a default for the partition set too, so that you do not mix your choices with the installation default.

Establishing partitioning
You can tell BMS which partition set to load for a particular transaction by naming it in the PARTITIONSET option of the TRANSACTION definition.

If you do this, and the named partition set is not already loaded at the terminal, BMS adds the partition definitions to your data on the first BMS SEND in the task.

You can also direct BMS not to change the partitions from their current state (PARTITIONSET=KEEP in the TRANSACTION definition) or indicate that you load the partitions yourself (PARTITIONSET=OWN). If you do not specify any PARTITIONSET value, BMS sets the terminal to base state (no partitions) at the time it initiates the transaction.

Whatever the PARTITIONSET value associated with the transaction, a task can establish new partitions at almost any time with a SEND PARTNSET command, except that you cannot issue the command while you are building a logical message.

SEND PARTNSET does not send anything to the terminal immediately. Instead, BMS remembers to send the partition information along with the next BMS command that sends data or control information, just as it sends a partition set named in the PARTITIONSET option of the TRANSACTION definition on the first BMS SEND. Consequently, you must issue a SEND MAP, SEND TEXT or SEND CONTROL command before you issue a RECEIVE or RECEIVE MAP that depends on the new partitions.

Note: You can get an unexpected change of partitions in the following situation. If CICS needs to send an error message to your terminal, and the current partition set does not include an error partition, CICS returns the terminal to base state, clear the screen, and write the message. For this reason, it is a good idea to designate one partition as eligible for error messages in every partition set.

When BMS loads a partition set, it suffixes the name requested with the letter that represents your terminal type if device-dependent support is in effect, in order to load the one appropriate to your terminal. It takes suffix from the ALTSUFFIX option value of the TYPETERM definition associated with your

terminal. Partition set suffixing is analogous to map set suffixing, and the same sequence of steps is taken if there is no partition set with the right suffix (see “Device-dependent maps” on page 382).

Partition options for BMS SEND commands

W hen you write to a partitioned screen, you write to only one partition, and the effects of your command are limited to that partition.

ERASE and ERASEAUP clear only within the partition, and FREEKB unlocks the keyboard only when the partition becomes active.

You can specify the partition to which you are sending with either the PARTN option in your map definition or with the OUTPARTN option on your SEND MAP. OUTPARTN overrides PARTN. If you don't specify either, BMS chooses the first partition in the set.

The use of partitions affects the suffixing of map set names that we described in “Device-dependent maps” on page 382 . The map set suffix is taken from the MAPSFX value for the partition instead of being determined as described in that section.

Determining the active partition

When you send to a partition, you can move the cursor to that partition or another one. A value of ACTIVATE in the PARTN option of the map definition puts the cursor in the partition to which you are writing.

If you specify ACTPARTN on your BMS SEND command, you can name any partition (not necessarily the one to which you are writing), and you override the ACTIVATE specification. Both ACTIVATE and ACTPARTN unlock the keyboard for the active partition, as well as placing the cursor there. If neither is present, the cursor does not move and the keyboard is not unlocked.

Although you can make a partition active by placing the cursor there when you send, you do not have the last word on this subject, because the operator can use the jump key on the terminal to move the cursor to another partition. This can complicate receiving data back from the terminal, but BMS provides help, as we are about to explain.

Partition options for BMS RECEIVE commands

When you issue a RECEIVE MAP command, you can tell BMS from which partition you expect data (that is, which partition you expect to be active) with either the PARTN option in the map definition or with the INPARTN option on your RECEIVE MAP.

INPARTN overrides PARTN. If you do, and the operator transmits from a different partition than the one you named, BMS repositions the cursor in the partition you named, unlocks the keyboard and repeats the RECEIVE command. It also sends a message to the error partition (the one with ATTRB=ERROR) asking the operator to use the right partition. (No message is sent if there is no error partition.) The input from the wrong partition is discarded, although it is not lost, because it can be reread later. BMS does this up to three times; if the operator persists for a fourth round, BMS raises the PARTNFAIL condition.

You do not have to specify an input partition; sometimes there is only one that allows input, and sometimes the same map applies to all. If you issue RECEIVE MAP without INPARTN and there is no PARTN option in the map, BMS accepts data from any partition and map it with the map named in the command. You also can determine the partition afterward, if you need to, with an ASSIGN command containing the INPARTN option.

INPARTN is not set until after the first BMS operation, however, and so if you need to know which partition is sending to select the right map, you need another approach. In this situation, you can issue a RECEIVE PARTN command, which reads data unmapped and tells you which partition sent it. Then you issue a RECEIVE MAP command using the map that matches the partition with the FROM option, using the map that matches the partition. RECEIVE MAP with FROM maps data already read, as explained in “Formatting other input” on page 405.

ASSIGN options for partitions
In addition to the INPARTN option , there are three other ASSIGN options to help you in programming for a partitioned terminal.

The PARTNS option tells you whether the terminal associated with your task supports partitions, and the PARTNSET option returns the name of the current partition set (blanks if none has been established). The fourth ASSIGN option, PARTNPAGE applies only to logical messages, which we talk about in "Partitions and logical messages" on page 432.

Partitions and logical messages

When you build a BMS logical message for a terminal for which partitions have been established, you can direct the pages of the message to multiple partitions.

You can even send text output to some partitions and mapped output to others, provided you do not mix them in the same partition. (This is an exception to the normal rule against mixing text and mapped output in a logical message.)

When the output is displayed, the first page for each partition is displayed initially. The pages are numbered by partition, and CSPG commands that the operator enters into a particular partition apply only to that partition, with the exception of the page purge command. The purge command deletes the entire logical message from all partitions.

On each BMS SEND that contributes to the message, you specify the partition to which the output goes. If you are not using ACCUM, BMS builds a page for that partition. If you are using ACCUM, BMS puts the output on the current page for that partition. Page overflows therefore occur by partition. If you are intercepting overflows and are not sure in which partition the overflow occurred, you can use the PARTNPAGE option of the ASSIGN command to find out.

Note: Because BMS uses both the page size and partition identifiers in building a logical message, you cannot change the partitions mid-message.

The bookkeeping required to handle page overflow when you are distributing pages among partitions is analogous to that required in a routing environment (see "Routing and page overflow" on page 424). In particular, you need to ensure that you finish overflow processing for one partition before doing anything that might cause overflow in another. Failure to do so can cause program loops as well as incorrect output.

Partitions and routing
You cannot route a logical message written to multiple partitions. BMS ignores the OUTPARTN and ACTPARTN options on BMS SEND commands in a routing environment.

You can route an ordinary message to a terminal that supports partitions, but BMS builds the message and the CSPG transaction displays it using the terminal in base (unpartitioned) state.

You also cannot use partitions and logical device codes together (LDCs are described in "Logical device components" on page 433). In addition, you cannot use partitions in combination with GDDM , although you can use partitions with outboard formats (see "Outboard formatting" on page 437).

Attention identifiers and exception conditions

Partitioned terminals have a CLEAR PARTITION key that clears the active partition in the same way that the CLEAR key clears the whole screen . You may need to check for this additional attention identifier in your program logic.

The CLEAR PARTITION AID value is included in DFHAID (see "Using the attention identifier" on page 401).

There are also some new exception conditions associated with partitions, and new ways to get some of the old ones. The new ones include INVPARTN (naming a partition that does not exist in the partition set), INVPARTNSET (naming a module that is not a partition set), and PARTNFAIL (receiving from a partition other than the one the operator transmitted from). They are all described in the BMS macros manual with the commands to which they apply.

Terminal sharing

You can share a terminal among several processes by assigning each a separate partition. You cannot have more than one task in progress at once at a terminal, but you can interleave the component tasks of several pseudoconversational transaction sequences at a partitioned terminal.

To take a simple example, suppose you decide to improve response time for an existing pseudoconversational data entry transaction by letting the operator enter data in two partitions (see “Data entry” on page 428). You could modify the application to work on two records at once, or you could modify it to send to the same partition from which it got its input. Then you could run it independently from each partition.

You can establish the partitions with the PARTITIONSET option in the TRANSACTION definition (all the transactions involved, if there are several in the sequence). As noted earlier, BMS does not reload the partitions as long as each transaction has the same PARTITIONSET value. Alternatively, you could establish the partitions with a preliminary transaction (for example, one that displayed the first entry screen in both partitions) and use a PARTITIONSET value of KEEP for the data entry transactions. Whenever you share a partitioned screen, whether among like transactions or different ones, you need to ensure that one does not the destroy the partition set required by another. Also, if two different CICS systems can share the same screen, they should name partition sets in common, so that BMS does not reload the partitions when it should not.

If the data entry transaction sequence uses the TRANSID option on the RETURN command to specify the next transaction identifier, you would need to make another small change, because the option applies to the whole terminal, not the partition. One solution would be to place the next transaction identifier in the first field on the screen (turning on the modified data tag in the field definition) and remove the TRANSID from the RETURN. CICS would then determine the next transaction from the input, as described in “How tasks are started” on page 73.

Support for special hardware

In addition to partitions, BMS supports other special hardware features.

BMS supports the following features:

- Logical device components
- 10/63 magnetic slot reader
- Field selection features: cursor select, light pen, trigger fields
- Outboard formatting

The magnetic slot reader and outboard formatting both require **standard** BMS. Support for the cursor select key, light pen and trigger fields is included in **minimum**.

Logical device components

Logical device components (LDCs) are another special hardware feature supported by BMS. Like partitions, LDCs require standard BMS.

A terminal that supports LDCs is one that consists of multiple functional components (logical devices) controlled through a single point (the logical unit). The components might be a printer, reader, keyboard and display, representing a remote work station, or they might be multiple like devices, such as word processing stations or passbook printers. The IBM 3601 logical unit, the 3770 batch logical unit, 3770, and 3790 batch data interchange logical units, and LU type 4 logical units all support logical device components.

Because the logical unit is a single entity to CICS , but consists of components that can be written and read independently, the CICS application programming interface for LDC terminals looks similar to that for partitioned terminals, each LDC corresponding to one partition in a partition set. There are many differences, of course, and you should consult the CICS manual that describes CICS support for your particular terminal type. The sections which follow describe the major differences that affect programming, which are:

- LDC definition

- SEND command options
- Logical messages
- Routing

Defining logical device components

The logical device components for a terminal are defined by a list called an LDC table. The TYPETERM component of the TERMINAL definition points to the table, which may be individual to the logical unit or shared by several logical units that have the same components.

The table itself is defined with DFHTCT TYPE=LDC (terminal control) macros. See TYPETERM resources and Terminal control table (TCT) for descriptions of both macros.

An LDC table contains the following information for each logical device component of the logical unit:

- A 2-character logical device identifier. These identifiers are usually standard abbreviations, such as CO for console and MS for a magnetic stripe encoder, but they need not be.
- A 1-character device code, indicating the device type (console, card reader, word processing station). Codes are assigned by CICS from the device type and other information provided in the macro.
- A BMS page size. BMS uses this size, rather than one associated with the logical unit, because different logical devices have different page sizes.
- A BMS page status (AUTOPAGE or NOAUTOPAGE); see "The AUTOPAGE option" on page 408.

Sending data to a logical device component

You direct BMS output to a specific logical device component of a terminal by naming it in the LDC option of your SEND MAP, SEND TEXT, or SEND CONTROL command or the LDC option of your mapset. A value in the command overrides one in the map set. If the LDC does not appear in either place, BMS uses a default that varies with the terminal type.

LDCs and logical messages

When you build a BMS logical message for your own terminal, you can distribute pages of the message among different logical device components in the same way that you can direct pages to a logical message to different partitions. BMS accumulates pages separately for each logical device component in the same way that it does for partitions (see "Partitions and logical messages" on page 432). You can include both text and mapped output in the message, provided you do not send both to one LDC. Page overflow occurs by LDC, and terminal operator paging commands operate on a logical device component basis.

When retrieving pages, the operator (or user code in the device controller) must indicate the LDC to which the request applies, because not all devices have keyboards. As in the case of partitions, a message purge request deletes the entire message, from all LDCs. See CSPG - page retrieval for more detail on page retrieval for logical devices.

If you are intercepting page overflows, you can tell which LDC overflowed by issuing an ASSIGN command with either the LDCMNEM or LDCNUM option. Both identify the device which overflowed, the first by its 2-character name and the second by the 1-byte numeric identifier. You can determine the page number for the overflowing device with ASSIGN PAGENUM, just as with a partitioned device.

There is one restriction associated with LDCs and page overflow that is unique to LDCs. After overflow occurs, you must send both a trailer map for the current page and a header for the next one to the LDC that overflowed. BMS raises the INVREQ (invalid request) condition if you fail to do this.

LDCs and routing

Routing is supported in an LDC environment, provided the message goes to the same component type for every destination that supports LDCs. You cannot route a multiple-LDC message.

You can supply the LDC value in several ways:

- If you use the LDC option on your ROUTE command, the value supplied overrides all other sources and is used for all eligible destinations to which LDCs apply.
- If you specify an LDC in a route list entry (and not in the ROUTE command), that value is used for the associated destination. (If you specify both and they do not agree, the ROUTE list value is used and the discrepancy is flagged in the status flag of the entry.)
- If you specify neither, the value is determined from terminal and system LDC tables in the same way as it is in a non-routing environment when you omit the LDC from the BMS SEND command. (The value on the SEND command is ignored when routing is in effect.)

10/63 magnetic slot reader

Some IBM display terminals support a magnetic slot reader (MSR), a device that reads data from small magnetic cards, as an optional feature. The MSR has indicator lights and an audible alarm to prompt operator actions. Some terminals control the MSR themselves, but others, such as the IBM 8775 and the IBM 3643, let you control the functions of the reader by program.

CICS provides an ASSIGN command option, MSR, that tells you whether the principal facility of your task has an MSR or not.

With BMS, you can control the state of such an MSR by using the MSR option of the BMS SEND commands. This option transmits four bytes of control data to the attached MSR, in addition to display data sent to the terminal. BMS provides a copybook, DFHMSRCA, containing most of the control sequences you might need. CICS Application development reference describes the supplied constants and explains the structure of the control data, so that you can expand the list if you need to.

The control sequence that you send to an MSR affects the next input from the device; hence it has no effect until a RECEIVE command is issued. Input from MSRs is placed in the device buffer and transmitted in the same way as keyboard input. If the MSR input causes transmission, you can detect this by looking at the attention identifier in EIBAID. A value of X'E6' indicates input from the MSR, and X'E7' signals input from the MSR extended (a second MSR that may be present). For information on how to format a screen for MSR input and other details on these devices, see IBM 3270 Data Stream Programmers Reference.

Field selection features

BMS supports several special hardware features that allow the operator to enter and transmit input by selecting a field on the screen.

The following features are supported:

- Trigger fields
- Cursor selectable fields
- Light pen detection

Trigger field support

Trigger fields are a special hardware feature of certain types of terminal, such as the 8775. A field defined as a trigger field causes the terminal to transmit its contents if the operator moves the cursor out of the field when it is primed.

The field gets primed when the operator moves the cursor into it and enters data or uses either the DELETE or ERASE EOF keys. It becomes unprimed after it causes transmission, or if the operator uses the ERASE INPUT key, or after a send to the terminal (if you are using partitions, the send must be to the partition that contains the trigger field to have this effect).

You define a field as a trigger field by setting the VALIDN extended attribute to a value of TRIGGER, either in the map or by program override.

Only the field itself is sent when a trigger field causes transmission; other fields are not sent, even if they have been modified. You can detect a transmission caused by a trigger field by checking the attention identifier, which has a value of X'7F'.

Terminals that support the validation feature buffer the keyboard, so that the operator can continue to enter data while the host is processing an earlier transmission. The program processing such inputs needs

to respond quickly, so that the operator does not exceed the buffer capacity or enter a lot of data before an earlier error is diagnosed.

The customary procedure is for the program receiving the input to check the contents of the trigger field immediately. If correct, the program unlocks the keyboard to let the operator continue (a BMS SEND command containing the FREEKB option does this). If the field is in error, you might want to discard the stored keystrokes, in addition to sending a diagnostic message. Any of the following actions does this:

- A BMS SEND command that contains ERASE, ERASEAUP, or ACTPARTN or that lacks FREEKB
- A BMS SEND directed to a partition other than the one containing the trigger field (where partitions are in use)
- A RECEIVE MAP, RECEIVE PARTITION, or terminal control RECEIVE command
- Ending the task

See IBM 3270 Data Stream Programmers Reference for more information about trigger fields.

Cursor and pen-detectable fields

BMS supports *detectable* fields, another special hardware feature available on some terminals. There are two hardware mechanisms for detectable fields: the “cursor select” key and the light pen.

A terminal has either the key or a pen, not both. Both work the same way and, as the key succeeded the pen, we talk about the key.

For a field to be detectable, it must have certain field attributes, and the first character of the data, known as the **designator character** , must contain one of five particular values. You can have other display data after the designator character if you want.

The bits in the field attributes byte that govern detectability also control brightness. High intensity (ATTRB=BRT) fields are detectable if the designator character is one of the detectable values. Normal intensity fields may or may not be detectable; you have to specify ATTRB=DET to make them so; nondisplay (ATTRB=DRK) fields cannot be detectable.

As usual, you can specify attributes and designator characters either in the map definition or by program override. However, DET has a special effect when it appears in an input-only map, as we explain in a moment.

Note that because high-intensity fields have, by definition, the correct field attributes for detectability, the terminal operator can make an *unprotected* high-intensity field detectable by keying a designator character into the first position of the field.

Selection fields

There are two types of detectable field, *selection* and *attention* fields; the type is governed by the *designator character*.

A selection field is defined by a designator character of either a question mark (?) or a greater-than sign (>). The convention is that (?) means the operator has not selected whatever the field represents, and (>) means he has. The hardware is designed around this convention, but it is not enforced, and you can use another if it suits. You can initialize the designator to either value and initialize the modified data tag off or on with either value.

Every time the operator presses the cursor select key when the cursor is in a selection field, the designator switches from one value to the other (? changes to > and > changes to ?). The MDT is turned *on* when the designator goes from ? to > and *off* when the designator goes from > to ?, regardless of its previous state. This allows the operator to change his mind about a field he has selected (by pressing cursor select under it again) and gives him ultimate control over the status of the MDT. The MDT governs whether the field is included when transmission occurs, as it does for other fields. No transmission occurs at this time, however; selection fields do not of themselves cause transmission; that is the purpose of attention fields.

Attention fields
Attention fields are defined by a designator character of blank, null, or ampersand.

A null in the data stream has the same effect as a blank in this function, but in BMS you should use a blank, because BMS does not transmit nulls in some circumstances, and because you cannot override the first position of a field with a null (see Where the values come from in “Building the output screen” on page 391). In contrast to a selection field, when the cursor select key is pressed with the cursor in an attention field, transmission occurs.

If the designator character is an ampersand, the effect of pressing the cursor select key is the same as pressing the ENTER key. However, if the designator is blank or null, what gets transmitted is the address of every field with the MDT on, the position of the cursor, and an attention identifier of X'7E' . The *contents* of these fields are not transmitted, as they are with the ENTER key (or a cursor select with an ampersand designator). In either case, the fields with the MDT bit on can be selection fields or normal fields which the operator changed or which were sent with the MDT on.

BMS input from detectable fields
After transmission caused by a cursor-select attention field with a blank or null designator, BMS tells you which fields were transmitted (that is, had the MDT on) by setting the first position of the corresponding input (I) subfield to X'FF' . The first position is otherwise set to X'00'.

You can tell which attention field caused transmission from this value if it was the only one transmitted, or from the position of the cursor otherwise.

If transmission is caused by a cursor-select attention field with an ampersand designator (or by the ENTER key or a function key), the I subfield contains the contents of the field if the MDT is on and the L subfield reflects its length, as typical, except if the DET attribute is specified for a field in the input map (that is MODE=IN or MODE=INOUT, DATA=FIELD). After a RECEIVE MAP naming such a map, this I subfield contains X'FF' with a length of 1 if the field is selected (that is, if the MDT was on), and a null (X'00') if not. BMS supplies no other input for the field, even if some was transmitted.

Consequently, if you need to receive data from a detectable field as well as knowing whether it was selected or not, you need to avoid the use of DET in an input map. You can use separate maps for output and input and specify the DET attribute only in the output map, or you can set the DET attribute in the datastream sent by the program rather than in the map. For high intensity fields you do not need to specify DET, because BRT implies DET. BMS will return the data for fields specified as BRT in the input map.

You also need to ensure that the data gets transmitted. When the cause of transmission is the ENTER key, a function key, or an attention field with an ampersand designator character, field data gets transmitted. It does not when the cause is an attention field with a blank or null designator.

See IBM 3270 Data Stream Programmers Reference for more information about detectable fields.

Outboard formatting
Outboard formatting is a technique for reducing the amount of line traffic between the host processor and an attached subsystem. The reduction is achieved by sending only variable data across the network. This data is combined with constant data, such as a physical map, by a program within the subsystem. The formatted data can then be displayed.

You can use outboard formatting with a 3650 Host Communication Logical Unit, an 8100 Series processor with DPPX and DPS Version 2, or a terminal attached through a 3174 control unit. Maps used by the 3650 must be redefined using the 3650 transformation definition language before they can be used. Maps to be used with the 8100 must be generated on the 8100 using either an SDF II utility or the interactive map definition component of the DPS Version 2.

If a program in the host processor sends a lot of mapped data to subsystems, you can reduce line traffic by telling BMS to transmit only the variable data in maps. The subsystem must then perform the mapping operation when it receives the data. BMS prefixes the variable data with information that identifies the subsystem map to be used to format the data.

Terminals that support outboard formatting have OBFORMAT(YES) in their TYPETERM definition. When a program issues a SEND MAP command for such a terminal, and the specified map definition contains

OBFMT=YES, BMS assumes that the subsystem is going to format the data and generates an appropriate data stream. If you send a map that has OBFMT=YES to a terminal that does not support outboard formatting, BMS ignores the OBFMT operand.

See "Using batch data interchange" on page 272 for more information about programming some of the devices that support outboard formatting.

BMS: design for performance

When building a formatted data stream with basic mapping support (BMS), you should bear in mind the factors described here.

Avoid turning on modified data tags (MDTs) unnecessarily

The MDT is the bit in the attribute byte that determines whether a field should be transmitted on a READ MODIFIED command (the command used by CICS for all but copy operations).

The MDT for a field is normally turned on by the 3270 hardware when the user enters data into a field. However, you can also turn the tag on when you send a map to the screen, either by specifying FSET in the map or by sending an override attribute byte that has the tag on. You should never set the tag on in this way for a field that is constant in the map, or for a field that has no label (and is not sent to the program that receives the map).

Also, you do not normally need to specify FSET for an ordinary input field. This is because, as already mentioned, the MDT is turned on automatically in any field in which the user enters data. This is then included in the next RECEIVE command. These tags remain on, no matter how many times the screen is sent, until explicitly turned ***off*** by the program (by the FRSET, ERASEAUP, or ERASE option, or by an override attribute with the tag off).

You can store information, between inputs, that the user did not enter on the screen. This is an intended reason for turning the MDT on by a program. However, this storage technique is appropriate only to small amounts of data, and is more suitable for local than for remote terminals, because of the transmission overhead involved. For example, this technique is particularly useful for storing default values for input fields. In some applications, the user must complete a screen in which some fields already contain default values. A user who does not want to change a default just skips that field. The program processing the input has to be informed what these defaults are. If they are always the same, they can be supplied as constants in the program. If they are variable, however, and depend on earlier inputs, you can save them on the screen by turning the MDT on with FSET in the map that writes the screen. The program reading the screen then receives the default value from a user who does not change the field and the new value from a user who does.

Note: The saved values are not returned to the screen if the CLEAR, PA1, PA2, or PA3 key is pressed.

Use FRSET to reduce inbound traffic

If you have a screen with many input fields, which you may have to read several times, you can reduce the length of the input data stream by specifying FRSET when you write back to the screen in preparation for the next read. FRSET turns off the MDTs, so that fields entered before that write are not present unless the user reenters them the next time. If you are dealing with a relatively full screen and a process where there may be a number of error cycles (or repeat transmissions for some other reason), this can be a substantial saving. However, because only ***changed*** fields are sent on subsequent reads, the program must save input from each cycle and merge the new data with the old. This is not necessary if you are not using FRSET, because the MDTs remain on, and all fields are sent regardless of when they were entered.

Do not send blank fields to the screen

Sending fields to the screen that consist entirely of blanks or that are filled out on the right by trailing blanks usually wastes line capacity. The only case where BMS requires you to do this is when you need to erase a field on the screen that currently contains data, or to replace it with data shorter than that currently on the screen, without changing the rest of the screen.

This is because, when BMS builds the data stream representing your map, it includes blanks (X'40') but omits nulls (X'00'). This makes the output data stream shorter. BMS omits any field whose first data character is null, regardless of subsequent characters in the field.

BMS requires you to initialize to nulls any area to be used to build a map. This is done by moving nulls (X'00') to the mapnameO field in the symbolic map structure. See "Initializing the output map" on page 386 for more information. BMS uses nulls in attribute positions and in the first position of data to indicate that no change is to be made to the value in the map. If you are reusing a map area in a program or in a TIOA, you should take special care to clear it in this way.

Address CICS areas correctly

There are several ways to check that CICS areas are addressed correctly. Ensure that:

- Each COBOL program with a LINKAGE SECTION structure that exceeds 4KB has the required definition and the setting of more than one contiguous BLL cell.
- Every BLL pointer points to an area that is a 01-level item.
- Call level DL/I is only used with PSBs that are correctly addressed.

Use the MAPONLY option when possible

The MAPONLY option sends only the ***constant*** data in a map, and does not merge any variable data from the program. The resulting data stream is not always shorter, but the operation has a shorter path length in BMS. When you send a skeleton screen to be used for data entry, you can often use MAPONLY.

Send only changed fields to an existing screen

Sending only changed fields is important when, for example, a message is added to the screen, or one or two fields on an input screen are highlighted to show errors. In these situations, you should use the DATAONLY option to send a map that consists of nulls except for the changed fields. For fields where the only the attribute byte has changed, you need send only that byte, and send the remaining fields as nulls. BMS uses this input to build a data stream consisting of only the fields in question, and all other fields on the screen remain unchanged.

It may be tempting to ignore this advice and send an unnecessarily long data stream. For example, when a program that is checking an input screen for errors finds one, there are two options.

- It can add the error information to the input map (highlighted attributes, error messages, and so on) and resend it.
- It can build an entirely new screen, consisting of just the error and message fields.

The former is slightly easier to code (you do not need to have two map areas or move any fields), but it may result in very much longer transmissions because the output data stream contains the correct input fields as well as the error and message fields. In fact, it may even be longer than the original input stream because, if there were empty or short fields in the input, BMS may have replaced the missing characters with blanks or zeros.

With the 3270 hardware, if the input stream for a terminal exceeds 256 bytes, the terminal control unit automatically breaks it up into separate transmissions of 256 bytes maximum. This means that a long input stream may require several physical I/O operations. Although this is transparent to the application program, it does cause additional line and processor overhead. The ***output*** stream is generally sent in a single transmission.

Design data entry operations to reduce line traffic

Often, users are required to complete the same screen several times. Only the data changes on each cycle; the titles, field labels, instructions, and so on remain unchanged. In this situation, when an entry is accepted and processed, you can respond with a SEND CONTROL ERASEAUP command (or a map that contains only a short confirmation message and specifies the ERASEAUP option). This causes all the **unprotected** fields on the screen (that is, all the input data from the last entry) to be erased and to have

their MDTs reset. The labels and other text, which are in protected fields, are unchanged, the screen is ready for the next data-entry cycle, and only the necessary data has been sent.

Compress data sent to the screen

When you send unformatted data to the screen, or create a formatted screen outside BMS, you can compress the data further by inserting set buffer address (SBA) and repeat-to-address (RA) orders into the data stream. SBA allows you to position data on the screen, and RA causes the character following it to be generated from the current point in the buffer until a specified ending address. SBA is useful whenever there are substantial unused areas on the screen that are followed by data. RA is useful when there are long sequences of the same character, such as blanks or dashes, on the screen. However, you should note that the speed with which RA processes is not uniform across all models of 3270 control units. You should check how it applies to your configuration before use.

CICS provides an exit that is driven just before output is sent to a terminal (XTC OUT). You may want to add SBA and RA substitutions to this exit to compress the data stream using a general subroutine. This has the dual benefit of removing compression logic from your application program and of applying to all output data streams, whether they are produced by BMS or not.

Use nulls instead of blanks

You should note that, outside BMS, nulls have no special significance in an ***output*** data stream. If you need a blank area on a screen, you can send either blanks or nulls to it; they take up the same space in the output stream. However, if the blank field is likely to be changed by the user and subsequently read, use nulls, because they are not transmitted back.

Use methods that avoid the need for nulls or blanks

For any ***large*** area of a screen that needs to be blank, you should consider methods other than transmitting blanks or nulls; for example, when using BMS, putting SBA and RA orders directly into the data stream, or using the ERASE and ERASEAUP options.

Page-building and routing operations

BMS page-building facilities provide a powerful and flexible tool for building and displaying long messages, sending messages to multiple destinations, and formatting a single message for several devices with different physical characteristics. However, as for any high-function tool, it requires a substantial overhead.

For more information, see “Efficient data set operations” on page 255. You might need the page-building option (ACCUM) when:

- Sending messages whose length exceeds the capacity of the output device (multipage output)
- Using destinations other than the input terminal
- Sending pages built from multiple maps
- Using the BMS page-copy facility

Sending multipage output

Transactions that produce large output messages, consisting of many screen-size pages, tend to tax system resources. First, all the pages have to be created, which involves processor activity, execution of the CSPG transaction, and data set I/O activity. The pages must then be saved in temporary storage. If the terminal user looks at every page in a message, many transactions are run to process the paging requests, each of which needs line and processor overhead. Obviously some overhead is caused by the size and complexity of the transaction, and it might be unavoidable. Indeed, if several users are scrolling rapidly through paged output at the same time, the transactions needed can monopolize a system.

If users really need to see all the pages, and need to scroll backward and forward frequently, it can be more efficient to produce all the pages at the same time and present them using "traditional" CICS paging

services. However, if users need only a few of the pages, or can easily specify how far back or forward in the message they would like to scroll, there are two choices:

1. First, construct a pseudoconversational transaction to produce just one screen of output. The first time this transaction is run, it produces the first page of the many-page output. The output screen contains space for users to indicate the page they want next. The transaction always sets the next transaction identifier to point to itself, so that it can display the requested page when it is next run.

   You will probably want to give users some of the options that CICS provides (such as one page forward, one page back, and skip to a selected page) and some relevant to the application, such as a data set key at which to begin the next page of output.
2. The alternative is to page-build a multipage output message with the ACCUM option, but to limit the number of pages in the message (say to five). Users page through the subset pages with the typical CICS page commands. On the last screen of the output, you add an indication that there is more output and a place for them to indicate whether they want to see the next segment. As in the first example, the next transaction identifier is set to the original transaction so that, if CICS does not receive a paging command, it invokes that transaction.

Sending messages to destinations other than the input terminal

If you need to send a message to a terminal other than the input terminal associated with a task, BMS routing might be the most efficient way of doing so. This is especially so if the message must be sent to multiple destinations or if it involves multiple pages. Routing is the recommended method if the message recipients need CICS paging commands to access it.

However, if neither of the previous conditions apply, you have a choice of two other methods of delivering output to a terminal not associated with the transaction.

1. You can use a START command, with the TERMID option, to specify the terminal to which you want to write and the FROM option to specify the data you want to send. Your own transaction is the started transaction. It issues an RETRIEVE command for the message and then sends it to its own terminal. See START for programming information about the START command.
2. Similarly, you can put messages destined for a particular terminal on to an intrapartition transient data queue. The definition for the transient data queue must specify:

   - The destination as a TERMINAL
   - The terminal identifier
   - A trigger level
   - A transaction name

   Your own transaction reads the transient data queue and sends the message to its terminal. It repeats this sequence until the queue is empty, and then terminates. The trigger level you specified means that it is invoked every time the specified number of messages have been placed on the queue.

Note: Because of the overhead associated with routing messages (by whatever means), you should use facilities such as ROUTE=ALL with caution.

Sending pages built from multiple maps

Although you can easily build a screen gradually using different maps, you can sometimes avoid considerable overhead by not using page-building operations, especially where there is only one screen of output and no other need for paging. An example of this is an application whose output consists of a header map, followed by a variable number of detail segments, sent with a second map, and finally a trailer map following the detail. Suppose the average output screen for such an application contains eight (two-line) detail segments, plus header and trailer, and all this fits on a single screen. Writing this screen with page-building requires 11 BMS calls (header, details, trailer, and page-out) whereas, if the program builds the output screen internally, it only needs one call.

Using the BMS page-copy facility

Because the individual pages that make up an accumulated BMS message are saved in temporary storage, BMS enables the terminal user to copy individual pages to other terminals. However, if the ability to copy is the only reason for using page-building, you should consider using either the 3274 control unit copy facilities or the CICS **copy key** facility instead.

The 3274 copy facilities require no participation from CICS and no transmission, and are by far the most efficient method. The CICS copy key facility does have an overhead (see “Requests for printed output” on page 346), although of a different type from the BMS copy facility. It also has destination restrictions that do not apply to BMS copying.

Chapter 5. Developing for asynchronous requests

You can develop a CICS application using the asynchronous API commands to start one or more child tasks and fetch responses from them. When you develop applications for use with the asynchronous API, you will need to consider the flow of the asynchronous API commands and the available options for each command. Channel handling must also be considered if you want to pass input or receive responses as part of an asynchronous request.

Using the TIMEOUT and NOSUSPEND options on the **FETCH CHILD** and **FETCH ANY** commands, you have more control over parent-child behavior, making it even easier to reach response time requirements with asynchronous applications.

Programming considerations and recommendations

Here are some practical guidance and recommended approaches on using the CICS asynchronous API. However, your environments and applications might vary and benefit from different approaches. For details, see the IBM Redbooks® publication *IBM CICS Asynchronous API: Concurrent Processing Made Simple*.

Child tasks should perform GETs of data and leave UPDATEs to the parent task.
Parent and child tasks are separate units of work (UOW). If, for example, a parent abends, it can back-out its own work but will be unconnected from the execution of the child tasks. If the child tasks are limited to GET actions, they can be safely discarded without compromising the state of the data. It is possible for child tasks to perform UPDATEs on data. However, you might be required to code compensation logic to revert changes, if the need arises.

Allow the same parent program to run and fetch child tasks.
For manageability, the parent program that begins a child task should remain the program that fetches the results from the child task. This process is particularly useful if you are using the **FETCH ANY** command, because it can become difficult to match the **RUN TRANSID** to **FETCH ANY** commands if they reside in different programs. Also note that fetched channels are scoped to a link level. So passing results can be problematic if it is not the parent that fetches the child's results.

Tip: If you want to retrieve response data from a child task, specify CHANNEL in **EXEC CICS RUN TRANSID** even if you have no input data to the child task.

Long-running parents should use the FREE CHILD command.
In a long-running parent application, there can be a build-up to control blocks as child tasks complete. In addition, there can be a build-up of sizeable channel data. Without task termination, the CICS system cannot safely discard this data. It is advisable that a long-running parent task deletes any fetched channels and issues **FREE CHILD** commands against child tasks that are no longer required.

Keep track of fetched channels.
Data is passed between parent and child tasks using CICS channel and containers. To return data from a child, the child task updates containers on its current channel. During the termination of child tasks, the channel is managed or owned by the CICS AS domain. The parent task can retrieve the child's channel by specifying the CHANNEL parameter on the **FETCH CHILD** or **FETCH ANY** commands. This approach is more active than a simple inquiry. The channel will bind to the issuing parent.

Review MAXTASK and set transaction classes.
The simplicity of the asynchronous API might generate more transactions in a CICS system. It's recommended to review the **MAXTASK** parameter setting. It is advised to have controls in place to limit the amount of work that a region accepts that cause tasks to be run asynchronously.

Parameterize timeouts.
In many cases, the results of a child are needed for the parent to continue their business logic. If you require a timeout, or if there is a possibility that you might need to in future, you can code the parent to specify a TIMEOUT parameter on the **FETCH** command. The value of the timeout should be

© Copyright IBM Corp. 1974, 2023

parameterized, such as being read from a file or database table, earlier in the application. By inserting the TIMEOUT parameter and parameterizing the TIMEOUT value, you prevent the need for a future code update or recompilation.

You can use TIMEOUT and NOSUSPEND on your **FETCH CHILD** and **FETCH ANY** commands to improve application versatility and meet business needs in a variety of situations:

- For any child tasks from which you will definitely need a response, use **FETCH** without TIMEOUT or NOSUSPEND.
- For child tasks from which you need the response in a timely fashion, specify a TIMEOUT value on your **FETCH** command.
- For child tasks where no response is acceptable, specify NOSUSPEND on your **FETCH** command.

Examples

- "Example: A credit card application" on page 444
- "Example: Passing the child token to a linked program to fetch response from the child task" on page 446
- "Example: CICS Asynchronous API Fetch variation" on page 447

Example: A credit card application

In this example, a credit card application is used to demonstrate how the commands can be used.

Start the credit check child tasks

1. Put the customer details onto a new channel:.

```
EXEC CICS PUT CONTAINER('CUSTDETAILS') CHANNEL(customerDetails)
```

2. Call the first credit check service with a channel:

```
EXEC CICS RUN TRANSID('CRD1') CHILD(creditCheck1_tkn) CHANNEL(customerDetails)
```

3. Call the second credit check service with the same channel:

```
EXEC CICS RUN TRANSID('CRD2') CHILD(creditCheck2_tkn) CHANNEL(customerDetails)
```

A parent program can continue with other processing after issuing an **EXEC CICS RUN TRANSID** command, and can issue an **EXEC CICS FETCH** command at any point afterward to retrieve the results of a child task.

Get a response from a specific child task

1. Fetch the response from the first child:

```
EXEC CICS FETCH CHILD(creditCheck1_tkn) CHANNEL(credReply1_chan)
          COMPSTATUS(credReply1_status)
```

   In this case, parent processing is blocked until the child task returns a result. Use of the optional parameters NOSUSPEND and TIMEOUT can prevent or limit blocking as preferred.
2. Check the response code of the **EXEC CICS FETCH CHILD** command to ensure the command executed normally.
3. If the **EXEC CICS FETCH CHILD** command executed normally, then check the COMPSTATUS of the child to ensure it completed successfully.
   - If COMPSTATUS indicates that the child task completed successfully, then get the response from the reply channel.

```
EXEC CICS GET CONTAINER('RESULT') CHANNEL(credReply1_chan)
```

- If COMPSTATUS indicates that the child task abended, then the logic of the parent application should decide what to do. CICS will clean up all resources when the parent transaction completes.

4. Repeat steps 1 through 3 for the second child task.

Get a response from any completed child task

It's also possible to fetch a response from any completed child task using the **EXEC CICS FETCH ANY** command, which will return the result of whichever child task completed first:

```
EXEC CICS FETCH ANY(creditCheck_tkn) CHANNEL(credReply_chan)
          COMPSTATUS(credReply_status)
```

Check the completion of the **FETCH ANY** command and the child's COMPSTATUS as normal.

Free a child

As part of your parent task, you can choose to free child tasks, which will tidy up the associated resources and keep system storage use to a minimum, which is especially useful for long-running parent tasks. If, for example, you started three child tasks, but only need the results of one, then you can clean up the remaining two child tasks using **FREE CHILD**.

```
EXEC CICS FREE CHILD(creditCheck1_tkn)
EXEC CICS FREE CHILD(creditCheck2_tkn)
```

Table 52. Reference: Optional parameters

Command	Optional parameter	Usage
EXEC CICS RUN TRANSID	CHANNEL	Use it to: • Pass data to a child. • Receive data from a child. • Both pass and receive data to and from a child.
	RESP	Use it to see if a command executed successfully

Table 52. Reference: Optional parameters (continued)		
Command	**Optional parameter**	**Usage**
EXEC CICS FETCH commands	CHANNEL	Use it to receive data from a child task, as long as a channel was supplied on the initial **RUN TRANSID** command.
	NOSUSPEND	Use it to stop a **FETCH** command from blocking: the command will return immediately regardless of whether a child task has completed. If the child task has finished when the command is issued, the command will return immediately with a NORMAL response. If the child hasn't finished, the NOTFINISHED condition will occur with RESP2 value of 52.
	TIMEOUT	Use it to make a **FETCH** command wait for a specified amount of time before returning. You can specify the maximum time in milliseconds that the parent task will wait for a child task to complete. When TIMEOUT is specified with a non-zero value, the parent task will not be subject to **DTIMOUT**. If the child task finishes before TIMEOUT is reached, the command will return immediately. If the child task hasn't finished before TIMEOUT is reached, a NOTFINISHED condition will occur with a RESP2 value of 53.
	ABCODE	If **COMPSTATUS** indicates that a child has abended, then **ABCODE** will contain the abend code of the child.

Example: Passing the child token to a linked program to fetch response from the child task

The following steps are an example of how input and output are passed between parent and child tasks using channels.

1. The parent task, PROGP1, passes input to a child task in a channel, and then links it to a local program PROGP2.

```
EXEC CICS PUT CONTAINER('REQ') FROM(REQUEST) CHANNEL('ASYNC-CHANNEL')
EXEC CICS RUN TRANSID('CHLD') CHANNEL('ASYNC-CHANNEL') CHILD(CHILD-TOKEN)
EXEC CICS PUT CONTAINER('CHILDTOKEN')  FROM(CHILD-TOKEN) CHANNEL('ASYNC-CHANNEL')
EXEC CICS LINK PROGRAM('PROGP2') CHANNEL('ASYNC-CHANNEL')
```

   A copy of the ASYNC-CHANNEL created by the parent is passed to the child, and the copy becomes the current channel for the child's initial program. The name of the channel is retained by the copying process, so if the child program PROGC1 issued **EXEC CICS ASSIGN CHANNEL** then ASYNC-CHANNEL would be returned.

2. The child task, with transaction ID CHLD and program PROGC1, gets the input from its current channel:

```
EXEC CICS GET CONTAINER('REQ')
```

The child will then continue with its logic, generate a response for the parent task, and then put it into a container within its current channel:

```
EXEC CICS PUT CONTAINER('RESP') FROM(RESPONSE)
EXEC CICS RETURN
```

3. The parent task, PROGP2, fetches the child response and continues with some business logic before returning to PROGP1:

```
EXEC CICS GET CONTAINER ('CHILDTOKEN') INTO(CHILD-TOKEN)
EXEC CICS FETCH CHILD(CHILD-TOKEN) CHANNEL (FETCHED-CHANNEL)
EXEC CICS GET CONTAINER('RESP') CHANNEL(FETCHED-CHANNEL)
```

When the parent task fetches the response channel from the child, CICS generates a unique channel name and binds the channel to the current program link level. In this example, the fetched channel will be owned by PROGP2. When PROGP2 returns to PROGP1, the fetched channel will be deleted by CICS. A child's response channel can only be fetched once by the parent task.

Example: CICS Asynchronous API Fetch variation

Before you begin: The source code for this example is available in full on GitHub where you will also find setup and running instructions.

This COBOL example has a parent program ASYNCPG1 to demonstrate the use of **EXEC CICS FETCH** commands with and without option TIMEOUT or NOSUSPEND. The application also have four child programs: ASYNCCH1, ASYNCCH2, ASYNCCH3, ASYNCCH4.

ASYNCPG1 starts four asynchronous child tasks. It then does 3 different flavors of fetch to get the response, according to the business requirement, as following:

- The parent program needs the response from the first child, so use **FETCH CHILD** with no SUSPEND or TIMEOUT option.

```
EXEC CICS FETCH CHILD(CHLDTOKN1) CHANNEL(CHLDCHNL1)
          COMPSTATUS(CVDA)
          RESP(W-RESP) RESP2(W-RESP2)
          END-EXEC.
```

- The parent program can wait for up to 1000 milliseconds when fetching the response from the second child, so use **FETCH CHILD TIMEOUT**.

```
EXEC CICS FETCH CHILD(CHLDTOKN2) CHANNEL(CHLDCHNL2)
          TIMEOUT(TIMEOUT1)
          COMPSTATUS(CVDA)
          RESP(W-RESP) RESP2(W-RESP2)
          END-EXEC.
```

where TIMEOUT1 is defined as shown below and can be changed dynamically in the program or by reading from a file or database to suit your environment:

```
01 TIMEOUT1     PIC S9(8) USAGE BINARY VALUE 1000.
```

- For the third and fourth children, the parent program can have any result, but can't afford to wait, so use **FETCH CHILD NOSUSPEND**.

```
EXEC CICS FETCH ANY(ANYTOKN) CHANNEL(ANYCHNL)
          NOSUSPEND COMPSTATUS(CVDA)
          RESP(W-RESP) RESP2(W-RESP2)
          END-EXEC.
```

Chapter 6. Developing Assembler language applications

Use this information to help you code assembler language programs that you want to use as CICS application programs.

Working storage

Working storage for assembler language programs is allocated either above or below the 16 MB line, according to the value of the **DATALOCATION** parameter on the PROGRAM definition in the CSD.

Sample programs

A set of sample application programs is provided to show how **EXEC CICS** commands can be used in a program written in assembler language. These programs are AMODE(64) and use relative addressing, except for DFH$AREP. DFH$AREP uses relative addressing but is AMODE(31) because it demonstrates the use of the HANDLE CONDITION LABEL command.

Table 53. Sample programs

Sample program	Map set	Map source	Transaction ID
DFH$AMNU **Operator instruction (3270)**	DFH$AGA	DFH$AMA	AMNU
DFH$AALL **Update (3270)**	DFH$AGB	DFH$AMB	AADD, AINQ, AUPD
DFH$ABRW **Browse (3270)**	DFH$AGC	DFH$AMC	ABRW
DFH$AREN **Order entry (3270)**	DFH$AGK	DFH$AMK	AORD
DFH$ACOM **Order entry queue print (3270)**	DFH$AGL	DFH$AML	AORQ
DFH$AREP **Report (3270)**	DFH$AGD	DFH$AMD	AREP

The transaction and program definitions are provided in group DFH$AFLA in the CSD and can be installed using the following command:

```
CEDA INSTALL GROUP(DFH$AFLA)
```

The following record description files are provided:

- DFH$AFIL: FILEA record descriptor
- DFH$AL86: L860 record descriptor

Assembler language programming restrictions and requirements

Some restrictions and requirements apply to an assembler language program that is used as a CICS application program.

LEASM option

The following restrictions apply to an assembler language program that is translated with the LEASM option:

- Register 2 cannot be used as a code base register.

© Copyright IBM Corp. 1974, 2023

- Register 12 is reserved by Language Environment (LE) to point to the Language Environment common anchor area (CAA) and so cannot be used at all by the program without being saved and restored as appropriate.
- Register 13 must be used as the only working storage base register.
- The program cannot be a Global User Exit program (GLUE) or a Task-Related User Exit program (TRUE).
- The program must not use, or depend on, any AMODE(24) code.
- AMODE(64) programs are not supported.

LENGTH option in EXEC CICS commands

When you specify a LENGTH option for a CICS command, ensure that the way you use in assembler language specifies a valid halfword length. Do not specify a zero length, or a variable that the CICS translator cannot recognize, because this might cause a storage violation or a program check. For more information, see Assembler language argument values.

The following example of a LINK command specifies the length correctly and passes the address of the variable COMMAL to the command processor:

```
EXEC CICS LINK PROGRAM('PROG2') COMMAREA(COMMA)
LENGTH(COMMAL)
...
COMMA DC CL20'This is the COMMAREA'
COMMAL DC H'20'
```

The following example also specifies the length correctly:

```
EXEC CICS LINK PROGRAM('PROG2') COMMAREA(COMMA)
LENGTH(=AL2(COMMAL))
...
COMMA DC CL20'This is the COMMAREA'
COMMAL EQU *-COMMA
```

The following example is incorrect because the CICS translator cannot know the type of the variable COMMAL, and passes the value of COMMAL as the address of the halfword field that contains the length. This could provide a random length value, or a program check if the storage at that address is not available.

```
EXEC CICS LINK PROGRAM('PROG2') COMMAREA(COMMA)
LENGTH(COMMAL)
...
COMMA DC CL20'This is the COMMAREA'
COMMAL EQU *-COMMA
```

31-bit addressing

The following restrictions apply to an assembler language application program that runs in 31-bit addressing mode:

- The COMMAREA option is restricted in a mixed addressing mode transaction environment. For a discussion of the restriction, see “Using mixed addressing modes” on page 152.
- CICS does not allow the use of HANDLE ABEND LABEL in assembler language programs that do not use the DFHEIENT and DFHEIRET macros. Assembler language programs that use the Language Environment stub CEESTART should either use HANDLE ABEND PROGRAM or a Language Environment service such as CEEHDLR. See “ Language Environment abend and condition handling ” on page 535 for information about CEEHDLR.

The following restriction applies to an AMODE(24) or AMODE(31) assembler language application program that uses 64-bit registers to use 64-bit addressing mode or 64-bit binary operations:

- CICS does not always preserve the high-order words of 64-bit registers. You must save them before you invoke a CICS service, and restore them before using the 64-bit registers again.

64-bit addressing

CICS supports the program execution of non-Language Environment AMODE(64) assembler language CICS applications. Your program must use relative addressing. See “Developing AMODE(64) assembler language programs” on page 461. All CICS API commands, except for those listed later in this section, are supported. For information about the CICS API, see CICS API commands.

The following APIs are not supported for AMODE(64) programs:

- CICSPlex SM application programming interface (API)
- CICS API commands for use in APPC basic conversations (GDS commands)
- Front End Programming Interface (FEPI)
- Common Programming Interface (CPI) Communications API
- DL/I requests

The following interfaces are not supported for AMODE(64) programs:

- CICS Db2 interface
- CICS-MQ bridge external interface

The following CICS API commands are provided for use with AMODE(64) programs:

- FREEMAIN64
- GETMAIN64
- GET64 CONTAINER
- PUT64 CONTAINER

The following CICS API commands, which are related to nonstructured exception handling, are not supported for AMODE(64) assembler language programs:

- HANDLE ABEND LABEL(label)
- HANDLE AID
- HANDLE CONDITION
- IGNORE CONDITION
- POP HANDLE
- PUSH HANDLE

The translator detects use of these commands.

AMODE(64) assembler language programs cannot be invoked by a COBOL, C, C++, or PL/I CALL statement, or by an assembler language CALL macro. However, assembler language application programs can be invoked by COBOL, C, C++, PL/I, or assembler language application programs by using LINK or XCTL commands.

For AMODE(64) assembler language programs, use of the NOHANDLE condition is implied on all EXEC CICS commands. This means that no action will be taken for any condition that results from the execution of that command. You can use the RESP option with any command to test whether any condition was raised during execution of that command. See RESP and RESP2 options and NOHANDLE option.

For AMODE(64) assembler language programs, CICS uses the whole of the 64-bit register. Ensure that all of the register is valid.

A COMMAREA cannot be in 64-bit storage. For more information the COMMAREA, see “Passing data to other programs by using COMMAREA” on page 149.

AMODE 64 task-related user exits (TRUEs) are not supported, and AMODE(64) applications cannot invoke a TRUE.

For more information about using 64-bit addressing mode and 64-bit binary operations, see z/OS MVS Programming: Assembler Services Guide.

64-bit residency

CICS does not support 64-bit residency mode (RMODE(64)) and treats any RMODE(64) programs as RMODE(31). That is, RMODE(64) programs are loaded into 31-bit (above-the-line) storage, not 64-bit (above-the-bar) storage.

Access registers

The following restrictions apply to an assembler language application program that uses access registers to use the extended addressability of z/Architecture® systems:

- You must be in primary addressing mode when invoking any CICS service. The primary address space must be the home address-space. All parameters passed to CICS must reside in the primary address space.
- CICS does not always preserve access registers. You must save them before you invoke a CICS service, and restore them before using the access registers again.

For more guidance information about using access registers, see z/OS MVS Programming: Extended Addressability Guide.

BAKR instructions (branch and stack)

When using BAKR instructions (branch and stack) to provide linkage between assembler language programs, ensure that the linked-to program does not issue EXEC CICS requests. If CICS receives control and performs a task switch before the linked-to program returns by a PR instruction (program return), other tasks might be dispatched and issue further BAKR / PR calls. These calls modify the linkage-stack and result in the wrong environment being restored when the original task issues its PR instruction.

Instructions that cannot be used

You cannot use the following instructions in an assembler language program that is used as a CICS application program:

COM
Identify blank common control section.

ICTL
Input format control.

OPSYN
Equate operation code.

Comments

If you want to add comments against CICS commands, you can do this, in assembler language only, by using a period or a comma as a delimiter after the last argument. For example:

```
EXEC CICS ADDRESS EIB(MYUEIB), @F1A
```

If a period or a comma is used with an EXEC CICS command, the following line must begin between column 2 and column 16, with the continuation character in column 72. The following line cannot start after column 17. If there is no comma or period added, the following line must begin at or after column 2 and end by column 71, with the continuation character in column 72.

Language Environment coding requirements for assembler language applications

Assembler language programs are classified as either conforming or non-conforming with respect to Language Environment. Conformance depends on the linkage and register conventions observed, rather than the assembler used.

A Language Environment-conforming assembler language routine is defined as one that is coded using the CEEENTRY and associated Language Environment macros.

Conformance governs the use of assembler programs by call from an HLL program. Both conforming and non-conforming assembler language subroutines can be called either statically or dynamically from C, C++, COBOL or PL/I. However, there are differences in register conventions and other requirements for the two types. For example, to communicate properly with Language Environment-conforming assembler language routines, you must observe certain register conventions on entry to the assembler language routine, while it is running, and on exit from the assembler language routine.

Rules for mixing languages, including assembler language, are discussed in "Mixing languages in Language Environment" on page 538.

64-bit addressing mode is not supported for Language Environment-conforming assembler language programs.

For more detailed information, or for explanations of the terms used in this section, see z/OS Language Environment Programming Guide.

Conforming MAIN programs

If you are coding a new assembler language MAIN program that you want to conform to the Language Environment interface, or if your assembler language routine calls Language Environment services, observe the following:

- Use the macros provided by Language Environment. For a list of these macros, see z/OS Language Environment Programming Guide.
- Ensure that the CEEENTRY macro contains the option MAIN=YES. (MAIN=YES is the default).
- Translate your assembler language routine with *ASM XOPTS(LEASM) or, if it contains CICS commands, with *ASM XOPTS(LEASM NOPROLOG NOEPILOG).

Conforming subroutines

If you are coding a new assembler language subroutine that you want to conform to the Language Environment interface, or if your assembler language routine calls Language Environment services, observe the following:

- Use the macros provided by Language Environment. For a list of these macros, see z/OS Language Environment Programming Guide.
- Ensure that the CEEENTRY macro contains the option MAIN=NO. (MAIN=YES is the default).
- Translate your assembler language routine with *ASM XOPTS(NOPROLOG NOEPILOG) if it contains CICS commands.
- Ensure that the CEEENTRY macro contains the option NAB=NO if your routine is invoked by a static call from VS COBOL II. (NAB is Next Available Byte (of storage). NAB=NO means that this field might not be available, so the CEEENTRY macro generates code to find the available storage.)

Register conventions for entry into conforming routines

On entry into a Language Environment-conforming assembler language subroutine, the following registers must contain the following values when NAB=YES is specified on the CEEENTRY macro:

R0
Reserved

R1
Address of the parameter list, or zero

R12
Common anchor area (CAA) address

R13
Caller's dynamic storage area (DSA)

R14
Return address

R15
Entry point address

Language Environment-conforming HLLs generate code that follows these register conventions, and the supplied macros do the same when you use them to write your Language Environment-conforming assembler language routine. On entry to an assembler language routine, CEEENTRY saves the caller's registers (R14 through R12) in the DSA provided by the caller. It allocates a new DSA and sets the NAB field correctly in this new DSA. The first halfword of the new DSA is set to binary zero and the back chain in the second word is set to point to the caller's DSA.

Register conventions while a conforming routine is running

R13 must point to the DSA of the routine at all times while the Language Environment-conforming assembler language routine is running.

At any point in your code where you call another program, R12 must contain the common anchor area (CAA) address, except in the following cases:

- When calling a COBOL program.
- When calling an assembler language routine that is not Language Environment-conforming.
- When calling a Language Environment-conforming assembler language routine that specifies NAB=NO on the CEEENTRY macro.

Register conventions for exit from conforming routines

On exit from a Language Environment-conforming assembler language routine, R0, R1, R14, and R15 are undefined. All the other registers must have the contents that they had upon entry.

The CEEENTRY macro automatically sets a module to AMODE (ANY) and RMODE (ANY). If you are converting an existing assembler language routine to be Language Environment-conforming and the routine contains data management macros coded using 24-bit addressing mode, you should change the macros to use 31-bit mode. If it is not possible to change all the modules in a program to use 31-bit addressing mode, and if none of the modules explicitly sets RMODE (24), you should set the program to be RMODE (24) during the link-edit process.

Non-conforming assembler language routines running under Language Environment

Observe the following conventions when running non-Language Environment-conforming assembler language routines under Language Environment:

- R13 must contain the address of the executing routine's register save area.
- The first two bytes of the register save area must be binary zeros.
- The register save area back chain must be set to a valid 31-bit address (the high-order byte must be zero if it is a 24-bit address).

If your assembler language routine relies on C, C++, COBOL, or PL/I control blocks (for example, a routine that tests flags or switches in these control blocks), check that these control blocks have not changed

under Language Environment. For more information, see the *Compiler and Run-Time Migration Guide* for the language in use.

Non-conforming assembler language routines cannot use Language Environment callable services.

Coding the DFHEIENT macro for AMODE(24) and AMODE(31) assembler language programs

For AMODE(24) and AMODE(31) programs, the DFHEIENT macro calls the PROLOG program, which allocates working storage to hold any user variables and for CICS use. The PROLOG program sets up the CICS portion of this storage. When the program returns, this code sets up the registers specified by the DFHEIENT parameters. The values provided by the DFHEIENT macro that the translator inserts automatically might be inadequate for application programs that produce a translated output greater than 4095 bytes. In this situation, you can provide your own version of the DFHEIENT macro and specify the NOPROLOG translator option to prevent the translator from automatically inserting its version of the DFHEIENT macro.

For example, by default, the translator sets up only one base register (register 3), which might cause addressability problems for your program. You can provide your own DFHEIENT macro with the CODEREG operand so that you can specify more than one base register. If you code your own version of the DFHEIENT macro with the same label as the CSECT statement, it can replace the CSECT statement in your source program. If you code the DFHEIENT macro without a label, it must immediately follow the CSECT statement.

You can specify the following operands for the DFHEIENT macro:

CODEREG
Specify a value of 0 (the default) to specify relative addressing.

Or specify base registers. Note that registers 13,14,15, and 1 are not allowed.

DATAREG
Specify one or more working storage registers for the application program. The default is register 13, and it is advisable to use register 13 as your first data dynamic-storage register. If you do not, the code generated by the DFHECALL macro adds extra instructions to manipulate register 13. The DFHECALL macro ensures that register 13 addresses the save area that DFHEISTG defined in dynamic storage.

EIBREG
Specify the register to use to address the EXEC interface block (EIB). The default is register 11.

STATREG
Specify one or more static registers for the application program to use. The default is register 3.

STATIC
Specify the assembler label of the start of the static area. You must specify a value; there is no default for this parameter.

Using base registers in your program

You can use the following operands to specify base registers:

- CODEREG - base registers (registers 13,14,15, and 1 not allowed)
- DATAREG - dynamic-storage registers
- EIBREG - register to address the EIB

For example, the following simple assembler language application program uses the BMS command SEND MAP to send a map to a terminal.

```
INSTRUCT CSECT
         EXEC CICS SEND MAP('DFH$AGA') MAPONLY ERASE
         END
```

The following example code increases the number of base and data registers for this program:

```
INSTRUCT DFHEIENT CODEREG=(2,3,4),
         DATAREG=(13,5),
         EIBREG=6
         EXEC CICS SEND
         MAP('DFH$AGA')
         MAPONLY ERASE
         END
```

The symbolic register DFHEIPLR is equated to the first DATAREG either explicitly specified or obtained by default. The DFHECALL macro ensures that register 13 addresses the save area that DFHEISTG defined in dynamic storage, so it is advisable to use register 13 as your first data dynamic-storage register. If you do not, the code generated by DFHECALL adds extra instructions to manipulate register 13.

DFHEIPLR is assumed by the expansion of a CICS command to contain the value set up by DFHEIENT. You should either dedicate this register or ensure that it is restored before each CICS command.

Using relative addressing in your program

When you use relative addressing, you do not need to use any base registers to address your program instructions, but you must use at least one register to address static data in your program. Specify the following operands on the DFHEIENT macro:

- CODEREG=0 to specify that no registers are to be used to address program instructions.
- STATREG to specify one or more registers to address the static data area in your program.
- STATIC to specify the address of the start of the static data in your program.

If you use relative addressing, use the IEABRCX DEFINE macro (provided by z/OS) to redefine the assembler mnemonics for branch instructions to use relative branch instructions. Also, ensure that any LTORG statements, and instructions that are the target of EXECUTE instructions, appear after the label specified in the STATIC operand. For example:

```
         IEABRCX DEFINE                 Define relative branch mnemonics
RELATIVE DFHEIENT CODEREG=0,STATREG=(8,9),STATIC=MYSTATIC
         ....
         EX    R2,VARMOVE             Execute instruction in static area
         ....

MYSTATIC DS    0D                     Static data area
MYCONST  DC    C'constant'            Static data value
VARMOVE  MVC   WORKA(0),WORKB         Executed instruction
         LTORG ,                       Literal pool
```

For more information about the IEABRCX macro, see IEABRCX - Relative branch macro extension in z/OS MVS Programming: Assembler Services Reference IAR-XCT.

Where LTORG statements are needed

Assembler language programs that are translated with the DLI option have a DLI initialization call inserted after each CSECT statement. Assembler language programs that are larger than 4095 bytes and that do not use the CODEREG operand of the DFHEIENT macro to establish multiple base registers, must include an LTORG statement to ensure that the literals, generated by either DFHEIENT or a DLI initialization call, fall in the range of the base register.

In general, an LTORG statement is needed for every CSECT that exceeds 4095 bytes in length.

Related information

- For reference information about the DFHEIENT macro, see DFHEIENT macro.
- For more information about the DFHEIRET macro, see “Coding the DFHEIRET macro for assembler language programs” on page 458 and DFHEIRET macro.
- For more information about the DFHEISTG storage, see “Extending dynamic storage for assembler language programs” on page 460.

- For reference information for the EXEC interface, including design overview, control blocks, modules, exits, and trace, see EXEC interface in Reference.

Coding the DFHEIENT macro for AMODE(64) assembler language programs

For AMODE(64) programs, the DFHEIENT macro calls the AMODE(64) PROLOG program, which allocates working storage to hold any user variables and for CICS use. The AMODE(64) PROLOG program sets up the CICS portion of this storage. When the program returns, this code sets up the registers specified by the DFHEIENT parameters. You must specify the DFHEIENT macro parameters to specify that your program uses relative addressing instructions, and you must use at least one register to address static data in your program.

You can specify the following operands for the DFHEIENT macro:

CODEREG
Specify a value of 0 (the default) to specify relative addressing.

DATAREG
Specify one or more working storage registers for the application program. The default is register 13, and it is advisable to use register 13 as your first data dynamic-storage register. If you do not, the code generated by the DFHECALL macro adds extra instructions to manipulate register 13. The DFHECALL macro ensures that register 13 addresses the save area that DFHEISTG defined in dynamic storage.

EIBREG
Specify the register to use to address the EXEC interface block (EIB). The default is register 11.

STATREG
Specify one or more static registers for the application program to use. The default is register 3.

STATIC
Specify the assembler label of the start of the static area. You must specify a value; there is no default for this parameter.

AMODE(64) programs must use relative addressing, because only relative addressing is supported. Use the NOPROLOG translator option and specify the DFHEIENT macro with the appropriate parameters for relative addressing. If you do not specify the DFHEIENT macro, the translator inserts a DFHEIENT macro without the required parameters and the following error occurs:

```
12,DFHEIENT - AMODE 64 - STATIC REQUIRED
```

For relative addressing, you do not need any base registers to address your program instructions, but you must use the STATREG and STATIC parameters to set up at least one static register to address static data in your program.

Examples

The following two example DFHEIENT statements generate the same code.

- In the first statement, all the parameters are coded (specifying the default values).

```
DFHEIENT CODEREG=0,DATAREG=13,EIBREG=11,STATREG=3,STATIC=STAT
```

- In the second statement, only the parameter that does not have a default value is coded.

```
DFHEIENT STATIC=STAT
```

Related information

- For reference information about the DFHEIENT macro, see DFHEIENT macro.
- For more information about the DFHEIRET macro, see “Coding the DFHEIRET macro for assembler language programs” on page 458 and DFHEIRET macro.

- For more information about the DFHEISTG storage, see “Extending dynamic storage for assembler language programs” on page 460.
- For more information about the DFHECALL and DFHEIRET macros, see DFHECALL macro.
- For reference information for the EXEC interface, including design overview, control blocks, modules, exits, and trace, see EXEC interface in Reference.

Coding the DFHEIRET macro for assembler language programs

The DFHEIRET macro performs epilog code to release the working storage of the application program.

- For AMODE(24) and AMODE(31) programs, the DFHEIRET macro calls the EPILOG program.
- For AMODE(64) programs, the DFHEIRET macro calls the AMODE(64) EPILOG program.

You can specify the following operand for the DFHEIRET macro:

RCREG
Specify the register into which the return code is placed. If you do not specify a value, a return code is not passed back to the caller of the program. There is no default value.

The translator inserts the DFHEIRET macro, with no parameters specified, immediately before the END statement (unless you specify the NOEPILOG translator option to prevent this). END must be in uppercase for the translator to recognize it. If the DFHEIRET macro is invoked before this translator-inserted DFHEIRET macro, the translator-inserted macro does not generate any code.

Related information

- For reference information about the DFHEIRET macro, see DFHEIRET macro.
- For more information about the DFHEIENT macro, see “Coding the DFHEIENT macro for AMODE(24) and AMODE(31) assembler language programs” on page 455, “Coding the DFHEIENT macro for AMODE(64) assembler language programs” on page 457 and DFHEIENT macro.
- For more information about the DFHEISTG storage, see “Extending dynamic storage for assembler language programs” on page 460.
- For more information about the DFHECALL and DFHEIRET macros, see DFHECALL macro.
- For reference information for the EXEC interface, including design overview, control blocks, modules, exits, and trace, see EXEC interface in Reference.

Calling assembler language programs

Assembler language application programs can be invoked by COBOL, C, C++, PL/I, or assembler language application programs by using LINK or XCTL commands.

64-bit addressing mode

AMODE (64) assembler language application programs that contain commands can have their own RDO program definition. Such programs can be invoked by COBOL, C, C++, PL/I, or assembler language application programs by using LINK or XCTL commands.

24-bit and 31-bit addressing mode

AMODE(24) and AMODE(31) assembler language application programs that contain commands can have their own RDO program definition. Such programs can be invoked by COBOL, C, C++, PL/I, or assembler language application programs by using LINK or XCTL commands. However, because AMODE(24) and AMODE(31) programs that contain commands are invoked by a system standard call, they can also be invoked by a COBOL, C, C++, or PL/I CALL statement, or by an assembler language CALL macro.

A single CICS application program, as defined in an RDO program definition, can consist of separate CSECTs compiled or assembled separately, but linked together.

An assembler language application program that contains commands can be linked with other assembler language programs, or with programs written in one or more high-level languages (COBOL, C, C++, or PL/I). For more information about mixing languages in an application load module, see “Mixing languages in Language Environment” on page 538 . For information about programs with different addressing modes, see “Using mixed addressing modes” on page 152.

If an assembler language program (that is link-edited separately) contains command-level calls, and is called from a high-level language program, it requires its own CICS interface stub. If the assembler program is link-edited with the high-level language program that calls it, then the assembler program does not need a stub. If you do provide one, the message MSGIEW024I is issued, but this can be ignored.

Because assembler language application programs containing commands are always passed the parameters EIB and COMMAREA when invoked, the CALL statement or macro must pass these two parameters, optionally followed by other parameters.

For example, the PL/I program in file PLITEST PLI calls the assembler language program ASMPROG, which is in file ASMTEST ASSEMBLE. The PL/I program passes three parameters to the assembler language program: the EIB, the COMMAREA, and a message string.

```
 PLIPROG:PROC OPTIONS(MAIN);
DCL ASMPROG ENTRY EXTERNAL;
DCL COMA CHAR(20), MSG CHAR(14) INIT('HELLO FROM PLI');
CALL ASMPROG(DFHEIBLK,COMA,MSG);
EXEC CICS RETURN;
END;
```

Figure 121. PLITEST PLI

The assembler language program performs an **EXEC CICS SEND TEXT** command, which displays the message string passed from the PL/I program.

```
DFHEISTG DSECT
MSG DS CL14
MYRESP DS F
ASMPROG CSECT
L 5,8(1)
L 5,0(5)
MVC MSG,0(5)
EXEC CICS SEND TEXT FROM(MSG) LENGTH(14) RESP(MYRESP)
END
```

Figure 122. ASMTEST ASSEMBLE

You can use JCL procedures supplied by CICS to compile and link the application, as follows:

1. Assemble and link ASMTEST using the DFHEITAL procedure:

   ```
   //ASMPROG EXEC DFHEITAL
   //TRN.SYSIN DD *
   .... program source ...
   /*
   //LKED.SYSIN DD *
   NAME ASMTEST(R)
   /*
   ```

2. Compile and link PLITEST using the DFHYITPL procedure, and provide linkage editor control statements that include the ASMTEST load module created by the DFHEITAL procedure:

   ```
   //PLIPROG EXEC DFHYITPL
   //TRN.SYSIN DD *
   .... program source ...
   /*
   //LKED.SYSIN DD *
   INCLUDE SYSLIB(ASMTEST)
   ENTRY CEESTART
   NAME PLITEST(R)
   /*
   ```

   Note: Step 2 assumes that the ASMTEST load module created by DFHEITAL is stored in a library that is included in the SYSLIB data set concatenation.

The load module created by the DFHYITPL procedure includes both the DFHEAI stub (included by DFHEITAL) and the DFHELII stub (included by DFHYITPL). The linkage editor or binder program issues a warning message because both stubs contain an entry point named DFHEII. This message can be ignored.

For information about including DFHEAI, see DFHEAI.

If you write your own JCL, you must include the DFHELII stub, because this contains the entry points that are needed for all languages.

An assembler language application program can begin with the DFHEIENT macro and end with the DFHEIRET macro. The CICS translator inserts these for you, so if the program contains EXEC CICS commands and is passed to the translator, as in the example just given, you do not need to code these macros. Note that DFHEIRET accommodates relative addressing. If there is no base register, the macro expansion no longer generates an LTORG. Normally, when the END is reached, the assembler generates an automatic LTORG to gather up remaining VCONs. If you have code that results in a private code CSECT at the start of the CSECT concatenation, this can cause issues. When the END statement is reached, the assembler reverts to your first CSECT and cannot resolve the VCON from it. It is recommended that you check assembler listings and eliminate this kind of private control section. For details, see Unnamed section information in the HLASM Language Reference.

Extending dynamic storage for assembler language programs

You can extend dynamic storage to provide extra storage for user variables by defining these variables in your source program in a DSECT called DFHEISTG.

24-bit and 31-bit addressing mode

For AMODE(24) and AMODE(31) programs, the maximum amount of dynamic storage obtainable using the DFHEISTG DSECT is 65 264 bytes. DFHEISTG is a reserved name. This storage is initialized to X'00'. At translation, the translator inserts the DFHEISTG macro immediately following your DFHEISTG DSECT instruction. In this way, the DSECT describes dynamic storage needed for the parameter list, for the command-level interface, and for any user variables. At link-edit time, use the STORAGE option of the CEEXOPT macro to ensure that the DFHEISTG storage is initialized to x'00', for example, CEEXOPT STORAGE=(,,00). Make sure that your application propagates or initializes any constants that are defined in the user DFHEISTG area.

The example in Figure 123 on page 460 shows a simple assembler language application program that uses such variables in dynamic storage.

```
Source program

INSTRUCT CSECT
         EXEC CICS SEND MAP('DFH$AGA') MAPONLY ERASE
         END


This source program is translated to:

         DFHEIGBL ,                INSERTED BY TRANSLATOR
INSTRUCT CSECT
         DFHEIENT                  INSERTED BY TRANSLATOR
*        EXEC CICS SEND MAP('DFH$AGA') MAPONLY ERASE
         DFHECALL =X'1804C000080000000046204000020',
               (CHA7,=CL7'DFH$AGA*'),(______RF,DFHEIV00)
         DFHEIRET                  INSERTED BY TRANSLATOR
         DFHEISTG                  INSERTED BY TRANSLATOR
         DFHEIEND                  INSERTED BY TRANSLATOR
         END
```

Figure 123. Translated code for user variables

64-bit addressing mode

For non-Language Environment AMODE(64) assembler language programs, the DFHEISTG macro generates an AMODE(64) DSECT. DFHEISTG storage is obtained from 31-bit storage (above 16 MB but below 2 GB), not from 64-bit storage. The maximum amount of dynamic storage obtainable using the DFHEISTG DSECT is 65 264 bytes. This storage is initialized to X'00'.

At translation, the translator inserts the DFHEISTG macro immediately following your DFHEISTG DSECT instruction. In this way, the DSECT describes dynamic storage needed for the parameter list, for the command-level interface, and for any user variables.

CICS defines the front portion of DFHEISTG storage.

The EIB pointer, DFHEIBP, and the COMMAREA pointer, DFHEICAP, are set up before the DFHEIENT macro returns control to the application program. These pointers are 64-bit pointers in 24-bit or 31-bit storage.

Developing AMODE(64) assembler language programs

CICS Transaction Server supports non-Language Environment (LE) assembler language programs that run in 64-bit addressing mode (AMODE(64)) and thereby access up to 16 exabytes (EB) of virtual storage. This support allows you to develop non-LE assembler language programs, which are able to process large amounts of data.

Because CICS also allows linkage between AMODE(64) and AMODE(31) or even AMODE(24) programs, it is possible to provide service routines that handle data in 64-bit storage without rewriting other parts of the application.

CICS AMODE(64) native assembler application programming

Writing a CICS AMODE(64) native assembler application program is very similar to writing a CICS AMODE(31) native assembler application program. The same CICS macros that are included in the AMODE(31) version of the application program are included in an AMODE(64) application program.

There are a number of differences between writing in AMODE(64) and writing in AMODE(31), as follows:

- The z/OS SYSSTATE macro must specify AMODE64=YES. The CICS macros DFHEISTG, DFHEIEND, DFHEIENT, DFHEIRET and DFHECALL are SYSSTATE aware and generate AMODE(64) assembler when SYSSTATE AMODE64=YES has been specified. The SYSSTATE macro must be invoked before any of the SYSSTATE aware CICS macros are invoked.
- Macro DFHEIENT must use relative addressing. To specify that relative addressing is to be used, CODEREG=0 must be specified and STATREG and STATIC values must be supplied.
- The AMODE(64) version of the CICS stub must be included in the Binder (Link-edit) step in a similar way that the AMODE(31) CICS stub must be included in the Binder step for an AMODE(31) or AMODE(24) application. The AMODE(64) CICS stub is DFHEAG.

When writing AMODE(64) assembler, whether for CICS or batch, you must ensure that the registers used to reference storage have valid addresses. There are assembler instructions that ensure that a clean 31-bit address is loaded into a 64-bit register. These instructions are LLGT and LLGTR. LLGT loads a 31-bit address into a registers and clears the top 33 bits of the registers. LLGTR loads a register from another register and clears the top 33 bits. Instruction LLGF can be used to load a fullword into a 64-bit register; the fullword is loaded into the bottom 32 bits of the register and the top 32 bits of the registers are cleared.

EXEC CICS 64-bit commands

To allow AMODE(64) programs to use above-the-bar storage, CICS provides 64-bit variants of some of the CICS API commands to manipulate data in 64-bit storage (also referred to *as above the bar*). These commands are as follows:

GETMAIN64
To get storage above the bar (by default: other storage locations can be explicitly specified)

FREEMAIN64
To release storage that was acquired by a program running in AMODE(64), typically used to free storage above the bar

PUT64 CONTAINER
To put data from 64-bit storage into a named channel container

GET64 CONTAINER
To get data from a named channel container into 64-bit storage

EXEC CICS assembler language macros

If the application is declared as AMODE(64), the following CICS-supplied macros generate AMODE(64) code:

DFHEIENT
For AMODE(64) programs, the DFHEIENT macro calls the AMODE(64) PROLOG program, which allocates working storage to hold any user variables and for CICS use. The PROLOG program sets up the CICS portion of this storage. When the PROLOG program returns, this code sets up the registers specified by the DFHEIENT parameters. For more information, see “Coding the DFHEIENT macro for AMODE(64) assembler language programs” on page 457.

DFHEISTG
For AMODE(64) programs, the DFHEISTG macro generates an AMODE(64) DSECT. At translation, the translator inserts the DFHEISTG macro immediately following your DFHEISTG DSECT instruction. In this way, the DSECT describes dynamic storage needed for the parameter list, for the command-level interface, and for any user variables. For more information, see “Extending dynamic storage for assembler language programs” on page 460.

DFHEIRET
For AMODE(64) programs, the DFHEIRET macro calls the AMODE(64) EPILOG program to release the working storage of the application program. For more information, see “Coding the DFHEIRET macro for assembler language programs” on page 458.

DFHECALL
For an assembler language application program, when the CICS translator detects a CICS command, each command is replaced by an invocation of the DFHECALL macro. The DFHECALL macro sets up the command parameters and calls the initial CICS command processor to handle the command. For more information, see DFHECALL macro.

Procedure

The following procedure summarizes the information that relates specifically to developing AMODE(64) assembler language programs.

Note: Check the relevant section of “Assembler language programming restrictions and requirements” on page 449.

1. Use the z/OS Assembler SYSSTATE macro to declare that an application is AMODE(64).
   a. Include a SYSLIB statement for the SYS1.MACLIB macro library (which contains the SYSSTATE macro) in your compile JCL.
   b. Ensure one of the following:
      - The SYSSTATE macro with parameter AMODE64=YES is placed immediately after the CICS translator options statement. For example:

```
*ASM   XOPTS(NOPROLOG NOEPILOG)
       SYSSTATE AMODE64=YES
```

      - If the program does not specify CICS translator options, the SYSSTATE macro with parameter AMODE64=YES is the first statement.

The SYSSTATE statement must be on one line, because CICS does not support continuation for this statement. For more information about the SYSSTATE macro, see SYSSTATE - Identify system state in z/OS MVS Programming: Assembler Services Reference IAR-XCT.

Note: If the SYSSTATE macro is not specified, or is specified without parameter AMODE64=YES, when any of the CICS-supplied macros, DFHEIENT, DFHEISTG, DFHEIRET, and DFHECALL, first generates code, these macros generate AMODE(24) or AMODE(31) code.

2. Specify the DFHEIENT macro with the appropriate parameters for relative addressing. See "Coding the DFHEIENT macro for AMODE(64) assembler language programs" on page 457.

   For information about the DFHEISTG storage, see "Extending dynamic storage for assembler language programs" on page 460.
3. If you are not using **EXEC CICS RETURN**, specify the DFHEIRET macro, or use the translator-inserted version. See "Coding the DFHEIRET macro for assembler language programs" on page 458.
4. Translate, assemble and link-edit the program. See "Translating, assembling, and link-editing assembler language application programs" on page 692.

   For information about the DFHECALL macro, see DFHECALL macro.

   For information about link-editing the stub programs, see "Using the EXEC interface modules for AMODE(64) applications" on page 620.

CICS AMODE(64) assembler example

Here is an example of a CICS AMODE(64) native assembler application program. The source code has been augmented with annotations. There are many styles in which assembler code can be written, and no one is better than any other, it is just a matter of the writer's preference. This example illustrates a possible style for writing AMODE(64) assembler in CICS.

The example shows how to obtain shared storage located above the bar and save the address of the shared storage address in the CWA. The example does not show how data is loaded into the shared storage, as there are many ways in which this could be done. For example, you might want to read data from a VSAM file or from a Db2 database.

The data stored in the shared storage above the bar in this example cannot be accessed by AMODE(31) or AMODE(24) applications, but an AMODE(64) assembler program called by the AMODE(31) or AMODE(24) application would be able to access this data and then return all, or portions of, the data to its caller.

There are several ways in which the data could be returned to the AMODE(31) or AMODE(24) caller, including:

- CICS COMMAREA
- CICS container, within a channel
- A buffer supplied by the caller

The *ASM XOPTS statement is used to pass options to the translator. In this case NOPROLOG and NOEPILOG are specified to stop the translator inserting PROLOG and EPILOG instructions.

*ASM XOPTS(NOPROLOG NOEPILOG) Translator options

The SYSSTATE macro tells SYSSTATE aware macros that the AMODE is 64 and also specifies the architecture level of the machine that the program will execute on. The CICS native assembler macros are SYSSTATE aware so will generate AMODE 64 code when AMODE64=YES is set.

```
         SYSSTATE AMODE64=YES,ARCHLVL=4   Sysstate AMODE 64, Z10
*********************************************************************
*        Register equates.
*********************************************************************
r0       equ   0
r1       equ   1
r2       equ   2
r3       equ   3
r4       equ   4
r5       equ   5
```

```
r6       equ   6
r7       equ   7
r8       equ   8
r9       equ   9
r10      equ   10
r11      equ   11
r12      equ   12
r13      equ   13
r14      equ   14
r15      equ   15
*********************************************************************
*        Module's working storage.
*********************************************************************
```

Macros DFHEISTG and DFHEIEND are SYSSTATE aware. These macros define DSECT DFHEISTG. As the SYSSTATE macro specified AMODE64=YES the CICS portion of this DSECT is defined for an AMODE 64 application.

The CICS portion of the DSECT defines a format-four save area, the 64-bit address of the COMMAREA and EIB, and other fields used by CICS when the application program issues a CICS command.

The application's working storage variables are defined after the DFHEISTG macro and before the DFHEIEND macro. In this case these variables are wk_shared_storage_addr, wk_cwa_address and wk_flength.

The DFHEIEND indicates the end of the DSECT.

```
         dfheistg ,
         ds    0ad
wk_shared_storage_addr   ds  ad      Shared storage address
wk_cwa_addr              ds  ad      CWA address
wk_flength               ds  f       Getmain64 flength
         dfheiend
The next two DSECTs (cwa_mapping and shared_data) map the CWA and the shared data storage area.
*********************************************************************
*        CWA dsect.
*********************************************************************
cwa_mapping  dsect
cwa_shared_storage_addr  ds  ad      Shared storage address
*********************************************************************
*        Shared data dsect.
*********************************************************************
shared_data  dsect
shared_thing ds   xl1000             Shared data object
shared_data_length equ *-shared_data Length of shared data
```

The following statements name the CSECT and its AMODE and RMODE.

```
AMODE64X csect
AMODE64X amode 64
AMODE64X rmode 31
```

Macro DFHEIENT generates the program's entry code. This code:

- Saves the caller's registers in the format-four save area pointed to be R13.
- Makes a call to DFHEAG0 is obtain the program's working storage. This storage must be obtained below the bar to allow all CICS command processors to function correctly.
- Sets up the working storage, static and EIB registers. The working storage register is R13, the static register is R12 and the EIB register is R11.

```
         DFHEIENT CODEREG=0,           Relative addressing            -
               DATAREG=R13,            Data register                  -
               EIBREG=R11,             EIB register                   -
               STATREG=R12,            Static register                -
               STATIC=STATIC_AREA      Static label
*********************************************************************
*        Obtain above the bar shared storage.
*********************************************************************
```

The CICS command below obtains above the bar shared storage. The address of the shared storage obtained is returned in R2 if the command is successful.

```
         EXEC CICS GETMAIN64                                          -
               FLENGTH(LENGTH_OF_SHARED_STORAGE)                      -
               SET(R2)                                                -
               SHARED                                                 -
               NOHANDLE
```

The code below checks the EIB response and if it is not normal jumps to label abend1.

```
         clc  eibresp,dfhresp(normal)
         jne  abend1
```

If the response to the command is normal the address of the shared storage is saved in the program's working storage.

```
         stg  r2,wk_shared_storage_addr  Save shared storage address
```

Insert code here to load data into the shared storage. Data could be read from many sources including VSAM files or databases.

```
*********************************************************************
*        Obtain address of CWA.
*********************************************************************
```

The CICS command below returns the address of the CWA in R3 if the command is successful.

```
         EXEC CICS ADDRESS CWA(R3) NOHANDLE
```

The code below checks the EIB response and if it is not normal jumps to label abend2.

```
         clc  eibresp,dfhresp(normal)
         jne  abend2
```

If the response to the command is normal, the address of the CWA is saved in the program's working storage.

```
         stg  r3,wk_cwa_addr             Save CWA address
*********************************************************************
*        Save shared storage address in CWA.
*********************************************************************
```

The code below uses the CWA mapping to save the address of the shared data area in the CWA.

```
         using cwa_mapping,r3
         stg  r2,cwa_shared_storage_addr Save shared storage addr
         drop r3                         Drop r3
*********************************************************************
*        Return to CICS.
*********************************************************************
return   ds   0h
```

The command below returns control to CICS. In fact the DFHEIRET macro could have been used instead the of the CICS return command.

```
         EXEC CICS RETURN
*********************************************************************
```

```
*        Issue abend.
*********************************************************************
```

The CICS command below issues abend ERR1 if the CICS GETMAIN command is not successful.

```
Abend1   ds   0h
         EXEC CICS ABEND ABCODE('ERR1')
The CICS command below issues abend ERR2 if the CICS ADDRESS command is not successful.
Abend2   ds   0h
         EXEC CICS ABEND ABCODE('ERR2')
*********************************************************************
*        Static
*********************************************************************
```

The program's static area

```
static_area   ds  0fd
length_of_shared_storage  dc  al4(shared_data_length)
*********************************************************************
*        Ltorg
*********************************************************************
         ltorg
         end
```

Using AMODE(64) native assembler programs

An AMODE(64) program such as the example shown above could be used to store large amounts of data, and return portions of this to other programs within an application. In this scenario, the AMODE(64) program could act as a service routine which is called by programs in other addressing modes to manipulate large storage areas on their behalf.

One particularly valuable way of using an AMODE(64) program could be to put data from a 64-bit storage area directly into a container within a channel (either a named channel or the default current channel). This data in this container can then be accessible to an AMODE(31), or even AMODE(24), program by using the container API. The **EXEC CICS GET CONTAINER** command will return the 64-bit data into a 31-bit (or 24-bit) storage area provided by the invoking program. This can be a useful way of sharing data between 64-bit 'service' routines and other programs in the application, without the need to convert all of the application to AMODE(64) native assembler.

To invoke an AMODE(64) program from a program in another addressing mode, use **EXEC CICS LINK** or XCTL.

Chapter 7. Developing C and C++ applications

Use this information to help you code, translate, and compile C and C++ programs that you want to use as CICS application programs.

Changes to CICS support for application programming languages lists the C and C++ compilers that are supported by CICS Transaction Server for z/OS, Version 5 Release 6 , and their service status on z/OS. All references to C and C++ in CICS Transaction Server for z/OS, Version 5 Release 6 documentation imply the use of a supported Language Environment-conforming compiler, unless stated otherwise. All **EXEC CICS** commands available in COBOL, PL/I, and assembler language applications are also supported in C and C++ applications, except those commands related to nonstructured exception handling.

C++ applications can also use the CICS C++ OO classes to access CICS services, instead of the **EXEC CICS** interface. See Using the CICS foundation classes for more information about this interface. C++ supports object-oriented programming and you can use this language in the same way as you would use the C language. You must specify that the translator is to translate C++ using the CPP option. C++ programs must also be defined with the LANGUAGE(LE370) option.

Working storage

In C and C++, working storage consists of the stack and the heap. The location of the stack and heap, with respect to the 16 MB line, is controlled by the ANYWHERE and BELOW options on the stack and heap runtime options. The default is that both the stack and heap are located above the 16 MB line.

Sample programs

A set of sample application programs is provided to show how **EXEC CICS** commands can be used in a program written in the C or C++ language.

Table 54. Sample programs

Sample program	Map set	Map source	Transaction ID
DFH$DMNU **Operator instruction (3270)**	DFH$DGA	DFH$DMA	DMNU
DFH$DALL **Update (3270)**	DFH$DGB	DFH$DMB	DINQ, DADD, DUPD
DFH$DBRW **Browse (3270)**	DFH$DGC	DFH$DMC	DBRW
DFH$DREN **Order entry (3270)**	DFH$DGK	DFH$DMK	DORD
DFH$DCOM **Order entry queue print (3270)**	DFH$DGL	DFH$DML	DORQ
DFH$DREP **Report (3270)**	DFH$DGD	DFH$DMD	DREP

The transaction and program definitions are provided in group DFH$DFLA in the CICS system definition data set (CSD) and can be installed using the command:

```
CEDA INSTALL GROUP(DFH$DFLA)
```

The following record description files are provided as C or C++ language header files:

- DFH$DFIL: FILEA record descriptor
- DFH$DL86: L860 record descriptor

FLOAT compiler option

For z/OS V1.11 XL C (or C++) or later, specify either the FLOAT(NOAFP) compiler option, or the FLOAT(AFP(VOLATILE)) compiler option.

© Copyright IBM Corp. 1974, 2023

- If your program makes little or no use of floating point, specify the FLOAT(NOAFP) option. The program uses only the traditional four floating point registers, and has less work to do when saving registers.
- If your program makes significant use of floating point, specify the FLOAT(AFP) option or the FLOAT(NOVOLATILE) option. The program can use all 16 floating point registers and CICS preserves the floating point registers used by the program.
- If you specify the FLOAT(AFP(VOLATILE)) option, CICS, C, and C++ preserve the floating point registers. Extra code is generated and can therefore degrade performance.

C and C++ programming restrictions and requirements

Some restrictions and requirements apply to a C or C++ program that is used as a CICS application program.

Functions and commands that cannot be used

The following EXEC CICS commands, which are related to nonstructured exception handling, are not supported for C and C++ applications:

- **HANDLE ABEND LABEL(label)**
- **HANDLE AID**
- **HANDLE CONDITION**
- **IGNORE CONDITION**
- **PUSH HANDLE**
- **POP HANDLE**

The translator diagnoses use of these commands. **HANDLE ABEND PROGRAM** commands are allowed.

CICS does not support the system() function, but two CICS commands, **LINK** and **XCTL** , provide equivalent function.

CICS does not support extended precision floating point.

C++ does not support packed decimal data. The application has access to packed decimal data using the character string data type. No C++ standard library functions are available to perform arithmetic on this data, but you can write your own. When using CICS commands that have options to specify time (for example, the **DELAY** or **POST** commands), it is advisable to use the HOURS, MINUTES, and SECONDS options. You can define times by using the TIME or INTERVAL options, which are packed decimal data types, if you provide functions to handle them in your application.

C and C++ do not support the use of CICS commands in macros.

Native C or C++ file operations operate only on files that are opened with *type=memory* specified. I/O to CICS-supported access methods must use the CICS API.

All native C and C++ functions are allowed in the source program, but the following functions are not recommended. Some are not executable and result in return codes or pointers indicating that the function has failed. Some might work but can impact the performance or execution of CICS.

- CDUMP
- CSNAP
- CTEST
- CTRACE
- CLOCK (The clock() function returns a value (time_t) of -1.)
- CTDLI
- SVC99
- SYSTEM
- SETLOCALE

Coding requirements

- You can enter all CICS keywords in mixed case, except for CICS keywords on #pragma directives, which must be in upper case only.
- Where CICS expects a fixed-length character string such as a program name, map name, or queue name, you must pad the literal with blanks up to the required length if it is shorter than expected. For EXEC DLI commands, the SEGMENT name is padded by the translator if a literal is passed.
- Do not use field names that might be acceptable to the assembler, but that cause the C or C++ compiler to abend. These names include $, #, and @.
- C++ uses '//' for single line comments. Do not put such a comment in the middle of an EXEC CICS command. For example, the following code is not valid:

```
EXEC CICS SEND TEXT FROM(errmsg)
LENGTH(msglen) // Send error message to screen
RESP(rcode)
RESP2(rcode2);
```

  The following code examples are valid:

```
EXEC CICS SEND TEXT FROM(errmsg)
LENGTH(msglen)
RESP(rcode)
RESP2(rcode2); //Send error message to screen
```

```
EXEC CICS SEND TEXT FROM(errmsg)
LENGTH(msglen) /* Send error message to screen */
RESP(rcode)
RESP2(rcode2);
```

Condition handling

In a C or C++ application, every EXEC CICS command is treated as if the NOHANDLE or RESP option is specified. Therefore, the set of "system action" transaction abends that result from a condition occurring, but not being handled, is not possible. Control always flows to the next instruction, and the application is responsible for testing for a normal response.

COMMAREA

The address of the communication area is not passed as an argument to a C or C++ main function. This means that C and C++ functions must use ADDRESS COMMAREA to obtain the address of the communications area.

EIB

The address of the EXEC interface block (EIB) is not passed as an argument to a C or C++ main function. This means that C and C++ functions must use ADDRESS EIB to obtain the address of the EIB. See “Accessing the EIB from C and C++” on page 472 for more information.

LENGTH

If you do not specify the LENGTH option on commands that support LENGTH (for example, **READ** , **READNEXT** , **READPREV** , and **WRITE** commands), the translator does not supply a default value. In effect, NOLENGTH is implicit for C programs.

OVERFLOW conditions

If you want any OVERFLOW condition to be indicated in the RESP field on return from a **SEND MAP** command with the ACCUM option, you should specify the NOFLUSH option.

Addressing mode

All C and C++ language programs running under CICS must be link-edited with the attributes AMODE(31), RMODE(ANY). They can reside above the 16 MB line.

Consequently, when you pass parameters to a program that is produced by the Cross-System Product (CSP) interactive application generator, you must do one of the following:

- Pass parameters below 16 MB
- Re-link the CSP load library with AMODE(31)

64-bit addressing mode (AMODE(64)) is not supported for C and C++ language programs.

64-bit residency mode

CICS does not support 64-bit residency mode (RMODE(64)) and treats any RMODE(64) programs as RMODE(31). That is, RMODE(64) programs are loaded into 31-bit (above-the-line) storage, not 64-bit (above-the-bar) storage.

Return value

If you terminate a C or C++ program with an exit() function or the return statement instead of a CICS **RETURN** command, the value passed through the exit() function is saved in the EIBRESP2 field of the EIB on return from the program.

Note: If a program uses DPL to link to a program in another CICS region, EIBRESP2 values from the remote region are **not** returned to the program doing the DPL.

Data declarations

The following data declarations are provided by CICS for C and C++:

- Execution interface block definitions (EIB). The EIB declarations are enclosed in #ifndef and #endif lines, and are included in all translated files. The C or C++ compiler ignores duplicated declarations. The inserted code contains definitions of all the fields in the EIB, coded in C and C++.
- BMS screen attributes definitions: C and C++ versions of the DFHBMSCA, DFHMSRCA, and DFHAID files are supplied by CICS, and can be included by the application programmer when using BMS.
- DL/I support: a C language version of DFHDIB is included by the DLI translator if the translator option has been specified. (You must include DLIUIB if the CALL DLI interface is used.)

Fetch function

Language Environment -conforming programs support the fetch() and release() functions. Modules to be fetched must be defined as PROGRAM resources to CICS, either explicitly or implicitly through autoinstall.

Locale functions

All locale functions are supported for locales that have been defined in the CICS system definition data set (CSD). The setlocale() function returns NULL if the locale is not defined.

Debugging functions

The dump functions csnap(), cdump(), and ctrace() are supported. The output is sent to the CESE transient data queue. The dump cannot be written if the queue does not have a sufficient record length (LRECL). An LRECL of at least 161 is recommended.

iscics function

The iscics() function can be useful if you are adapting an existing program, or writing a new program, that is designed to run outside CICS as well as under CICS. The function returns a non-zero value if your program is currently running under CICS, or zero if it is not. This function is an extension to the C library.

String handling functions

The string handling functions in the C or C++ standard library use a null character as an end-of-string marker. Because CICS does not recognize a null as an end-of-string marker, you must take care when using C or C++ functions, for example strcmp, to operate on CICS data areas.

argc and argv arguments

Two arguments, argc and argv, are normally passed to a C or C++ main function. argc denotes how many variables have been passed; argv is an array of zero-terminated variable strings. In CICS, the value of argc is 1, argv[0] is the transaction ID, and argv[1] is NULL.

Passing arguments in C and C++

Arguments in C and C++ language are copied to the program stack at run time, where they are read by the function. These arguments can either be values in their own right, or they can be pointers to areas of memory that contain the data being passed. Passing a pointer is also known as passing a value **by reference**.

Other languages, such as COBOL and PL/I, usually pass their arguments by reference, which means that the compiler passes a list of addresses pointing to the arguments to be passed. This is the call interface supported by CICS. To pass an argument by reference, you prefix the variable name with **&** , unless it is already a pointer, as in the case when an array is being passed.

As part of the build process, the compiler may convert arguments from one data type to another. For example, an argument of type **char** may be converted to type **short** or type **long**.

When you send values from a C or C++ program to CICS, the translator takes the necessary action to generate code that results in an argument list of the correct format being passed to CICS. The translator does not always have enough information to enable it to do this, but in general, if the argument is a single-character or halfword variable, the translator makes a precall assignment to a variable of the correct data type and passes the address of this temporary variable in the call.

When you receive data from CICS, the translator prefixes the receiving variable name with **&** , which causes the C or C++ compiler to pass it values *by reference* rather than *by value* (with the exception of a character string name, which is left unchanged). Without the addition of **&** , the compiler would copy the receiving variable and then pass the address of the copy to CICS. Any promotion occurring during this copying could result in data returned by CICS being lost.

Table 55 on page 471 shows the rules that apply when passing values as arguments in EXEC CICS commands.

Table 55. Rules for passing values as arguments in EXEC CICS commands

Data type	Usage	Coding the argument
Character literal	Data-value (Sender)	The user must specify the character literal directly. The translator takes care of any required indirection.
Character variable (char)	Data-area (Receiver)	The user must specify a pointer to the variable, possibly by prefixing the variable name with **&**.
Character variable (char)	Data-value (Sender)	The user must specify the character variable directly. The translator takes care of any required indirection.
Character string literal	Name (Sender)	The user can either code the string directly as a literal string or use a pointer which points to the first character of the string.
Character string variable	Data-area (Receiver) Name (Sender)	Whether receiving or sending, the argument should be the name of the character array containing the string—the address of the first element of the array.
Integer variable (short, long, or int)	Data-area (Receiver)	The user must specify a pointer to the variable, possibly by prefixing the variable name with **&**.

Table 55. Rules for passing values as arguments in EXEC CICS commands (continued)		
Data type	**Usage**	**Coding the argument**
Integer variable (short, long, or int)	Data-value (Sender)	The user must specify the name of the variable. The translator looks after any indirection that is required.
Integer constant (short, long, or int)	Data-value (Sender)	The user must specify the integer constant directly. The translator takes care of any required indirection.
Structure or union	Data-area (Sender) Data-area (Receiver)	The user must code the address of the start of the structure or union, possibly by prefixing its name with **&**.
Array (of anything)	Data-area (Receiver) Data-value (Sender)	The translator does nothing. You must code the address of the first member of the array. This is normally done by coding the name of the array, which the compiler interprets as the address of the first member.
Pointer (to anything)	Ptr-ref (Receiver) Data-area (Sender)	Whether receiving or sending, the argument should be the name of the variable that denotes the address of interest. The translator takes care of the extra level of indirection that is necessary to allow CICS to update the pointer.

Note: Receiver is where data is being received from CICS; Sender is where data is being passed to CICS.

Accessing the EIB from C and C++

The address of the EXEC interface block (EIB) is not passed as an argument to a C or C++ main function. This means that C and C++ functions must use the ADDRESS EIB command to obtain the address of the EIB.

You must code an ADDRESS EIB statement at the beginning of each application if you want access to the EIB, or if you are using a command that includes the RESP or RESP2 option.

Addressability is achieved by using the command:

```
EXEC CICS ADDRESS EIB(dfheiptr);
```

or by passing the EIB address or particular fields therein as arguments to the CALL statement that invokes the external procedure.

If access to the EIB is required, an ADDRESS EIB command is required at the beginning of each program.

Within a C or C++ application program, fields in the EIB are referred to in lower case and fully qualified as, for example, “dfheiptr->eibtrnid”.

The following mapping of data types is used:

- Halfword binary integers are defined as “short int”
- Fullword binary integers are defined as “long int”
- Single-character fields are defined as “unsigned char”
- Character strings are defined as “unsigned char” arrays

Locale support for C and C++

The CICS translator, by default, assumes that programs written in the C or C++ language have been edited with the EBCDIC Latin-1 code page IBM -1047.

If you have used an alternative code page, you can specify this in a pragma filetag directive at the start of the application program. The pragma statement must be the first non-comment statement in the program, and the filetag directive must be specified before any other directive in the pragma statement. The CICS translator scans for the presence of the filetag directive. The CICS translator only supports the default code page IBM-1047, the Danish EBCDIC code page IBM-277, the German EBCDIC code page IBM-273, and the Chinese EBCDIC code pages IBM-935 and IBM-1388.

For example, if the program has been prepared with an editor using the German EBCDIC code page, it should begin with the following directive:

```
??=pragma filetag ("IBM-273")
```

If your application program uses a mix of different code pages (for example, if you are including header files edited in a code page different to that used for the ordinary source files), all of the files must include the pragma filetag directive, even if they are in the default code page IBM-1047.

Some older IBM C compilers which are no longer in service, but can still be used with the CICS translator, might not support the use of the pragma filetag directive. Check the documentation for your compiler if you are not sure whether your compiler supports this.

XPLink and C and C++ programming

CICS provides support for C and C++ programs compiled with the XPLINK option. All programs using CICS XPLink support must be reentrant and threadsafe.

Extra Performance Linkage, normally abbreviated to XPLink, is a z/OS feature which provides high performance subroutine call and return mechanisms. This results in short and highly optimized execution path lengths.

Object Oriented programming is built upon the concept of sending 'messages' to objects which result in that object performing some actions. The message sending activity is implemented as a subroutine invocation. Subroutines, known as member functions in C++ terminology, are normally small pieces of code. The characteristic execution flow of a typical C++ program is of many subroutine invocations to small pieces of code. Programs of this nature benefit from the XPLink optimization technology.

MVS has a standard subroutine calling convention which can be traced back to the early days of System/360. This convention was optimized for an environment in which subroutines were more complex, there were relatively few of them, and they were invoked relatively infrequently. Object oriented programming conventions have changed this. Subroutines have become simpler but they are numerous, and the frequency of subroutine invocations has increased by orders of magnitude. This change in the size, numbers, and usage pattern, of subroutines made it desirable that the system overhead involved be optimized. XPLink is the result of this optimization.

To use XPLink, your C or C++ application code must be **reentrant** and **threadsafe**. The same code instance can be executing on more than one MVS TCB and, without threadsafe mechanisms to protect shared resources, the execution behavior of application code is unpredictable. This cannot be too strongly emphasized.

If you plan to compile C and C++ programs for the CICS environment with the XPLINK option, the application developer is expected to do the following to take advantage of CICS XPLink support:

- Develop the code, strictly adhering to threadsafe programming principles and techniques.
- Compile the C or C++ program with the XPLINK option set on.
- Indicate in the PROGRAM resource definition that the program is threadsafe.
- Consider the use of CICSVAR in CEEUOPT or in #pragma (see the note in " Defining runtime options for Language Environment " on page 541 for details).

All programs using CICS XPLink support must be reentrant and threadsafe. Only the application developer can guarantee that the code for a particular application satisfies these requirements.

XPLink uses X8 and X9 mode TCBs

CICS provides support for C and C++ programs compiled with the XPLINK option by using the multiple TCB feature in the CICS Open Transaction Environment (OTE) technology. X8 and X9 mode TCBs are defined to support XPLink tasks in CICS key and USER key. Each instance of an XPLink program uses one X8 or X9 TCB.

CICS support for programs compiled with the XPLINK option requires only that you show in the PROGRAM resource definition that the program is threadsafe. This indication, and the XPLink “signature” in the load module, are the only things required to put the task on an X8 or X9 TCB.

In the selection of a suitable TCB for a particular program, XPLink takes precedence over the existence of the OPENAPI value for the API attribute on the PROGRAM resource definition.

Passing control between XPLink and non-XPLink objects

Each transfer of control from XPLink objects to non-XPLink objects, or the reverse, causes a switch between the QR TCB and an open TCB (either an X8 or an X9 TCB). In performance terms, TCB switching is costly, and you must take this performance overhead into account.

An XPLink object can invoke a non-XPLink object using either the EXEC CICS interface or the Language Environment interface.

A non-XPLink object can only invoke an XPLink object using the EXEC CICS interface. Use of the Language Environment interface for such invocations is not supported.

Global user exits and XPLink

The XPCFTCH and XPCTA exits are affected by the use of the XPLINK option. CICS disregards any attempt by XPCFTCH to modify the entry point, and any attempt by XPCTA to define a resume address. Other global user exits are unaffected by XPLink support.

XPCFTCH
When the exit XPCFTCH is invoked for a C or C++ program that was compiled with the XPLINK option, a flag is set indicating that any modified entry point address, if specified by the exit, will be ignored.

XPCTA
When the exit XPCTA is invoked for a C or C++ program that was compiled with the XPLINK option, a flag is set indicating that a resume address, if specified by the exit, will be ignored.

These activities are ignored because the batch Language Environment runtime used for XPLink programs does not give control to CICS when a program abends, but goes through its own abend handling. When control reaches CICS, the Language Environment enclave has terminated, so CICS is unable to honor an entry point address or a resume address.

If you have an application program that carries out these activities, you must find other ways to manage these requirements, or conclude that the program is not a suitable candidate for XPLINK optimization. One possible solution is to write a Language Environment abnormal termination exit, as described in the information about customizing user exits in z/OS Language Environment Customization.

Using the CICS foundation classes

This section describes the CICS foundation classes and how to use them. There is a formal listing of the user interface in Foundation Classes: reference.

C++ Objects

This section describes how to create, use, and delete objects.

This section describes how to create, use, and delete objects. In our context an object is an instance of a class. An object cannot be an instance of a base or abstract base class. It is possible to create objects of all the concrete (non-base) classes described in the reference part of this book.

Creating an object

If a class has a constructor it is executed when an object of that class is created. This constructor typically initializes the state of the object. Foundation Classes' constructors often have mandatory positional parameters that the programmer must provide at object creation time.

C++ objects can be created in one of two ways:

1. Automatically, where the object is created on the C++ stack. For example:

   ```
   {
   ClassX objX
   ClassY objY(parameter1);
   } //objects deleted here
   ```

   Here, objX and objY are automatically created on the stack. Their lifetime is limited by the context in which they were created; when they go out of scope they are automatically deleted (that is, their destructors run and their storage is released).
2. Dynamically, where the object is created on the C++ heap. For example:

   ```
   {
   ClassX* pObjX = new ClassX;
   ClassY* pObjY = new ClassY(parameter1);
   } //objects NOT deleted here
   ```

   Here we deal with pointers to objects instead of the objects themselves. The lifetime of the object outlives the scope in which it was created. In the previous sample the pointers (pObjX and pObjY) are 'lost' as they go out of scope but the objects they pointed to still exist! The objects exist until they are explicitly deleted as shown here:

   ```
   {
   ClassX* pObjX = new ClassX;
   ClassY* pObjY = new ClassY(parameter1);
   :
   pObjX->method1();
   pObjY->method2();
   :
   delete pObjX;
   delete pObjY;
   }
   ```

Most of the samples in this book use automatic storage. You are ***advised*** to use automatic storage, because you do not have remember to explicitly delete objects, but you are free to use either style for CICS C++ Foundation Class programs. For more information on Foundation Classes and storage management see “Storage management” on page 511.

Using an object

Any of the class public methods can be called on an object of that class.

Any of the class public methods can be called on an object of that class. The following example creates object *obj* and then calls method **doSomething** on it:

```
ClassY obj("TEMP1234");
obj.doSomething();
```

Alternatively, you can do this using dynamic object creation:

```
ClassY* pObj = new ClassY("parameter1");
pObj->doSomething();
```

Deleting an object

When an object is destroyed its destructor function, which has the same name as the class preceded with ~(tilde), is automically called. (You cannot call the destructor explicitly).

If the object was created automatically it is automatically destroyed when it goes out of scope.

If the object was created dynamically it exists until an explicit **delete** operator is used.

Overview of the foundation classes

This topic is a formal introduction to what the Foundation Classes can do for you.

See ICC$HEL: C++ Hello World sample for a simple example to get you started. The section takes a brief look at the CICS C++ Foundation Class library by considering the categories in turn.

See Foundation classes reference for more detailed information on the Foundation Classes.

Every class that belongs to the CICS Foundation Classes is prefixed by **Icc**.

Base classes

All classes inherit, directly or indirectly, from **IccBase** .

IccBase
  IccRecordIndex
  IccResource
  IccControl
  IccTime
  IccResourceId

Figure 124. Base classes

All resource identification classes, such as **IccTermId** , and **IccTransId** , inherit from **IccResourceId** class. These are typically CICS table entries.

All CICS resources—in fact any class that needs access to CICS services—inherit from **IccResource** class.

Base classes enable common interfaces to be defined for categories of class. They are used to create the foundation classes, as provided by IBM , and they can be used by application programmers to create their own derived classes.

IccBase
The base for every other foundation class. It enables memory management and allows objects to be interrogated to discover which type they are.

IccControl
The abstract base class that the application program has to subclass and provide with an implementation of the **run** method.

IccResource
The base class for all classes that access CICS resources or services. See “Resource classes” on page 478.

IccResourceId
The base class for all table entry (resource name) classes, such as **IccFileId** and **IccTempStoreId**.

IccTime
The base class for the classes that store time information: **IccAbsTime** , **IccTimeInterval** and **IccTimeOfDay**.

Resource identification classes

Resource identification classes are as follows.

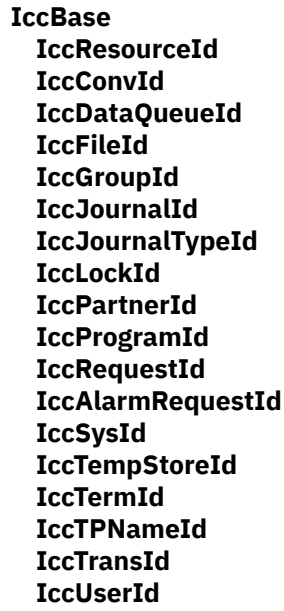


Figure 125. Resource identification classes

CICS resource identification classes define CICS resource identifiers – typically the name of the resource as specified in its RDO resource definition. For example an **IccFileId** object represents a CICS file name. All concrete resource identification classes have the following properties:

- The name of the class ends in **Id**.
- The class is a subclass of the **IccResourceId** class.
- The constructors check that any supplied resource identifier meets CICS standards. For example, an **IccFileId** object must contain a 1 to 8 byte character field; providing a 9-byte field is not tolerated.

The resource identification classes improve type checking; methods that expect an **IccFileId** object as a parameter do not accept an **IccProgramId** object instead. If character strings representing the resource names are used instead, the compiler cannot check for validity – it cannot check whether the string is a file name or a program name.

Many of the resource classes, described in “Resource classes” on page 478 , contain resource identification classes. For example, an **IccFile** object contains an **IccFileId** object. You must use the resource object, not the resource identification object, to operate on a CICS resource. For example, you must use **IccFile** , rather than **IccFileId** to read a record from a file.

Class	CICS resource
IccAlarmRequestId	alarm request
IccConvId	conversation
IccDataQueueId	transient data queue

Class	CICS resource
IccFileId	file
IccGroupId	group
IccJournalId	journal
IccJournalTypeId	journal type
IccLockId	(Not applicable)
IccPartnerId	APPC partner definition files
IccProgramId	program
IccRequestId	request
IccSysId	remote system
IccTempStoreId	temporary storage queue
IccTermId	terminal
IccTPNameId	remote APPC TP name
IccTransId	transaction
IccUserId	user

Resource classes

All CICS resource classes inherit from the **IccResource** base class.

IccBase
  IccResource
  IccAbendData
  IccClock
  IccConsole
  IccControl
  IccDataQueue
  IccFile
  IccFileIterator
  IccJournal
  IccProgram
  IccSemaphore
  IccSession
  IccStartRequestQ
  IccSystem
  IccTask
  IccTempStore
  IccTerminal
  IccTerminalData
  IccUser

Figure 126. Resource classes

These classes model the behavior of the major CICS resources, for example:

- Terminals are modelled by **IccTerminal**.
- Programs are modelled by **IccProgram**.
- Temporary Storage queues are modelled by **IccTempStore**.
- Transient Data queues are modelled by **IccDataQueue**.

Any operation on a CICS resource may raise a CICS condition; the **condition** method of **IccResource** (see IccResource method: condition) can interrogate it.

(Any class that accesses CICS services ***must*** be derived from **IccResource**).

Class	CICS resource
IccAbendData	task abend data
IccClock	CICS time and date services
IccConsole	CICS console
IccControl	control of executing program
IccDataQueue	transient data queue
IccFile	file
IccFileIterator	file iterator (browsing files)
IccJournal	user or system journal
IccProgram	program (outside executing program)
IccSemaphore	semaphore (locking services)
IccSession	session
IccStartRequestQ	start request queue; asynchronous transaction starts
IccSystem	CICS system
IccTask	current task
IccTempStore	temporary storage queue
IccTerminal	terminal belonging to current task
IccTerminalData	attributes of **IccTerminal**
IccTime	time specification
IccUser	user (security attributes)

Support Classes

Support classes are as follows.

IccBase
- **IccBuf**
- **IccEvent**
- **IccException**
- **IccMessage**
- **IccRecordIndex**
- **IccKey**
- **IccRBA**
- **IccRRN**
- **IccResource**
- **IccTime**
- **IccAbsTime**
- **IccTimeInterval**
- **IccTimeOfDay**

Figure 127. Support classes

These classes are tools that complement the resource classes: they make life easier for the application programmer and thus add value to the object model.

Resource class	Description
IccAbsTime	Absolute time (milliseconds since January 1 1900)
IccBuf	Data buffer (makes manipulating data areas easier)
IccEvent	Event (the outcome of a CICS command)
IccException	Foundation Class exception (supports the C++ exception handling model)
IccTimeInterval	Time interval (for example, five minutes)
IccTimeOfDay	Time of day (for example, five minutes past six)

IccAbsTime, IccTimeInterval and **IccTimeOfDay** classes make it simpler for the application programmer to specify time measurements as objects within an application program. **IccTime** is a base class: **IccAbsTime** , **IccTimeInterval** , and **IccTimeOfDay** are derived from **IccTime**.

Consider method **delay** in class **IccTask** , whose signature is as follows:

```
void delay(const IccTime& time, const IccRequestId*
reqId = 0);
```

To request a delay of 1 minute and 7 seconds (that is, a time interval) the application programmer can do this:

```
IccTimeInterval time(0, 1, 7);
task()->delay(time);
```

Note: The task method is provided in class **IccControl** and returns a pointer to the application's task object.

Alternatively, to request a delay until 10 minutes past twelve (lunchtime?) the application programmer can do this:

```
IccTimeOfDay lunchtime(12, 10);
task()->delay(lunchtime);
```

The **IccBuf** class allows easy manipulation of buffers, such as file record buffers, transient data record buffers, and COMMAREAs (for more information on **IccBuf** class see “Buffer objects” on page 481).

IccMessage class is used primarily by **IccException** class to encapsulate a description of why an exception was thrown. The application programmer can also use **IccMessage** to create their own message objects.

IccException objects are thrown from many of the methods in the Foundation Classes when an error is encountered.

The **IccEvent** class allows a programmer to gain access to information relating to a particular CICS event (command).

Using CICS resources

To use a CICS resource, such as a file or program, you must first create an appropriate object and then call methods on the object.

Creating a resource object

When you create a resource object you create a representation of the actual CICS resource (such as a file or program). You do not create the CICS resource; the object is the application's view of the resource. The same is true of destroying objects.

U se an accompanying resource identification object when creating a resource object. For example:

```
IccFileId id("XYZ123");
IccFile file(id);
```

This allows the C++ compiler to protect you against doing something wrong such as:

```
IccDataQueueId id("WXYZ");
IccFile file(id); //gives error at compile time
```

The alternative of using the text name of the resource when creating the object is also permitted:

```
IccFile file("XYZ123");
```

Singleton classes
Many resource classes, such as **IccFile** , can be used to create multiple resource objects within a single program.

```
IccFileId id1("File1");
IccFileId id2("File2");
IccFile file1(id1);
IccFile file2(id2);
```

However, some resource classes are designed to allow the programmer to create only ***one*** instance of the class; these are called singleton classes. The following Foundation Classes are singleton:

- **IccAbendData** provides information about task abends.
- **IccConsole** , or a derived class, represents the system console for operator messages.
- **IccControl** , or a derived class, such as **IccUserControl** , controls the executing program.
- **IccStartRequestQ** , or a derived class, allows the application program to start CICS transactions (tasks) asynchronously.
- **IccSystem** , or a derived class, is the application view of the CICS system in which it is running.
- **IccTask** , or a derived class, represents the CICS task under which the executing program is running.
- **IccTerminal** , or a derived class, represents your task's terminal, provided that your principal facility is a 3270 terminal.

Any attempt to create more than one object of a singleton class results in an error – a C++ exception is thrown.

A class method, **instance** , is provided for each of these singleton classes, which returns a pointer to the requested object and creates one if it does not already exist. For example:

```
IccControl* pControl = IccControl::instance();
```

Calling methods on a resource object

Any of the public methods can be called on an object of that class.

For example:

```
IccTempStoreId id("TEMP1234");
IccTempStore temp(id);
temp.writeItem("Hello TEMP1234");
```

Method **writeItem** writes the contents of the string it is passed ("Hello TEMP1234") to the CICS Temporary Storage queue "TEMP1234".

Buffer objects

The Foundation Classes make extensive use of **IccBuf** objects – buffer objects that simplify the task of handling pieces of data or records.

Understanding the use of these objects is a necessary precondition for much of the rest of this book.

Each of the CICS Resource classes that involve passing data to CICS (for example by writing data records) and getting data from CICS (for example by reading data records) make use of the **IccBuf** class. Examples of such classes are **IccConsole** , **IccDataQueue** , **IccFile** , **IccFileIterator** , **IccJournal** , **IccProgram** , **IccSession** , **IccStartRequestQ** , **IccTempStore** , and **IccTerminal**.

IccBuf class

IccBuf , which is described in detail in the reference part of this book, provides generalized manipulation of data areas.

Because it can be used in a number of ways, there are several **IccBuf** constructors that affect the behavior of the object. Two important attributes of an **IccBuf** object are now described.

Data area ownership

IccBuf has an attribute indicating whether the data area has been allocated inside or outside of the object.

The possible values of this attribute are 'internal' and 'external'. It can be interrogated by using the **dataAreaOwner** method.

Internal/External ownership of buffers
When **DataAreaOwner** = external, it is the application programmer's responsibility to ensure the validity of the storage on which the **IccBuf** object is based. If the storage is invalid or inappropriate for a particular method applied to the object, unpredictable results will occur.

Data area extensibility

This attribute defines whether the length of the data area within the **IccBuf** object, once created, can be increased.

The possible values of this attribute are 'fixed' and 'extensible'. It can be interrogated by using the **dataAreaType** method.

As an object that is 'fixed' cannot have its data area size increased, the length of the data (for example, a file record) assigned to the **IccBuf** object must not exceed the data area length, otherwise a C++ exception is thrown.

Note: By definition, an 'extensible' buffer ***must*** also be 'internal'.

IccBuf constructors

There are several forms of the **IccBuf** constructor, used when creating **IccBuf** objects.

Some examples are shown here.

```
IccBuf buffer;
```

This creates an 'internal' and 'extensible' data area that has an initial length of zero. When data is assigned to the object the data area length is automatically extended to accommodate the data being assigned.

```
IccBuf buffer(50);
```

This creates an 'internal' and 'extensible' data area that has an initial length of 50 bytes. The data length is zero until data is assigned to the object. If 50 bytes of data are assigned to the object, both the data length and the data area length return a value of 50. When more than 50 bytes of data are assigned into the object, the data area length is automatically (that is, without further intervention) extended to accommodate the data.

```
IccBuf buffer(50, IccBuf::fixed);
```

This creates an 'internal' and 'fixed' data area that has a length of 50 bytes. If an attempt is made to assign more than 50 bytes of data into the object, the data is truncated and an exception is thrown to notify the application of the error situation.

```
struct MyRecordStruct
{
short id;
short code;
char data(30);
char rating;
};
MyRecordStruct myRecord;
IccBuf buffer(sizeof(MyRecordStruct), &myRecord);
```

This creates an **IccBuf** object that uses an 'external' data area called myRecord. By definition, an 'external' data area is also 'fixed'. Data can be assigned using the methods on the **IccBuf** object or using the `myRecord` structure directly.

```
IccBuf buffer("Hello World");
```

This creates an 'internal' and 'extensible' data area that has a length equal to the length of the string "Hello World". The string is copied into the object's data area. This initial data assignment can then be changed using one of the manipulation methods (such as **insert** , **cut** , or **replace**) provided.

```
IccBuf buffer("Hello World");
buffer << " out there";
IccBuf buffer2(buffer);
```

Here the copy constructor creates the second buffer with almost the same attributes as the first; the exception is the data area ownership attribute – the second object always contains an 'internal' data area that is a copy of the data area in the first. In the given example buffer2 contains "Hello World out there" and has both data area length and data length of 21.

IccBuf methods

An **IccBuf** object can be manipulated using a number of supplied methods; for example you can append data to the buffer, change the data in the buffer, cut data out of the buffer, or insert data into the middle of the buffer.

The operators **const char*** , **=** , **+=** , **==** , **!=** , and **<<** have been overloaded in class **IccBuf** . There are also methods that allow the **IccBuf** attributes to be queried. For more details see the reference section.

Working with IccResource subclasses

To illustrate working with IccResource subclasses, consider writing a queue item to CICS temporary storage using **IccTempstore** class.

```
IccTempStore store("TEMP1234");
IccBuf buffer(50);
```

The **IccTempStore** object created is the application's view of the CICS temporary storage queue named "TEMP1234". The **IccBuf** object created holds a 50-byte data area (it also happens to be 'extensible').

```
buffer = "Hello Temporary Storage Queue";
store.writeItem(buffer);
```

The character string "Hello Temporary Storage Queue" is copied into the buffer. This is possible because the **operator=** method has been overloaded in the **IccBuf** class.

The **IccTempStore** object calls its **writeItem** method, passing a reference to the **IccBuf** object as the first parameter. The contents of the **IccBuf** object are written out to the CICS temporary storage queue.

Now consider the inverse operation, reading a record from the CICS resource into the application program's **IccBuf** object:

```
buffer = store.readItem(5);
```

The **readItem** method reads the contents of the fifth item in the CICS Temporary Storage queue and returns the data as an **IccBuf** reference.

The C++ compiler resolves the given line of code into two method calls, **readItem** defined in class **IccTempStore** and **operator=** which has been overloaded in class **IccBuf** . This second method takes the contents of the returned **IccBuf** reference and copies its data into the buffer.

The given style of reading and writing records using the foundation classes is typical. The final example shows how to write code – using a similar style to the above example – but this time accessing a CICS transient data queue.

```
IccDataQueue queue("DATQ");
IccBuf buffer(50);
buffer = queue.readItem();
buffer << "Some extra data";
queue.writeItem(buffer);
```

The **readItem** method of the **IccDataQueue** object is called, returning a reference to an **IccBuf** which it then assigns (via **operator=** method, overloaded in class **IccBuf**) to the `buffer` object. The character string – "Some extra data" – is appended to the buffer (via **operator chevron «** method, overloaded in class **IccBuf**). The **writeItem** method then writes back this modified buffer to the CICS transient data queue.

You can find further examples of this syntax in the samples presented in the following sections, which describe how to use the foundation classes to access CICS services.

Refer to the reference section for further information on the **IccBuf** class. You might also find the supplied sample – `ICC$BUF` – helpful.

Using CICS Services

This section describes how to use CICS services. The services are considered in turn.

File control

The file control classes **IccFile** , **IccFileId** , **IccKey** , **IccRBA** , and **IccRRN** allow you to read, write, update and delete records in files.

In addition, **IccFileIterator** class allows you to browse through all the records in a file.

An **IccFile** object is used to represent a file. It is convenient, but not necessary, to use an **IccFileId** object to identify a file by name.

An application program reads and writes its data in the form of individual records. Each read or write request is made by a method call. To access a record, the program must identify both the file and the particular record.

VSAM (or VSAM-like) files are of the following types:

KSDS
: Key-sequenced: each record is identified by a key – a field in a predefined position in the record. Each key must be unique in the file.

  The logical order of records within a file is determined by the key. The physical location is held in an index which is maintained by VSAM.

  When browsing, records are found in their logical order.

ESDS
: Entry-sequenced: each record is identified by its relative byte address (RBA).

  Records are held in an ESDS in the order in which they were first loaded into the file. New records are always added at the end and records may not be deleted or have their lengths altered.

  When browsing, records are found in the order in which they were originally written.

RRDS file
: Relative record: records are written in fixed-length slots. A record is identified by the relative record number (RRN) of the slot which holds it.

Reading records

A read operation uses two classes – **IccFile** to perform the operation and one of **IccKey** , **IccRBA** , and **IccRRN** to identify the particular record, depending on whether the file access type is KSDS, ESDS, or RRDS.

The **readRecord** method of **IccFile** class reads the record.

Reading KSDS records

Before reading a record you must use the **registerRecordIndex** method of **IccFile** to associate an object of class **IccKey** with the file.

You must use a key, held in the **IccKey** object, to access records. A 'complete' key is a character string of the same length as the physical file's key. Every record can be separately identified by its complete key.

A key can also be 'generic'. A generic key is shorter than a complete key and is used for searching for a set of records. The **IccKey** class has methods that allow you to set and change the key.

IccFile class has methods **isReadable** , **keyLength** , **keyPosition** , **recordIndex** , and **recordLength** , which help you when reading KSDS records.

Reading ESDS records

You must use a relative byte address (RBA) held in an **IccRBA** object to access the beginning of a record.

Before reading a record you must use the **registerRecordIndex** method of **IccFile** to associate an object of class **IccRBA** with the file.

IccFile class has methods **isReadable** , **recordFormat** , **recordIndex** , and **recordLength** that help you when reading ESDS records.

Reading RRDS records

You must use a relative record number (RRN) held in an **IccRRN** object to access a record.

Before reading a record you must use **registerRecordIndex** method of **IccFile** to associate an object of class **IccRRN** with the file.

IccFile class has methods **isReadable** , **recordFormat** , **recordIndex** , and **recordLength** which help you when reading RRDS records.

Writing records

Writing records is also known as "adding records".

This topic describes writing records that have not previously been written. Writing records that already exist is not permitted unless they have been previously been put into 'update' mode. See “Updating records” on page 486 for more information.

Before writing a record you must use **registerRecordIndex** method of **IccFile** to associate an object of class **IccKey** , **IccRBA** , or **IccRRN** with the file. The **writeRecord** method of **IccFile** class writes the record.

A write operation uses two classes – **IccFile** to perform the operation and one of **IccKey** , **IccRBA** , and **IccRRN** to identify the particular record, depending on whether the file access type is KSDS, ESDS, or RRDS.

If you have more than one record to write, you can improve the speed of writing by using mass insertion of data. You begin and end this mass insertion by calling the **beginInsert** and **endInsert** methods of **IccFile**.

Writing KSDS records
You must use a key, held in an **IccKey** object to access records.

A 'complete' key is a character string that uniquely identifies a record. Every record can be separately identified by its complete key.

The **writeRecord** method of **IccFile** class writes the record.

IccFile class has methods **isAddable** , **keyLength** , **keyPosition** , **recordIndex** , **recordLength** , and **registerRecordIndex** which help you when writing KSDS records.

Writing ESDS records
You must use a relative byte address (RBA) held in an **IccRBA** object to access the beginning of a record.

IccFile class has methods **isAddable** , **recordFormat** , **recordIndex** , **recordLength** , and **registerRecordIndex** that help you when writing ESDS records.

Writing RRDS records
Use the **writeRecord** method to add a new ESDS record.

IccFile class has methods **isAddable** , **recordFormat** , **recordIndex** , **recordLength** , and **registerRecordIndex** that help you when writing RRDS records.

Updating records
Updating a record is also known as "rewriting a record".

Before updating a record you must first read it, using **readRecord** method in 'update' mode. This locks the record so that nobody else can change it.

Use **rewriteRecord** method to update the record. Note that the **IccFile** object remembers which record is being processed and this information is not passed in again.

For an example, see code fragment: "Read record for update".

The base key in a KSDS file must not be altered when the record is modified. If the file definition allows variable-length records, the length of the record can be changed.

The length of records in an ESDS, RRDS, or fixed-length KSDS file must not be changed on update.

For a file defined to CICS as containing fixed-length records, the length of record being updated must be the same as the original length. The length of an updated record must not be greater than the maximum defined to VSAM.

Deleting records
Records can never be deleted from an ESDS file.

Deleting normal records
The **deleteRecord** method of **IccFile** class deletes one or more records, provided they are not locked by virtue of being in 'update' mode.

The records to be deleted are defined by the **IccKey** or **IccRRN** object.

Deleting locked records
The **deleteLockedRecord** method of **IccFile** class deletes a record which has been previously locked by virtue of being put in 'update' mode by the **readRecord** method.

Browsing records

Browsing, or sequential reading of files uses another class – **IccFileIterator** .

An object of this class must be associated with an **IccFile** object and an **IccKey** , **IccRBA** , or **IccRRN** object. After this association has been made the **IccFileIterator** object can be used without further reference to the other objects.

Browsing can be done either forwards, using **readNextRecord** method or backwards, using **readPreviousRecord** method. The **reset** method resets the **IccFileIterator** object to point to the record specified by the **IccKey** or **IccRBA** object.

Examples of browsing files are shown in page Code fragment "List all records in assending order of key".

Example of file control

This sample program demonstrates how to use the **IccFile** and **IccFileIterator** classes.

The source for this sample can be found in C++ sample programs , in file `ICC$FIL`. Here the code is presented without any of the terminal input and output that can be found in the source file.

```
#include "icceh.hpp"
#include "iccmain.hpp"
```

The first two lines include the header files for the Foundation Classes and the standard **main** function which sets up the operating environment for the application program.

```
const char* fileRecords[] =
{
//NAME KEY PHONE USERID
"BACH, J S 003 00-1234 BACH ",
"BEETHOVEN, L 007 00-2244 BEET ",
"CHOPIN, F 004 00-3355 CHOPIN ",
"HANDEL, G F 005 00-4466 HANDEL ",
"MOZART, W A 008 00-5577 WOLFGANG "
};
```

This defines several lines of data that are used by the sample program.

```
void IccUserControl::run()
{
```

The **run** method of **IccUserControl** class contains the user code for this example. As a terminal is to be used, the example starts by creating a terminal object and clearing the associated screen.

```
 short recordsDeleted = 0;
IccFileId id("ICCKFILE");
IccKey key(3,IccKey::generic);
IccFile file( id );
file.registerRecordIndex( &key );
key = "00";
recordsDeleted = file.deleteRecord();
```

The *key* and *file* objects are first created and then used to delete all the records whose key starts with "00" in the KSDS file "ICCKFILE". *key* is defined as a generic key having 3 bytes, only the first two of which are used in this instance.

```
 IccBuf buffer(40);
                       key.setKind( IccKey::complete );
                       for (short j = 0; j < 5; j++)
                       {
                       buffer = fileRecords[j];
                       key.assign(3, fileRecords[j]+15);
                       file.writeRecord( buffer );
                       }
```

This next fragment writes all the data provided into records in the file. The data is passed by means of an **IccBuf** object that is created for this purpose. **setKind** method is used to change *key* from 'generic' to 'complete'.

The **for** loop between these calls loops round all the data, passing the data into the buffer, using the **operator=** method of **IccBuf** , and thence into a record in the file, by means of **writeRecord**. On the way the key for each record is set, using **assign** , to be a character string that occurs in the data (3 characters, starting 15 characters in).

```
 IccFileIterator fIterator( &file,
                        &key );
                        key = "000";
                        buffer = fIterator.readNextRecord();
                        while (fIterator.condition() == IccCondition::NORMAL)
                        {
                        term->sendLine("- record read: [%s]",(const char*) buffer);
                        buffer = fIterator.readNextRecord();
                        }
```

The loop shown here lists to the terminal, using **sendLine** , all the records in ascending order of key. It uses an **IccFileIterator** object to browse the records. It starts by setting the minimum value for the key which, as it happens, does not exist in this example, and relying on CICS to find the first record in key sequence.

The loop continues until any condition other than NORMAL is returned.

```
 key = "\xFF\xFF\xFF";
fIterator.reset( &key );
buffer = fIterator.readPreviousRecord();
while (fIterator.condition() == IccCondition::NORMAL)
{
buffer = fIterator.readPreviousRecord();
}
```

The next loop is nearly identical to the last, but lists the records in reverse order of key.

```
 key = "008";
                        buffer = file.readRecord( IccFile::update );
                        buffer.replace( 4, "5678", 23);
                        file.rewriteRecord( buffer );
```

This fragment reads a record for update, locking it so that others cannot change it. It then modifies the record in the buffer and writes the updated record back to the file.

```
 buffer = file.readRecord();
```

The same record is read again and sent to the terminal, to show that it has indeed been updated.

```
 return;
}
```

The end of **run** , which returns control to CICS.

See C++ sample programs for the expected output from this sample.

Program control

This section describes how to access and use a program other than the one that is currently executing.

Program control uses **IccProgram** class, one of the resource classes.

Programs may be loaded, unloaded and linked to, using an **IccProgram** object. An **IccProgram** object can be interrogated to obtain information about the program. See IccProgram class for more details.

The example shown here shows one program calling another two programs in turn, with data passing between them via a COMMAREA. One program is assumed to be local, the second is on a remote CICS system. The programs are in two files, ICC$PRG1 and ICC$PRG2. See C++ sample programs for the location of these files and the expected output from these sample programs.

Most of the terminal IO in these samples has been omitted from the code that follows.

```
#include "icceh.hpp"
#include "iccmain.hpp"
void IccUserControl::run()
{
```

The code for both programs starts by including the header files for the Foundation Classes and the stub for **main** method. The user code is located in the **run** method of the **IccUserControl** class for each program.

```
IccSysId sysId( "ICC2" );
IccProgram icc$prg2( "ICC$PRG2" );
IccProgram remoteProg( "ICC$PRG3" );
IccBuf commArea( 100, IccBuf::fixed );
```

The first program (ICC$PRG1) creates an **IccSysId** object representing the remote region, and two **IccProgram** objects representing the local and remote programs that will be called from this program. A 100 byte, fixed length buffer object is also created to be used as a communication area between programs.

```
icc$prg2.load();
if (icc$prg2.condition() == IccCondition::NORMAL)
{
term->sendLine( "Loaded program: %s <%s> Length=%ld Address=%x",
icc$prg2.name(),
icc$prg2.conditionText(),
icc$prg2.length(),
icc$prg2.address() );
icc$prg2.unload();
}
```

The program then attempts to load and interrogate the properties of program ICC$PRG2.

```
 commArea = "DATA SET BY ICC$PRG1";
icc$prg2.link( &commArea );
```

The communication area buffer is set to contain some data to be passed to the first program that ICC$PRG1 links to (ICC$PRG2). ICC$PRG1 is suspended while ICC$PRG2 is run.

The called program, ICC$PRG2, is a simple program, the gist of which is as follows:

```
IccBuf& commArea = IccControl::commArea();
commArea = "DATA RETURNED BY ICC$PRG2";
return;
```

ICC$PRG2 gains access to the communication area that was passed to it. It then modifies the data in this communication area and passes control back to the program that called it.

The first program (ICC$PRG1) now calls another program, this time on another system, as follows:

```
remoteProg.setRouteOption( sysId );
commArea = "DATA SET BY ICC$PRG1";
remoteProg.link( &commArea );
```

The **setRouteOption** requests that calls on this object are routed to the remote system. The communication area is set again (because it will have been changed by ICC$PRG2) and it then links to the remote program (ICC$PRG3 on system ICC2).

The called program uses CICS temporary storage but the three lines we consider are:

```
IccBuf& commArea = IccControl::commArea();
commArea = "DATA RETURNED BY ICC$PRG3";
return;
```

Again, the remote program (ICC$PRG3) gains access to the communication area that was passed to it. It modifies the data in this communication area and passes control back to the program that called it.

```
 return;
};
```

Finally, the calling program itself ends and returns control to CICS.

Starting transactions asynchronously

The **IccStartRequestQ** class enables a program to start another CICS transaction instance asynchronously (and optionally pass data to the started transaction).

The same class is used by a started transaction to gain access to the data that the task that issued the start request passed to it. Finally start requests (for some time in the future) can be cancelled.

Starting transactions

You can use any of the following methods to establish what data will be sent to the started transaction.

- **registerData** or **setData**
- **setQueueName**
- **setReturnTermId**
- **setReturnTransId**

The actual start is requested using the **start** method.

Accessing start data

A started transaction can access its start data by invoking the **retrieveData** method.

This method stores all the start data attributes in the **IccStartRequestQ** object such that the individual attributes can be accessed using the following methods:

- **data**
- **queueName**
- **returnTermId**
- **returnTransId**

Cancelling unexpired start requests

Unexpired start requests (that is, start requests for some future time that has not yet been reached) can be cancelled using the **cancel** method.

Example of starting transactions

start transaction ISR1 on terminal PEO1 on system ICC1.

CICS system	**ICC1**	**ICC2**
Transaction	ISR1/ITMP	ISR2
Program	ICC$SRQ1/ICC$TMP	ICC$SRQ2
Terminal	PEO1	PEO2

This issues two start requests; the first is cancelled before it has expired. The second starts transaction ISR2 on terminal PEO2 on system ICC2. This transaction accesses its start data and finishes by starting transaction ITMP on the original terminal (PEO1 on system ICC1).

The programs and the expected output from them, can be found in C++ sample programs as files `ICC$SRQ1` and `ICC$SRQ2`. Here the code is presented without the terminal IO requests.

Transaction ISR1 runs program `ICC$SRQ1` on system ICC1. Let us consider this program first:

```
#include "icceh.hpp"
#include "iccmain.hpp"
void IccUserControl::run()
{
```

These lines include the header files for the Foundation Classes, and the **main** function needed to set up the class library for the application program. The **run** method of **IccUserControl** class contains the user code for this example.

```
 IccRequestId req1;
IccRequestId req2("REQUEST1");
IccTimeInterval ti(0,0,5);
IccTermId remoteTermId("PE02");
IccTransId ISR2("ISR2");
IccTransId ITMP("ITMP");
IccBuf buffer;
IccStartRequestQ* startQ = startRequestQ();
```

Here we are creating a number of objects:

req1
An empty **IccRequestId** object ready to identify a particular start request.

req2
An **IccRequestId** object containing the user-supplied identifier "REQUEST1".

ti
An **IccTimeInterval** object representing 0 hours, 0 minutes, and 5 seconds.

remoteTermId
An **IccTermId** object; the terminal on the remote system where we start a transaction.

ISR2
An **IccTransId** object; the transaction we start on the remote system.

ITMP
An **IccTransId** object; the transaction that the started transaction starts on this program's terminal.

buffer
An **IccBuf** object that holds start data.

Finally, the **startRequestQ** method of **IccControl** class returns a pointer to the single instance (singleton) class **IccStartRequestQ**.

```
 startQ->setRouteOption( "ICC2" );
startQ->registerData( &buffer );
startQ->setReturnTermId( terminal()->name() );
startQ->setReturnTransId( ITMP );
startQ->setQueueName( "startqnm" );
```

This code fragment prepares the start data that is passed when we issue a start request. The **setRouteOption** says we will issue the start request on the remote system, ICC2. The **registerData** method associates an **IccBuf** object that will contain the start data (the contents of the **IccBuf** object are not extracted until we issue the start request). The **setReturnTermId** and **setReturnTransId** methods allow the start requester to pass a transaction and terminal name to the started transaction. These fields are typically used to allow the started transaction to start ***another*** transaction (as specified) on another terminal, in this case ours.

The **setQueueName** is another piece of information that can be passed to the started transaction.

```
 buffer = "This is a greeting from program
'icc$srq1'!!";
req1 = startQ->start( ISR2, &remoteTermId, &ti );
startQ->cancel( req1 );
```

Here we set the data that we pass on the start requests. We start transaction ISR2 after an interval *ti* (5 seconds). The request identifier is stored in *req1* . Before the five seconds has expired (that is, immediately) we cancel the start request.

```
 req1 = startQ->start( ISR2, &remoteTermID,
&ti, &req2 );
return;
}
```

Again we start transaction ISR2 after an interval *ti* (5 seconds). This time the request is allowed to expire so transaction ISR2 is started on the remote system. Meanwhile, we end by returning control to CICS.

Let us now consider the started program, ICC$SRQ2.

```
 IccBuf buffer;
IccRequestId req("REQUESTX");
IccTimeInterval ti(0,0,5);
IccStartRequestQ* startQ = startRequestQ();
```

Here, as in ICC$SRQ1 , we create a number of objects:

buffer
An **IccBuf** object to hold the start data we were passed by our caller (ICC$SRQ1).

req
An **IccRequestId** object to identify the start we will issue on our caller's terminal.

ti
An **IccTimeInterval** object representing 0 hours, 0 minutes, and 5 seconds.

The **startRequestQ** method of **IccControl** class returns a pointer to the singleton class **IccStartRequestQ**.

```
 if ( task()->startType() != IccTask::startRequest )
{
term->sendLine(
"This program should only be started via the StartRequestQ");
task()->abend( "OOPS" );
}
```

Here we use the **startType** method of **IccTask** class to check that ICC$SRQ2 was started by the **start** method, and not in any other way (such as typing the transaction name on a terminal). If it was not started as intended, we abend with an "OOPS" abend code.

```
 startQ->retrieveData();
```

We retrieve the start data that we were passed by ICC$SRQ1 and store within the **IccStartRequestQ** object for subsequent access.

```
 buffer = startQ->data();
term->sendLine( "Start buffer contents = [%s]", buffer.dataArea() );
term->sendLine( "Start queue= [%s]", startQ->queueName() );
term->sendLine( "Start rtrn = [%s]",
startQ->returnTransId().name());
term->sendLine( "Start rtrm = [%s]", startQ->returnTermId().name() );
```

The start data buffer is copied into our **IccBuf** object. The other start data items (queue, returnTransId, and returnTermId) are displayed on the terminal.

```
 task()->delay( ti );
```

We delay for five seconds (that is, we sleep and do nothing).

```
 startQ->setRouteOption( "ICC1" );
```

The **setRouteOption** signals that we will start on our caller's system (ICC1).

```
 startQ->start(
startQ->returnTransId(),startQ->returnTermId());
return;
```

We start a transaction called ITMP (the name of which was passed by ICC$SRQ1 in the returnTransId start information) on the originating terminal (where ICC$SRQ1 completed as it started this transaction). Having issued the start request, ICC$SRQ1 ends, by returning control to CICS.

Finally, transaction ITMP runs on the first terminal. This is the end of this demonstration of starting transactions asynchronously.

Transient Data

The transient data classes, **IccDataQueue** and **IccDataQueueId** , allow you to store data in transient data queues for subsequent processing.

You can:

- Read data from a transient data queue (**readItem** method)
- Write data to a transient data queue (**writeItem** method)
- Delete a transient data queue (**empty** method)

An **IccDataQueue** object is used to represent a temporary storage queue. An **IccDataQueueId** object is used to identify a queue by name. Once the **IccDataQueueId** object is initialized it can be used to identify the queue as an alternative to using its name, with the advantage of additional error detection by the C++ compiler.

The methods available in **IccDataQueue** class are similar to those in the **IccTempStore** class. For more information on these see "Temporary storage" on page 494.

Reading data

The **readItem** method is used to read items from the queue.

It returns a reference to the **IccBuf** object that contains the information.

Writing data

The **writeItem** method of **IccDataQueue** adds a new item of data to the queue, taking the data from the buffer specified.

Deleting queues

The **empty** method deletes all items on the queue.

Example of managing transient data

This sample program demonstrates how to use the **IccDataQueue** and **IccDataQueueId** classes.

It can be found, along with the expected output, in C++ sample programs as file `ICC$DAT`. Here the code is presented without the terminal IO requests.

```
#include "icceh.hpp"
#include "iccmain.hpp"
```

The first two lines include the header files for the foundation classes and the standard **main** function that sets up the operating environment for the application program.

```
const char* queueItems[] =
{
"Hello World - item 1",
"Hello World - item 2",
"Hello World - item 3"
};
```

This defines some buffer for the sample program.

```
void IccUserControl::run()
{
```

The **run** method of **IccUserControl** class contains the user code for this example.

```
 short itemNum =1;
IccBuf buffer( 50 );
IccDataQueueId id( "ICCQ" );
IccDataQueue queue( id );
queue.empty();
```

This fragment first creates an identification object, of type IccDataQueueId containing "ICCQ". It then creates an **IccDataQueue** object representing the transient data queue "ICCQ", which it empties of data.

```
 for (short i=0 ; i<3 ; i++)
{
buffer = queueItems[i];
queue.writeItem( buffer );
}
```

This loop writes the three data items to the transient data object. The data is passed by means of an **IccBuf** object that was created for this purpose.

```
 buffer = queue.readItem();
while ( queue.condition() == IccCondition::NORMAL )
{
buffer = queue.readItem();
}
```

Having written out three records we now read them back in to show they were successfully written.

```
 return;
}
```

The end of **run** , which returns control to CICS.

Temporary storage

The temporary storage classes, **IccTempStore** and **IccTempStoreId** , allow you to store data in temporary storage queues.

You can:

- Read an item from the temporary storage queue (**readItem** method)
- Write a new item to the end of the temporary storage queue (**writeItem** method)
- Update an item in the temporary storage queue (**rewriteItem** method)
- Read the next item in the temporary storage queue (**readNextItem** method)
- Delete all the temporary data (**empty** method)

An **IccTempStore** object is used to represent a temporary storage queue. An **IccTempStoreId** object is used to identify a queue by name. Once the **IccTempStoreId** object is initialized it can be used to identify the queue as an alternative to using its name, with the advantage of additional error detection by the C++ compiler.

The methods available in **IccTempStore** class are similar to those in the **IccDataQueue** class. For more information on these see “Transient Data” on page 493.

Reading items

The **readItem** method of **IccTempStore** reads the specified item from the temporary storage queue.

It returns a reference to the **IccBuf** object that contains the information.

Writing items

Writing items is also known as "adding" items.

This section describes writing items that have not previously been written. Writing items that already exist can be done using the **rewriteItem** method. See “Updating items” on page 495 for more information.

The **writeItem** method of **IccTempStore** adds a new item at the end of the queue, taking the data from the buffer specified. If this is done successfully, the item number of the record added is returned.

Updating items

Updating an item is also known as "rewriting" an item.

The **rewriteItem** method of **IccTempStore** class is used to update the specified item in the temporary storage queue.

Deleting items

You cannot delete individual items in a temporary storage queue.

To delete *all* the temporary data associated with an **IccTempStore** object use the **empty** method of **IccTempStore** class.

Example of Temporary Storage

This sample program demonstrates how to use the **IccTempStore** and **IccTempStoreId** classes.

This program, and the expected output from it, can be found in C++ sample programs , as file `ICC$TMP`. The sample is presented here without the terminal IO requests.

```
#include "icceh.hpp"
#include "iccmain.hpp"
#include <stdlib.h>
```

The first three lines include the header files for the foundation classes, the standard **main** function that sets up the operating environment for the application program, and the standard library.

```
const char* bufferItems[] =
{
"Hello World - item 1",
"Hello World - item 2",
"Hello World - item 3"
};
```

This defines some buffer for the sample program.

```
void IccUserControl::run()
{
```

The **run** method of **IccUserControl** class contains the user code for this example.

```
 short itemNum = 1;
IccTempStoreId id("ICCSTORE");
IccTempStore store( id );
IccBuf buffer( 50 );
store.empty();
```

This fragment first creates an identification object, **IccTempStoreId** containing the field "ICCSTORE". It then creates an **IccTempStore** object representing the temporary storage queue "ICCSTORE", which it empties of records.

```
 for (short j=1 ; j <= 3 ; j++)
{
buffer = bufferItems[j-1];
store.writeItem( buffer );
}
```

This loop writes the three data items to the Temporary Storage object. The data is passed by means of an **IccBuf** object that was created for this purpose.

```
 buffer = store.readItem( itemNum );
while ( store.condition() == IccCondition::NORMAL )
{
buffer.insert( 9, "Modified " );
store.rewriteItem( itemNum, buffer );
itemNum++;
buffer = store.readItem( itemNum );
}
```

This next fragment reads the items back in, modifies the item, and rewrites it to the temporary storage queue. First, the **readItem** method is used to read the buffer from the temporary storage object. The data in the buffer object is changed using the **insert** method of **IccBuf** class and then the **rewriteItem** method overwrites the buffer. The loop continues with the next buffer item being read.

```
 itemNum = 1;
buffer = store.readItem( itemNum );
while ( store.condition() == IccCondition::NORMAL )
{
term->sendLine( " - record #%d = [%s]", itemNum,
(const char*)buffer );
buffer = store.readNextItem();
}
```

This loop reads the temporary storage queue items again to show they have been updated.

```
 return;
}
```

The end of **run** , which returns control to CICS.

Terminal control

The terminal control classes, **IccTerminal** , **IccTermId** , and **IccTerminalData** , allow you to send data to, receive data from, and find out information about the terminal belonging to the CICS task.

An **IccTerminal** object is used to represent the terminal that belongs to the CICS task. It can only be created if the transaction has a 3270 terminal as its principal facility. The **IccTermId** class is used to identify the terminal. **IccTerminalData** , which is owned by **IccTerminal** , contains information about the terminal characteristics.

Sending data to a terminal

The **send** and **sendLine** methods of **IccTerminal** class are used to write data to the screen.

The **set...** methods allow you to do this. You may also want to erase the data currently displayed at the terminal, using the **erase** method, and free the keyboard so that it is ready to receive input, using the **freeKeyboard** method.

Receiving data from a terminal

The **receive** and **receive3270data** methods of **IccTerminal** class are used to receive data from the terminal.

Finding out information about a terminal

You can find out information about both the characteristics of the terminal and its current state.

The **data** object points to the **IccTerminalData** object that contains information about the characteristics of the terminal. The methods in **IccTerminalData** allow you to discover, for example, the height of the screen or whether the terminal supports Erase Write Alternative. Some of the methods in **IccTerminal** also give you information about characteristics, such as how many lines a screen holds.

Other methods give you information about the current state of the terminal. These include **line** , which returns the current line number, and **cursor** , which returns the current cursor position.

Example of terminal control

This sample program demonstrates how to use the **IccTerminal** , **IccTermId** , and **IccTerminalData** classes.

This program, and the expected output from it, can be found in C++ sample programs , as file `ICC$TRM`.

```
#include "icceh.hpp"
#include "iccmain.hpp"
```

The first two lines include the header files for the Foundation Classes and the standard **main** function that sets up the operating environment for the application program.

```
void IccUserControl::run()
{
IccTerminal& term = *terminal();
term.erase();
```

The **run** method of **IccUserControl** class contains the user code for this example. As a terminal is to be used, the example starts by creating a terminal object and clearing the associated screen.

```
 term.sendLine( "First part of the line..." );
term.send( "... a continuation of the line." );
term.sendLine( "Start this on the next line" );
term.sendLine( 40, "Send this to column 40 of current line" );
term.send( 5, 10, "Send this to row 5, column 10" );
term.send( 6, 40, "Send this to row 6, column 40" );
```

This fragment shows how the **send** and **sendLine** methods are used to send data to the terminal. All of these methods can take **IccBuf** references (const IccBuf&) instead of string literals (const char*).

```
 term.setNewLine();
```

This sends a blank line to the screen.

```
 term.setColor( IccTerminal::red );
term.sendLine( "A Red line of text.");
term.setColor( IccTerminal::blue );
term.setHighlight( IccTerminal::reverse );
term.sendLine( "A Blue, Reverse video line of text.");
```

The **setColor** method is used to set the color of the text on the screen and the **setHighlight** method to set the highlighting.

```
 term << "A cout sytle interface... " <<
endl;
term << "you can " << "chain input together; "
<< "use different types, eg numbers: " << (short)123 <<
" "
<< (long)4567890 << " " << (double)123456.7891234
<< endl;
term << "... and everything is buffered till you issue a flush."
<< flush;
```

This fragment shows how to use the iostream–like interface **endl** to start data on the next line. To improve performance, you can buffer data in the terminal until **flush** is issued, which sends the data to the screen.

```
 term.send( 24,1, "Program 'icc$trm' complete: Hit PF12
to End" );
term.waitForAID( IccTerminal::PF12 );
term.erase();
```

The **waitForAID** method causes the terminal to wait until the specified key is hit, before calling the **erase** method to clear the display.

```
 return;
}
```

The end of **run** , which returns control to CICS.

Time and date services

The **IccClock** class controls access to the CICS time and date services.

IccAbsTime holds information about absolute time (the time in milliseconds that have elapsed since the beginning of 1900), and this can be converted to other forms of date and time. The methods available on **IccClock** objects and on **IccAbsTime** objects are very similar.

Example of time and date services

This sample program demonstrates how to use **IccClock** class.

The source for this program, and the expected output from it, can be found in C++ sample programs , as file ICC$CLK . The sample is presented here without the terminal IO requests.

```
#include "icceh.hpp"
#include "iccmain.hpp"
void IccUserControl::run()
{
```

The first two lines include the header files for the Foundation Classes and the standard **main** function that sets up the operating environment for the application program.

The **run** method of **IccUserControl** class contains the user code for this example.

```
 IccClock clock;
```

This creates a clock object.

```
 term->sendLine( "date() = [%s]",
clock.date() );
term->sendLine( "date(DDMMYY) = [%s]",
clock.date(IccClock::DDMMYY) );
term->sendLine( "date(DDMMYY,':') = [%s]",
clock.date(IccClock::DDMMYY,':'));
term->sendLine( "date(MMDDYY) = [%s]",
clock.date(IccClock::MMDDYY));
term->sendLine( "date(YYDDD) = [%s]",
clock.date(IccClock::YYDDD));
```

Here the **date** method is used to return the date in the format specified by the *format* enumeration. In order the formats are system, DDMMYY, DD:MM:YY, MMDDYY and YYDDD. The character used to separate the fields is specified by the *dateSeparator* character (that defaults to nothing if not specified).

```
 term->sendLine( "daysSince1900() = %ld",
clock.daysSince1900());
term->sendLine( "dayOfWeek() = %d",
clock.dayOfWeek());
if ( clock.dayOfWeek() == IccClock::Friday )
term->sendLine( 40, "Today IS Friday" );
else
term->sendLine( 40, "Today is NOT Friday" );
```

This fragment demonstrates the use of the **daysSince1900** and **dayOfWeek** methods. **dayOfWeek** returns an enumeration that indicates the day of the week. If it is Friday, a message is sent to the screen, 'Today IS Friday'; otherwise the message 'Today is NOT Friday' is sent.

```
 term->sendLine( "dayOfMonth() = %d",
clock.dayOfMonth());
term->sendLine( "monthOfYear() = %d",
clock.monthOfYear());
```

This demonstrates the **dayOfMonth** and **monthOfYear** methods of **IccClock** class.

```
 term->sendLine( "time() = [%s]",
clock.time() );
term->sendLine( "time('-') = [%s]",
clock.time('-') );
term->sendLine( "year() = [%ld]",
clock.year());
```

The current time is sent to the terminal, first without a separator (that is HHMMSS format), then with '-' separating the digits (that is, HH-MM-SS format). The year is sent, for example 1996.

```
 return;
};
```

The end of **run** , which returns control to CICS.

Compiling, executing, and debugging

This section describes how to compile, execute, and debug a CICS Foundation Class program.

Compiling a CICS Foundation Class program

To compile and link a CICS Foundation Class program you need access to the program source, a compiler, header files and a dynamic link library.

You need access to the following items:

- The source of the program you are compiling

  Your C++ program source code needs `#include` statements for the Foundation Class headers and the Foundation Class `main()` program stub:

  ```
  #include "icceh.hpp"
  #include "iccmain.hpp"
  ```

- The IBM C++ compiler
- The Foundation Classes header files (see Header files)
- The Foundation Classes dynamic link library (DLL). The `ICCFCDLL` module is in CICSTS56.CICS .SDFHLOAD.

Note that, when using the Foundation Classes, you do not need to translate the "EXEC CICS " API before compile.

The following sample job statements show how to compile, prelink and link a program called `ICC$HEL` :

```
//ICC$HEL JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//PROCLIB JCLLIB ORDER=(
CICSTS56.CICS
.SDFHPROC)
//ICC$HEL EXEC ICCFCCL,INFILE=
indatasetname
(ICC$HEL),OUTFILE=
outdatasetname
(ICC$HEL)
//
```

Header files

The header files are the C++ class definitions needed to compile CICS C++ Foundation Class programs.

C++ Header File	Classes Defined in this Header
ICCABDEH	IccAbendData
ICCBASEH	IccBase
ICCBUFEH	IccBuf
ICCCLKEH	IccClock
ICCCNDEH	IccCondition (struct)

C++ Header File	Classes Defined in this Header
ICCCONEH	IccConsole
ICCCTLEH	IccControl
ICCDATEH	IccDataQueue
ICCEH	see “1” on page 500
ICCEVTEH	IccEvent
ICCEXCEH	IccException
ICCFILEH	IccFile
ICCFLIEH	IccFileIterator
ICCGLBEH	Icc (struct) (global functions)
ICCJRNEH	IccJournal
ICCMSGEH	IccMessage
ICCPRGEH	IccProgram
ICCRECEH	IccRecordIndex, IccKey, IccRBA and IccRRN
ICCRESEH	IccResource
ICCRIDEH	IccResourceId + subclasses (such as IccConvId)
ICCSEMEH	IccSemaphore
ICCSESEH	IccSession
ICCSRQEH	IccStartRequestQ
ICCSYSEH	IccSystem
ICCTIMEH	IccTime, IccAbsTime, IccTimeInterval, IccTimeOfDay
ICCTMDEH	IccTerminalData
ICCTMPEH	IccTempStore
ICCTRMEH	IccTerminal
ICCTSKEH	IccTask
ICCUSREH	IccUser
ICCVALEH	IccValue (struct)

Note:

1. A single header that #includes all the listed header files is supplied as `ICCEH`
2. The file `ICCMAIN` is also supplied with the C++ header files. This contains the **main** function stub that should be used when you build a Foundation Class program.
3. Header files are located in CICSTS56.CICS .SDFHC370.

Executing Programs

To run a compiled and linked (that is, executable) Foundation Classes program you need to do the following.

1. Make the executable program available to CICS . This involves making sure the program is in a suitable directory or load library. Depending on your server, you may also need to create a CICS program definition (using CICS resource definition facilities) before you can execute the program.
2. Logon to a CICS terminal.

3. Run the program.

Program debugging

Having successfully compiled, linked, and attempted to run your Foundation Classes program, you might need to debug it.

There are three options available to help debug a CICS Foundation Classes program:

- Use a symbolic debugger
- Run the Foundation Class Program with tracing active
- Run the Foundation Class Program with the CICS Execution Diagnostic Facility

Symbolic debugger

You can use a symbolic debugger to step through the source of your CICS Foundation Classes program. Debug Tool is shipped as a feature with IBM C/C++. To debug a CICS Foundation Classes program with a symbolic debugger, compile the program with a flag that adds debugging information to your executable program. For CICS Transaction Server for z/OS, this flag is TEST(ALL).

For more information, see Debug Tool for z/OS.

Tracing

You can configure the CICS Foundation Classes to write a trace file for debugging purposes.

Exception tracing is always active. The CETR transaction controls the auxiliary and internal traces for all CICS programs including those developed using the C++ classes.

Execution diagnostic facility

You can use the Execution Diagnostic Facility (EDF) to step through your CICS program, stopping at each **EXEC CICS** call. The display screen shows the procedural **EXEC CICS** call interface rather than the CICS Foundation Class type interface.

To enable EDF, use the preprocessor macro `ICC_EDF` in your source code before including the file `ICCMAIN`.

```
 #define ICC_EDF //switch EDF on
#include "iccmain.hpp"
```

Alternatively use the appropriate flag on your compiler CPARM to declare ICC_EDF.

Conditions, errors, and exceptions

This section describes how the Foundation Classes have been designed to respond to various error situations they might encounter.

Foundation Class Abend codes

For serious errors (such as insufficient storage to create an object) the Foundation Classes immediately terminate the CICS task.

All CICS Foundation Class abend codes are of the form ACLx. If your application is terminated with an abend code starting 'ACL' then please refer to CICS messages.

C++ Exceptions and the Foundation Classes

C++ exceptions are managed using the reserved words **try** , **throw** , and **catch**.

Refer to your compiler's documentation or one of the C++ books in the bibliography for more information.

Here is sample ICC$EXC1 (see C++ sample programs):

```
#include "icceh.hpp"
#include "iccmain.hpp"
class Test {
public:
void tryNumber( short num ) {
IccTerminal* term = IccTerminal::instance();
*term << "Number passed = " << num << endl <<
flush;
if ( num > 10 ) {
*term << ">>Out of Range - throwing exception" << endl
<< flush;
throw "!!Number is out of range!!";
}
}
};
```

The first two lines include the header files for the Foundation Classes and the standard **main** function that sets up the operating environment for the application program.

We then declare class **Test** , which has one public method, **tryNumber** . This method is implemented inline so that if an integer greater than ten is passed an exception is thrown. We also write out some information to the CICS terminal.

```
void IccUserControl::run()
{
IccTerminal* term = IccTerminal::instance();
term->erase();
*term << "This is program 'icc$exc1' ..." << endl;
try {
Test test;
test.tryNumber( 1 );
test.tryNumber( 7 );
test.tryNumber( 11 );
test.tryNumber( 6 );
}
catch( const char* exception ) {
term->setLine( 22 );
*term << "Exception caught: " << exception << endl
<< flush;
}
term->send( 24,1,"Program 'icc$exc1' complete: Hit PF12 to End" );
term->waitForAID( IccTerminal::PF12 );
term->erase();
return;
}
```

The **run** method of **IccUserControl** class contains the user code for this example.

After erasing the terminal display and writing some text, we begin our **try** block. A **try** block can scope any number of lines of C++ code.

Here we create a **Test** object and invoke our only method, **tryNumber** , with various parameters. The first two invocations (1, 7) succeed, but the third (11) causes **tryNumber** to throw an exception. The fourth **tryNumber** invocation (6) is not executed because an exception causes the program execution flow to leave the current **try** block.

We then leave the **try** block and look for a suitable **catch** block. A suitable **catch** block is one with arguments that are compatible with the type of exception being thrown (here a **char***). The **catch** block writes a message to the CICS terminal and then execution resumes at the line after the **catch** block.

The output from this CICS program is as follows:

```
This is program 'icc$exc1' ...
                    Number passed = 1
                    Number passed = 7
                    Number passed = 11
                    >>Out of Range - throwing exception
                    Exception caught: !!Number is out of range!!
                    Program 'icc$exc1' complete: Hit PF12 to End
```

The CICS C++ Foundation Classes do not throw **char*** exceptions as in the previous sample but they do throw **IccException** objects instead.

There are several types of **IccException** . The **type** method returns an enumeration that indicates the type. Here is a description of each type in turn.

objectCreationError
An attempt to create an object was invalid. This happens, for example, if an attempt is made to create a second instance of a singleton class, such as **IccTask**.

invalidArgument
A method was called with an invalid argument. This happens, for example, if an **IccBuf** object with too much data is passed to the **writeItem** method of the **IccTempStore** class by the application program.

It also happens when attempting to create a subclass of **IccResourceId** , such as **IccTermId** , with a string that is too long.

The following sample can be found in C++ sample programs , as file ICC$EXC2 . The sample is presented here without many of the terminal IO requests.

```
#include "icceh.hpp"
#include "iccmain.hpp"
void IccUserControl::run()
{
try
{
IccTermId id1( "1234" );
IccTermId id2( "12345");
}
catch( IccException& exception )
{
terminal()->send( 21, 1, exception.summary() );
}
return;
}
```

In the previous example the first **IccTermId** object is successfully created, but the second caused an **IccException** to be thrown, because the string "12345" is 5 bytes where only 4 are allowed. See C++ sample programs for the expected output from this sample program.

invalidMethodCall
A method cannot be called. A typical reason is that the object cannot honor the call in its current state. For example, a **readRecord** call on an **IccFile** object is only honored if an **IccRecordIndex** object, to specify ***which*** record is to be read, has already been associated with the file.

CICSCondition
A CICS condition, listed in the **IccCondition** structure, has occurred in the object and the object was configured to throw an exception.

familyConformanceError
Family subset enforcement is on for this program and an operation that is not valid on all supported platforms has been attempted.

internalError
The CICS foundation classes have detected an internal error. Please call service.

CICS conditions

The CICS foundation classes provide a powerful framework for handling conditions that happen when executing an application.

Accessing a CICS resource can raise a number of CICS conditions as documented in Foundation classes reference.

A condition might represent an error or information being returned to the calling application; the deciding factor is often the context in which the condition is raised.

The application program can handle the CICS conditions in a number of ways. Each CICS resource object, such as a program, file, or data queue, can handle CICS conditions differently, if required.

A resource object can be configured to take one of the following actions for each condition it can encounter:

noAction
Manual condition handling

callHandleEvent
Automatic condition handling

throwException
Exception handling

abendTask
Severe error handling.

Manual condition handling (noAction)

This is the default action for all CICS conditions (for any resource object).

This means that the condition must be handled manually, using the **condition** method. For example:

```
IccTempStore temp("TEMP1234");
IccBuf buf(40);
temp.setActionOnCondition(IccResource::noAction,
IccCondition::QIDERR);
buf = temp.readNextItem();
switch (temp.condition())
{
case IccCondition::QIDERR:
//do whatever here
:
default:
//do something else here
}
```

Automatic condition handling (callHandleEvent)

Activate this for any CICS condition, such as QIDERR, as follows.

```
IccTempStore temp("TEMP1234");
temp.setActionOnCondition(IccResource::callHandleEvent,
IccCondition::QIDERR);
```

When a call to any method on object 'temp' causes CICS to raise the QIDERR condition, **handleEvent** method is automatically called. As the **handleEvent** method is only a virtual method, this call is only useful if the object belongs to a subclass of **IccTempStore** and the **handleEvent** method has been overridden.

Make a subclass of **IccTempStore** , declare a constructor, and override the **handleEvent** method.

```
class MyTempStore : public IccTempStore
{
public:
MyTempStore(const char* storeName) : IccTempStore(storeName) {}
HandleEventReturnOpt handleEvent(IccEvent& event);
};
```

Now implement the **handleEvent** method.

```
IccResource::HandleEventReturnOpt
MyTempStore::handleEvent(IccEvent& event)
{
switch (event.condition())
{
case …
⋮
case IccCondition::QIDERR:
//Handle QIDERR condition here.
⋮
//
default:
return rAbendTask;
}
}
```

This code is called for any **MyTempStore** object which is configured to 'callHandleEvent' for a particular CICS condition.

Exception handling (throwException)

Activate this for any CICS condition, such as QIDERR, as follows.

```
IccTempStore temp("TEMP1234");
temp.setActionOnCondition(IccResource::throwException,
IccCondition::QIDERR);
```

Exception handling is by means of the C++ exception handling model using **try** , **throw** , and **catch** . For example:

```
try
{
buf = temp.readNextItem();
⋮
}
catch (IccException& exception)
{
//Exception handling code
⋮
}
```

An exception is thrown if any of the methods inside the try block raise the QIDERR condition for object 'temp'. When an exception is thrown, C++ unwinds the stack and resumes execution at an appropriate **catch** block – it is not possible to resume within the **try** block. For a fuller example, see sample `ICC$EXC3`.

Note: Exceptions can be thrown from the Foundation Classes for many reasons other than this example – see “C++ Exceptions and the Foundation Classes” on page 501 for more details.

Severe error handling (abendTask)

This option allows CICS to terminate the task when certain conditions are raised.

Activate this for any CICS condition, such as QIDERR, as follows:

```
IccTempStore temp("TEMP1234");
temp.setActionOnCondition(IccResource::abendTask,
IccCondition::QIDERR);
```

If CICS raises the QIDERR condition for object 'temp' the CICS task terminates with an ACL3 abend.

Platform differences

The CICS Foundation Classes, as described here, are designed to be independent of the particular CICS platform on which they are running. There are however some differences between platforms; these, and ways of coping with them, are described here.

Note: References in this section to other CICS platforms are included for completeness. There have been Technology Releases of the CICS Foundation Classes on those platforms.

Applications can be run in one of two modes:

fsAllowPlatformVariance
Applications written using the CICS Foundation Classes are able to access all the functions available on the target CICS server.

fsEnforce
Applications are restricted to the CICS functions that are available across all CICS Servers (z/OS and UNIX).

The default is to allow platform variance and the alternative is to force the application to only use features which are common to all CICS platforms.

The class headers are the same for all platforms and they "support" (that is, define) all the CICS functions that are available through the Foundation Classes on any of the CICS platforms. The restrictions on each platform are documented in Foundation classes reference. Platform variations exist at:

- object level
- method level
- parameter level

Object level
Some objects are not supported on certain platforms.

For example, **IccConsole** objects cannot be created on CICS(r) for AIX® as CICS(r) for AIX(r) does not support console services.

Any attempt to create an **IccConsole** object on CICS(r) for AIX(r) causes an **IccException** object of type 'platformError' to be thrown, but would be acceptable on the other platforms

```
IccConsole* cons = console(); //No good on CICS for AIX
```

If you initialize your application with 'fsEnforce' selected (see initializeEnvironment) the previous examples both cause an **IccException** object, of type 'familyConformanceError' to be thrown on all platforms.

Unlike objects of the **IccConsole** and **IccJournal** classes, most objects can be created on any CICS server platform. However the use of the methods can be restricted. Foundation classes reference fully documents all platform restrictions.

Method level
Methods that run successfully on one platform can cause a problem on another platform.

Consider, for example method **programId** in the **IccControl** class:

```
void IccUserControl::run()
{
if (strcmp(programId.name(), "PROG1234") == 0)
//do something
}
```

Here method **programId** executes correctly on CICS TS for z/OS but throws an **IccException** object of type 'platformError' on CICS(r) for AIX(r).

Alternatively, if you initialize your application with family subset enforcement on (see **initializeEnvironment** function of **Icc** structure), method **programId** throws an **IccException** object of type 'familyConformanceError' on *any* CICS server platform.

Parameter level
At this level a method is supported on all platforms, but a particular positional parameter has some platform restrictions.

Consider method **abend** in **IccTask** class.

```
task()->abend();
 1

task()->abend("WXYZ");
 2

task()->abend("WXYZ", IccTask::respectAbendHandler);
 3

task()->abend("WXYZ", IccTask::ignoreAbendHandler);
 4

task()->abend("WXYZ", IccTask::ignoreAbendHandler,
 5

IccTask::suppressDump);
```

Abends 1 to 4 run successfully on all CICS server platforms.

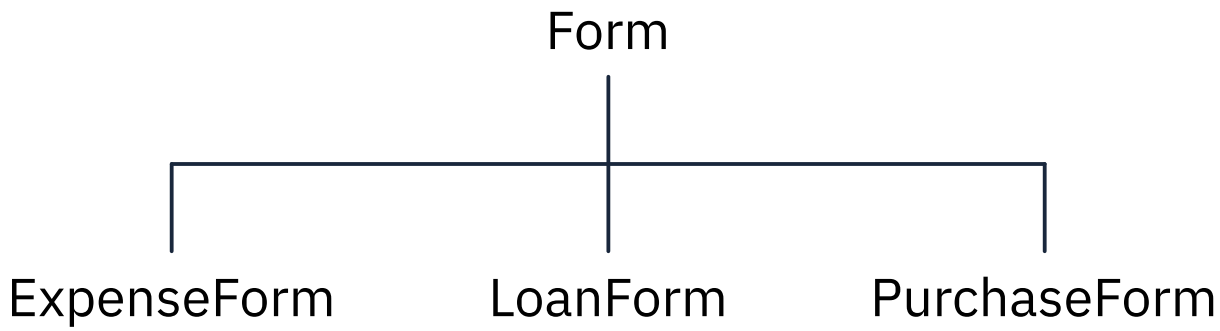
If family subset enforcement is off, abend 5 throws an **IccException** object of type 'platformError' on a CICS(r) for AIX(r) platform, but not on a CICS Transaction Server for z/OS platform.

If family subset enforcement is on, abend 5 throws an **IccException** object of type 'familyConformanceError', irrespective of the target CICS platform.

Polymorphic Behavior

Polymorphism (*poly* = many, *morphe* = form) is the ability to treat many different forms of an object as if they were the same.

Polymorphism is achieved in C++ by using inheritance and virtual functions. Consider the scenario where we have three forms (ExpenseForm, LoanForm, PurchaseForm) that are specializations of a general Form:


Form
ExpenseForm
LoanForm
PurchaseForm

Each form needs printing at some time. In procedural programming, we would either code a print function to handle the three different forms or we would write three different functions (printExpenseForm, printLoanForm, printPurchaseForm).

In C++, this can be achieved far more elegantly as follows:

```
 class Form {
public:
virtual void print();
};
class ExpenseForm : public Form {
public:
virtual void print();
};
class LoanForm : public Form {
public:
virtual void print();
};
class PurchaseForm : public Form {
public:
virtual void print();
};
```

Each of these overridden functions is implemented so that each form prints correctly. Now an application using form objects can do this:

```
Form* pForm[10]
//create Expense/Loan/Purchase Forms…
for (short i=0 ; i < 9 ; i++)
pForm->print();
```

Here we create ten objects that might be any combination of Expense, Loan, and Purchase Forms. However, because we are dealing with pointers to the base class, **Form** , we do not need to know which sort of form object we have; the correct **print** method is called automatically.

Limited polymorphic behavior is available in the Foundation Classes. Three virtual functions are defined in the base class **IccResource** :

```
virtual void clear();
virtual const IccBuf& get();
virtual void put(const IccBuf&

buffer

);
```

These methods have been implemented in the subclasses of **IccResource** wherever possible:

Class	clear	get	put
IccConsole	×	×	✓
IccDataQueue	✓	✓	✓
IccJournal	×	×	✓
IccSession	×	✓	✓
IccTempStore	✓	✓	✓
IccTerminal	✓	✓	✓

These virtual methods are ***not*** supported by any subclasses of **IccResource** except those in the table.

Note: The default implementations of **clear** , **get** , and **put** in the base class **IccResource** throw an exception to prevent the user from calling an unsupported method.

Example of polymorphic behavior

The following sample can be found in the samples directory as file ICC$RES2.

It is presented here without the terminal IO requests. See C++ sample programs.

```
#include "icceh.hpp"
#include "iccmain.hpp"
char* dataItems[] =
{
"Hello World - item 1",
"Hello World - item 2",
"Hello World - item 3"
};
void IccUserControl::run()
{
```

Here we include Foundation Class headers and the **main** function. **dataItems** contains some sample data items. We write our application code in the **run** method of **IccUserControl** class.

```
 IccBuf buffer( 50 );
IccResource* pObj[2];
```

We create an **IccBuf** object (50 bytes initially) to hold our data items. An array of two pointers to **IccResource** objects is declared.

```
 pObj[0] = new IccDataQueue("ICCQ");
pObj[1] = new IccTempStore("ICCTEMPS");
```

We create two objects whose classes are derived from **IccResource** – **IccDataQueue** and **IccTempStore**.

```
 for ( short index=0; index <= 1 ; index++ )
{
pObj[index]->clear();
}
```

For both objects we invoke the **clear** method. This is handled differently by each object in a way that is transparent to the application program; this is polymorphic behavior.

```
 for ( index=0; index <= 1 ; index++ )
{
for (short j=1 ; j <= 3 ; j++)
{
buffer = dataItems[j-1];
pObj[index]->put( buffer );
}
}
```

Now we **put** three data items in each of our resource objects. Again the **put** method responds to the request in a way that is appropriate to the object type.

```
 for ( index=0; index <= 1 ; index++ )
{
buffer = pObj[index]->get();
while (pObj[index]->condition() == IccCondition::NORMAL)
{
buffer = pObj[index]->get();
}
delete pObj[index];
}
return;
}
```

The data items are read back in from each of our resource objects using the **get** method. We delete the resource objects and return control to CICS.

Storage management

C++ objects are usually stored on the stack or heap.

Objects on the stack are automatically destroyed when they go out of scope, but objects on the heap are not.

Many of the objects that the CICS Foundation Classes create internally are created on the heap rather than the stack. This can cause a problem in some CICS server environments.

On CICS Transaction Server for z/OS ,, CICS and Language Environment manage all task storage so that it is released at task termination (normal or abnormal).

In a CICS for AIX environment, storage allocated on the heap is not automatically released at task termination. This can lead to "memory leaks" if the application programmer forgets to explicitly delete an object on the heap, or, more seriously, if the task abends.

This problem has been overcome in the CICS Foundation Classes by providing operators **new** and **delete** in the base Foundation Class, **IccBase** . These can be configured to map dynamic storage allocation requests to CICS task storage, so that ***all*** storage is automatically released at task termination. The disadvantage of this approach is a performance hit as the Foundation Classes typically issue a large number of small storage allocation requests rather than a single, larger allocation request.

This facility is affected by the **Icc::initializeEnvironment** call that must be issued before using the Foundation Classes. (This function is called from the default **main** function; see CICS C++ main function .)

The first parameter passed to the **initializeEnvironment** function is an enumeration that takes one of these three values:

cmmDefault
The default action is platform dependent:

z/OS
same as 'cmmNonCICS' - see the 'cmmNonCICS' section.

UNIX
same as 'cmmCICS' - see the 'cmmCICS' section.

cmmNonCICS
The **new** and **delete** operators in class **IccBase** ***do not*** map dynamic storage allocation requests to CICS task storage; instead the C++ default **new** and **delete** operators are invoked.

cmmCICS
The **new** and **delete** operators in class **IccBase** map dynamic storage allocation requests to CICS task storage (which is automatically released at normal or abnormal task termination).

The default **main** function supplied with the Foundation Classes calls **initializeEnvironment** with an enum of 'cmmDefault'. You can change this in your program without changing the supplied "header file" ICCMAIN as follows:

```
#define ICC_CLASS_MEMORY_MGMT Icc::cmmNonCICS
#include "iccmain.hpp"
```

Alternatively, set the option **DEV(ICC_CLASS_MEMORY_MGMT)** when compiling.

Parameter passing conventions

The convention used for passing objects on Foundation Classes method calls is if the object is mandatory, pass by reference; if it is optional pass by pointer.

For example, consider method **start** of class **IccStartRequestQ** , which has the following signature:

```
const IccRequestId& start( const IccTransId&
transId,
const IccTime* time=0,
const IccRequestId* reqId=0 );
```

Using the preceding convention, we see that an **IccTransId** object is mandatory, while an **IccTime** and an **IccRequestId** object are both optional. This enables an application to use this method in any of the following ways:

```
 IccTransId trn("ABCD");
IccTimeInterval int(0,0,5);
IccRequestId req("MYREQ");
IccStartRequestQ* startQ = startRequestQ();
startQ->start( trn );
startQ->start( trn, &int );
startQ->start( trn, &int, &req );
startQ->start( trn, 0, &req );
```

Scope of data in IccBuf reference returned from 'read' methods

Many of the subclasses of **IccResource** have 'read' methods that return **const IccBuf** references; for example, **IccFile::readRecord** , **IccTempStore::readItem** and **IccTerminal::receive**.

Care should be taken if you choose to maintain a reference to the **IccBuf** object, rather than copy the data from the **IccBuf** reference into your own **IccBuf** object. For example, consider the following

```
 IccBuf buf(50);
IccTempStore store("TEMPSTOR");
buf = store.readNextItem();
```

Here, the data in the **IccBuf** reference returned from **IccTempStore::readNextItem** is ***immediately*** copied into the application's own **IccBuf** object, so it does not matter if the data is later invalidated. However, the application might look like this

```
 IccTempStore store("TEMPSTOR");
const IccBuf& buf = store.readNextItem();
```

Here, the **IccBuf** reference returned from **IccTempStore::readNextItem** is ***not*** copied into the application's own storage and care must therefore be taken.

Note: You are recommended not to use this style of programming to avoid using a reference to an **IccBuf** object that does not contain valid data.

The returned **IccBuf** reference typically contains valid data until one of the following conditions is met:

- Another 'read' method is invoked on the **IccResource** object (for example, another **readNextItem** or **readItem** method in the example).
- The resource updates are committed (see method **IccTask::commitUOW**).
- The task ends (normally or abnormally).

Chapter 8. Developing COBOL applications

Use this information to help you code, translate, and compile COBOL programs that you want to use as CICS application programs.

Changes to CICS support for application programming languages lists the COBOL compilers that are supported by CICS Transaction Server for z/OS, Version 5 Release 6 , and their service status on z/OS.

All references to COBOL in CICS Transaction Server for z/OS, Version 5 Release 6 documentation imply the use of a supported Language Environment-conforming compiler such as Enterprise COBOL for z/OS , unless stated otherwise. The only COBOL compiler that has runtime support in CICS Transaction Server for z/OS, Version 5 Release 6 , but is not Language Environment-conforming, is the VS COBOL II compiler.

See the Enterprise COBOL for z/OS Migration Guide for information about migrating between COBOL compilers.

Support for VS COBOL II

In CICS Transaction Server for z/OS, Version 5 Release 6 , applications compiled with a VS COBOL II compiler run using the Language Environment runtime library routines. The runtime library provided with VS COBOL II is not supported.

"VS COBOL II programs" on page 517 lists some restrictions and considerations associated with programs compiled with the VS COBOL II compiler.

The VS COBOL II compiler can adjust Language Environment runtime options to allow these applications to run correctly. For more information, see the Enterprise COBOL for z/OS Migration Guide.

Support for OO COBOL

In CICS Transaction Server for z/OS, Version 5 Release 6 , COBOL class definitions and methods (object-oriented COBOL) cannot be used. This restriction includes both Java classes and COBOL classes.

Modules compiled in earlier CICS releases with the OOCOBOL translator option cannot run in CICS Transaction Server for z/OS, Version 5 Release 6 . The OOCOBOL translator option was used for the older SOM-based (System Object Manager-based) OO COBOL, and runtime support for this form of OO COBOL was withdrawn in z/OS V1.2. The newer Java-based OO COBOL, which is used in Enterprise COBOL, is not supported by the CICS translator.

If you have existing SOM-based OO COBOL programs, rewrite your OO COBOL into procedural (non-OO) COBOL in order to use the Enterprise COBOL compiler. Java-based OO COBOL is not compatible with SOM-based OO COBOL programs, and is not intended as a migration path for SOM-based OO COBOL programs.

Working storage

With compiler option DATA(24), the working storage is allocated below the 16 MB line. With compiler option DATA(31), the working storage is allocated above the 16 MB line.

```
Export-Package: com.ibm.jzos,
  com.ibm.jzos.fields
```

COBOL programming restrictions and requirements

Some restrictions and requirements apply to a COBOL program that is used as a CICS application program.

By default, the CICS translator and the COBOL compiler do not detect the use of COBOL words that are affected by the following restrictions. The use of a restricted word in a CICS environment might

© Copyright IBM Corp. 1974, 2023

cause a failure at execution time. However, COBOL provides a reserved-word table, IGYCCICS, for CICS application programs. If you specify the compiler option `WORD(CICS)`, the compiler uses the IGYCCICS table, and the compiler flags COBOL words that are not supported under CICS with an error message. The COBOL words that are normally restricted by the default IBM-supplied reserved-word table are also flagged. For a current listing of the words that are restricted and are in the IGYCCICS table, see Enterprise COBOL for z/OS Programming Guide.

Functions and statements that cannot be used

- You cannot use entry points in COBOL in CICS.
- You must use CICS commands for most input and output processing. Therefore, do not describe files or code any OPEN, CLOSE, READ, START, REWRITE, WRITE, or DELETE statements. Instead, use CICS commands to retrieve, update, insert, and delete data.
- Do not use a format-1 ACCEPT statement in a CICS program. Format-2 ACCEPT statements are supported by Language Environment enabled compilers.
- Do not use DISPLAY . . . UPON CONSOLE and DISPLAY . . . UPON SYSPUNCH. DISPLAY to the system logical output device (SYSOUT, SYSLIST,SYSLST) is supported.
- Do not use STOP "literal".
- Restrictions apply to the use of the SORT statement; see Enterprise COBOL for z/OS Programming Guide. Do not use MERGE.
- Do not use the following:
  - USE declaratives.
  - ENVIRONMENT DIVISION and FILE SECTION entries associated with data management, because CICS handles data management. You can use these entries when they are associated with the limited SORT facility, listed previously.
  - User-specified parameters to the main program.

Considerations for Enterprise COBOL V5 and higher

When using COBOL V5 or higher with CICS, be aware of the CICS conversion considerations documented in the Enterprise COBOL for z/OS documentation.

Coding requirements

- When a debugging line is to be used as a comment, it must not contain any unmatched quotation marks.
- Statements that produce variable-length areas, such as OCCURS DEPENDING ON, should be used with caution within the WORKING-STORAGE SECTION.
- Do not use EXEC CICS commands in a Declaratives Section.
- Start both EXEC CICS and END-EXEC statements in Area B (columns 12-71). Note that the integrated CICS translator does not produce an error message when an EXEC CICS statement starts in Area A (columns 8-11), because the keyword EXEC may be used in Area A as part of other statements, such as SQL statements.
- If no IDENTIFICATION DIVISION is present, only the CICS commands are expanded. If the IDENTIFICATION DIVISION only is present, only DFHEIVAR, DFHEIBLK, and DFHCOMMAREA are produced.
- For VS COBOL II programs with Language Environment runtime, the following limits apply to the length of WORKING-STORAGE:
  - When the compiler option DATA(24) is used, the limit is the available space below the 16 MB line.
  - When the compiler option DATA(31) is used, the limit is 128 MB.

  For storage accounting and save areas, 80 bytes are required; you must include this in the limits.

- If the DLI option is specified and an ENTRY statement immediately follows the PROCEDURE DIVISION header in an existing program, change the PROGRAM-ID name to the ENTRY statement literal and delete the ENTRY statement before calling the program in CICS.
- If you use HANDLE CONDITION or HANDLE AID, you can avoid addressing problems by using SET(ADDRESS OF A-DATA) or SET(A-POINTER) where A-DATA is a structure in the LINKAGE SECTION and A-POINTER is defined with the USAGE IS POINTER clause.

Language Environment coding requirements

If you are running CICS applications written in COBOL under Language Environment for the first time, you might need to review the Language Environment runtime options in use at your installation. In particular, if your applications are not coded to ensure that the WORKING-STORAGE SECTION is properly initialized (for example, cleared with binary zeros before sending maps), you should use the STORAGE runtime option. For information about customizing Language Environment runtime options, see z/OS Language Environment Programming Reference.

31-bit addressing

For a COBOL program running above the 16 MB line, the following restrictions apply for 31-bit addressing:

- If the receiving program is link-edited with AMODE(31), addresses passed to it must be 31 bits long (or 24 bits long with the left-most byte set to zeros).
- If the receiving program is link-edited with AMODE(24), addresses passed to it must be 24 bits long.

Specify the DATA(24) compiler option for programs running in 31-bit addressing mode that are passing data arguments to programs in 24-bit addressing mode. This ensures that the data will be addressable by the called program.

64-bit addressing

64-bit addressing mode is not supported for COBOL programs.

64-bit residency

CICS does not support 64-bit residency mode (RMODE(64)) and treats any RMODE(64) programs as RMODE(31). That is, RMODE(64) programs are loaded into 31-bit (above-the-line) storage, not 64-bit (above-the-bar) storage.

Compiler options

- Do not use the following compiler options:
  - DYNAM (if program is to be translated)
  - NOLIB (if program is to be translated)
  - NORENT
  - NUMPROC(PFD)

    NUMPROC(PFD) must not be used because values used in assignments to COBOL variables (including values assigned by CICS) may not meet the stricter numeric test. Such contraventions, when NUMCHECK(PAC) is also used, will result in a potentially high rate of IGZ0279W messages to be issued, impacting performance.

  You may use the DLL compiler option.
- The following compiler options have no effect in a CICS environment:
  - ADV
  - AWO
  - EXPORTALL

- FASTSRT
- NAME
- OOCOBOL
- OUTDD
- THREAD

- The use of the TEST compiler option might affect program performance. See TEST compiler option in Enterprise COBOL for z/OS Programming Guide for detailed information about the use of this option.
- Use TRUNC(OPT) for handling binary data items if they conform to the PICTURE definitions. Otherwise, use TRUNC(OPT) as the compiler option and USAGE COMP-5 for items where the binary value might be larger than the PICTURE clause would allow. TRUNC(BIN) inhibits runtime performance, so use this option only if you have no control over binary data items (such as those created by a code generator). (TRUNC(STD) is the default.)

  For more information about the TRUNC option, see Enterprise COBOL for z/OS Customization Guide.
- The use of the RMODE(24) compiler option means that the program always resides below the 16 MB line, so this is not recommended. RMODE(ANY) or RMODE(AUTO) should be used instead. For more information about the RMODE compiler option, see Enterprise COBOL for z/OS Programming Guide.

WITH DEBUGGING MODE

If a "D" is placed in column seven of the first line of a COBOL EXEC CICS command, that "D" is also found in the translated CALL statements. This translated command is only executed if WITH DEBUGGING MODE is specified. A "D" placed on any line other than the first line of the EXEC CICS statement is not required and is ignored by the translator.

Language Environment CBLPSHPOP option

The CBLPSHPOP runtime option controls whether Language Environment automatically issues an EXEC CICS PUSH HANDLE command during initialization and an EXEC CICS POP HANDLE command during termination whenever a COBOL subroutine is called.

If your application makes many COBOL subroutine CALLs under CICS , performance is better with CBLPSHPOP(OFF) than with CBLPSHPOP(ON). You can set CBLPSHPOP on an individual transaction basis by using CEEUOPT, as explained in “ Defining runtime options for Language Environment ” on page 541.

However, because condition handling has not been stacked, be aware that:

- If your called routine raises a condition that causes CICS to attempt to pass control to a condition handler in the calling routine, this is an error and your transaction will be abnormally terminated.
- If you use any of the PUSHable CICS commands, HANDLE ABEND, HANDLE AID, HANDLE CONDITION, or IGNORE CONDITION, within the called routine, you will be changing the settings of your caller and this could lead to later errors.
- If you call an assembler routine and need to suspend the current handles , and then reinstate them, the assembler routine must request the push and pop handles. The Language Environment does not do that automatically when a COBOL program calls an assembler routine.

Using the DL/I CALL interface

If you have COBOL programs that use CALL DL/I, and you have not yet made the following changes to them, you should now do so.

- Retain the user interface block (DLIUIB) declaration and at least one program control block (PCB) declaration in the LINKAGE SECTION.
- Change the PCB call to specify the UIB directly, as follows:

```
CALL 'CBLTDLI' USING PCB-CALL
PSB-NAME
ADDRESS OF DLIUIB.
```

- Obtain the address of the required PCB from the address list in the UIB.

Figure 128 on page 517 illustrates the whole of the this process. The example in the figure assumes that you have three PCBs defined in the PSB and want to use the second PCB in the database CALL. Therefore, when setting up the ADDRESS special register of the LINKAGE SECTION group item PCB, the program uses 2 to index the working-storage table, PCB-ADDRESS-LIST. To use the nth PCB, you use the number n to index PCB-ADDRESS-LIST.

```
WORKING-STORAGE SECTION.
77 PCB-CALL PIC X(4) VALUE 'PCB '.
77 GET-HOLD-UNIQUE PIC X(4) VALUE 'GHU '.
77 PSB-NAME PIC X(8) VALUE 'CBLPSB'.
77 SSA1 PIC X(40) VALUE SPACES.
01 DLI-IO-AREA.
02 DLI-IO-AREA1 PIC X(99).
*
LINKAGE SECTION.
COPY DLIUIB.
01 OVERLAY-DLIUIB REDEFINES DLIUIB.
02 PCBADDR USAGE IS POINTER.
02 FILLER PIC XX.
01 PCB-ADDR-LIST.
02 PCB-ADDRESS-LIST USAGE IS POINTER
OCCURS 10 TIMES.
01 PCB.
02 PCB-DBD-NAME PIC X(8).
02 PCB-SEG-LEVEL PIC XX.
02 PCB-STATUS-CODE PIC XX.
*
PROCEDURE DIVISION.
*SCHEDULE THE PSB AND ADDRESS THE UIB
CALL 'CBLTDLI' USING PCB-CALL PSB-NAME ADDRESS OF DLIUIB.
*
*MOVE VALUE OF UIBPCBAL, ADDRESS OF PCB ADDRESS LIST (HELD IN UIB)
*(REDEFINED AS PCBADDR, A POINTER VARIABLE), TO
*ADDRESS SPECIAL REGISTER OF PCB-ADDR-LIST TO PCBADDR.
SET ADDRESS OF PCB-ADDR-LIST TO PCBADDR.
*MOVE VALUE OF SECOND ITEM IN PCB-ADDRESS-LIST TO ADDRESS
SPECIAL
*REGISTER OF PCB, DEFINED IN LINKAGE SECTION.
SET ADDRESS OF PCB TO PCB-ADDRESS-LIST(2).
*PERFORM DATABASE CALLS ......
........
MOVE ........ TO SSA1.
CALL 'CBLTDLI' USING GET-HOLD-UNIQUE PCB DLI-IO-AREA SSA1.
*CHECK SUCCESS OF CALLS .......
IF UIBFCTR IS NOT EQUAL LOW-VALUES THEN
 ...... error diagnostic code
........
IF PCB-STATUS-CODE IS NOT EQUAL SPACES THEN
 ...... error diagnostic code
........
```

Figure 128. Using the DL/I CALL interface

VS COBOL II programs

Language Environment provides support for the execution of programs compiled by the VS COBOL II compiler. The native runtime library for this compiler is not supported. However, this compiler is not Language Environment-conforming (it is a pre-Language Environment compiler), so some restrictions and considerations are associated with its use.

For detailed information about upgrading VS COBOL II programs to Language Environment support, see the Enterprise COBOL for z/OS Compiler and Runtime Migration Guide.

Language Environment callable services

Programs compiled by Language Environment-conforming COBOL compilers can use all Language Environment callable services, either dynamically or statically. However, for CICS applications, the CEEMOUT (dispatch a message) and CEE3DMP (generate dump) services differ, in that the messages

and dumps are sent to the CESE transient data queue rather than to the ddname specified in the MSGFILE runtime option.

VS COBOL II programs can make dynamic calls to the date and time callable services, but no other calls, either static or dynamic, to Language Environment callable services are supported for VS COBOL II programs.

Re-linking VS COBOL II programs

If object modules are not available for re-linking existing VS COBOL II programs to use the runtime support provided by Language Environment , a sample job stream for performing the task is provided in the IGZWRLKA member of the SCEESAMP sample library.

CICS stub

Although COBOL programs linked with the old CICS stub, DFHECI, run under Language Environment , it is advisable to use the DFHELII stub, and it is essential to use the DFHELII stub in a mixed language environment. DFHECI must be link-edited at the top of your application, but DFHELII can be linked anywhere in the application.

Using CEEWUCHA

If you are adapting VS COBOL II programs to use the runtime support provided by Language Environment , the sample user condition handler, CEEWUCHA, supplied by Language Environment in the SCEESAMP library, can be used to advantage. It functions as follows:

- It provides compatibility with existing VS COBOL II applications running under CICS by allowing EXEC CICS HANDLE ABEND LABEL statements to get control when an error detected during runtime occurs.
- It converts all unhandled detected errors during runtime to the corresponding user 1xxx abend issued by VS COBOL II.
- It suppresses all IGZ0014W messages, which are generated when IGZETUN or IGZEOPT is link-edited with a VS COBOL II application. (Performance is better if the programs are not link-edited with IGZETUN or IGZEOPT.)

Ensure that the sample user condition handler, CEEWUCHA, is available at runtime (for example by using the STEPLIB concatenation or LPA). Define the condition handler in the CICS system definition data set (CSD) for your CICS region, rather than using program autoinstall.

Using based addressing with COBOL

COBOL provides a simple method of obtaining addressability to CICS data areas defined in the LINKAGE SECTION using pointer variables and the ADDRESS special register.

CICS application programs need to access data dynamically when the data is in a CICS internal area, and only the address is passed to the program. Examples are:

- CICS areas such as the CWA, TWA, and TCTTE user area (TCTUA), accessed using the ADDRESS command.
- Input data, obtained by EXEC CICS commands such as READ and RECEIVE with the SET option.

The ADDRESS special register holds the address of a record defined in the LINKAGE SECTION with level 01 or 77. This register can be used in the SET option of any command in ADDRESS mode. These commands include GETMAIN, LOAD, READ, and READQ.

Figure 129 on page 519 shows the use of ADDRESS special registers in COBOL. If the records in the READ or REWRITE commands are of fixed length, no LENGTH option is required. This example assumes variable-length records. After the read, you can get the length of the record from the field named in the LENGTH option (here, LRECL-REC1). In the REWRITE command, you must code a LENGTH option if you want to replace the updated record with a record of a different length.

```
WORKING-STORAGE SECTION.
77 LRECL-REC1 PIC S9(4) COMP.
LINKAGE SECTION.
01 REC-1.
02 FLAG1 PIC X.
02 MAIN-DATA PIC X(5000).
02 OPTL-DATA PIC X(1000).
01 REC-2.
02 ...
PROCEDURE DIVISION.
EXEC CICS READ UPDATE...
SET(ADDRESS OF REC-1)
LENGTH(LRECL-REC1)
END-EXEC.
IF FLAG1 EQUAL X'Y'
MOVE OPTL-DATA TO ...

EXEC CICS REWRITE...
FROM(REC-1)
END-EXEC.
```

Figure 129. Addressing CICS data areas in locate mode

Calling subprograms from COBOL programs

In a CICS system, when control is transferred from the active program to an external program, but the transferring program remains active and control can be returned to it, the program to which control is transferred is called a subprogram. In COBOL, there are three ways of transferring control to a subprogram.

EXEC CICS LINK

The calling program contains a command in one of these forms:

```
EXEC CICS LINK PROGRAM('subpgname')
EXEC CICS LINK PROGRAM(
name
)
```

In the first form, the called subprogram is specified as an alphanumeric literal. In the second form, *name* refers to the COBOL data area with length equal to that required for the name of the subprogram.

Static COBOL call

The calling program contains a COBOL statement of the form:

```
CALL 'subpgname'
```

The called subprogram is explicitly named as a literal string.

Dynamic COBOL call

The calling program contains a COBOL statement of the form:

```
CALL identifier
```

The identifier is the name of a COBOL data area that must contain the name of the called subprogram.

For information about the performance implications of using each of these methods to call a subprogram, see the Enterprise COBOL for z/OS Programming Guide, and Enterprise COBOL Version 4 Release 2 Performance Tuning (https://www.ibm.com/support/docview.wss?uid=swg27018287).

COBOL programs can call programs in any language supported by CICS, statically or dynamically. LINK or XCTL are not required for inter-language communication, unless you want to use CICS functions such as COMMAREA. See “Mixing languages in Language Environment” on page 538 for more information about inter-language communication.

The contents of any called or linked subprogram can be any function supported by CICS for the language (including calls to external databases, for example, Db2 and DL/I), with the exception that an assembler language subprogram cannot CALL a lower level subprogram.

Flow of control between programs and subprograms

There are a number of possible flows between COBOL main programs and subprograms.

A **run unit** is a running set of one or more programs that communicate with each other by COBOL static or dynamic CALL statements. In a CICS environment, a run unit is entered at the start of a CICS task, or invoked by a LINK or XCTL command. A run unit can be defined as the execution of a program defined by a PROGRAM resource definition, even though for dynamic CALL, the subsequent PROGRAM definition is needed for the called program. When control is passed by a XCTL command, the program receiving control cannot return control to the calling program by a RETURN command or a GOBACK statement, and is therefore not a subprogram.

Each LINK command creates a new **CICS application logical level** , the called program being at a level one lower than the level of the calling program (CICS is taken to be at level 0). Figure 130 on page 521 shows logical levels and the effect of RETURN commands and CALL statements in linked and called programs.

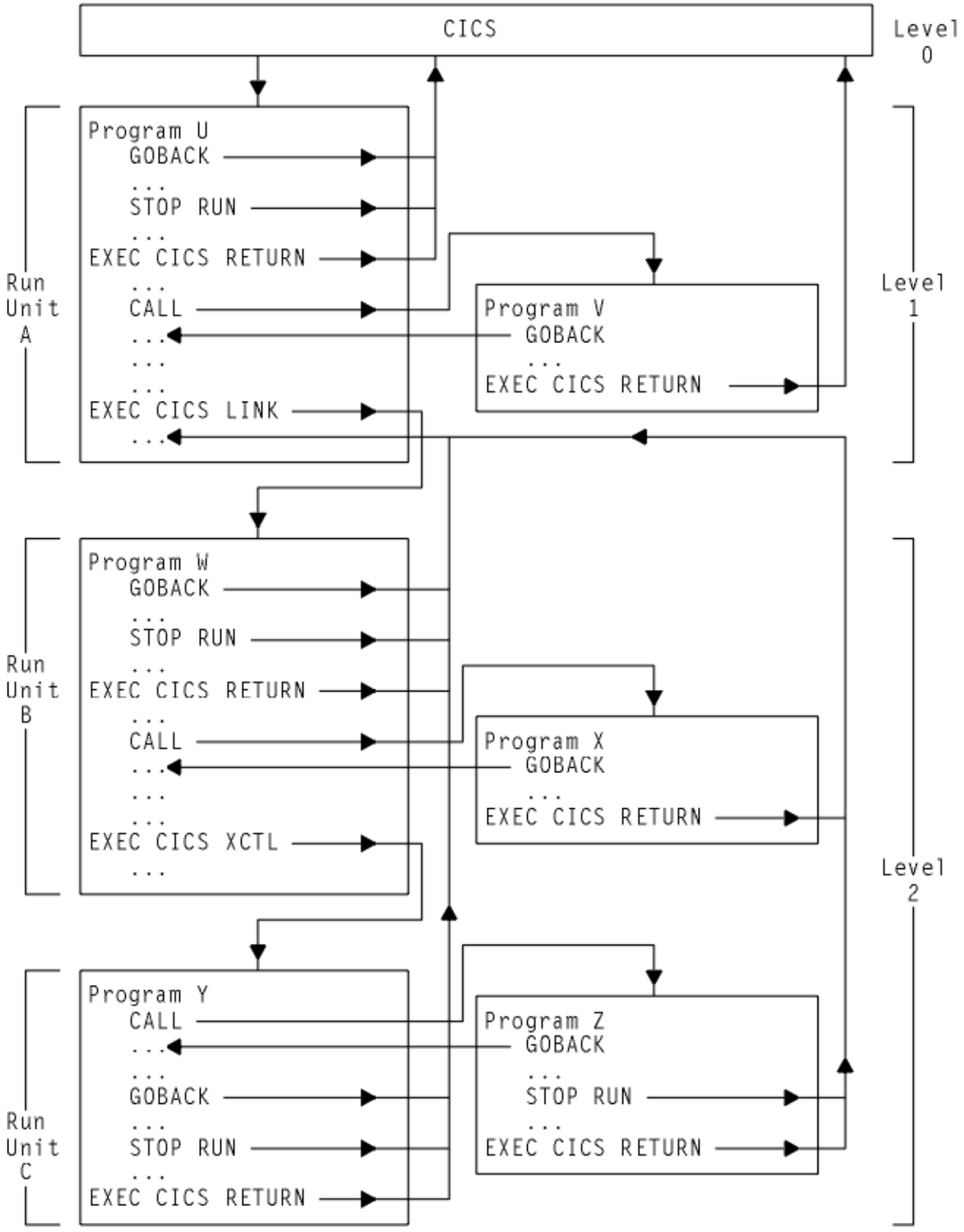


Figure 130. Flow of control between COBOL programs, run units, and CICS

A main, or *level 1* program can use the COBOL GOBACK or STOP RUN statements, or the CICS RETURN command to terminate and return to CICS. It can use a COBOL CALL statement to call a subprogram at the same logical level (level 1), or a CICS LINK command to call a subprogram at a lower logical level.

A called subprogram at level 1 can return to the caller using the COBOL GOBACK statement, or can terminate and return to CICS using EXEC CICS RETURN.

A subprogram executing at level 2 can use the COBOL GOBACK or STOP RUN statements, or the CICS RETURN command to terminate and return to the level 1 calling program. It can use a COBOL CALL statement or a CICS XCTL command to call a subprogram at the same level (level 2). A subprogram called using the COBOL CALL at level 2 can return to the caller (at level 2) using the COBOL GOBACK statement, or can return to the level 1 calling program using EXEC CICS RETURN. A subprogram called using XCTL at level 2 can only return to the level 1 calling program, using GOBACK, STOP RUN or EXEC CICS RETURN.

See "Program linking" on page 147 for more information about program logical levels.

Rules for calling subprograms

These rules describe the requirements and behavior of subprograms called or linked from a COBOL program. The rules which apply depend on how control is transferred to the subprogram, whether by an **EXEC CICS LINK** command, a static COBOL call, or a dynamic COBOL call.

Location of subprogram

EXEC CICS LINK
The subprogram can be remote.

Static or dynamic COBOL call
The subprogram must be local.

Translation

If a compiler with an integrated translator is used, translation is not required.

EXEC CICS LINK
The linked subprogram must be translated if it, or any subprogram invoked from it, contains CICS function.

Static or dynamic COBOL call
The called subprogram must be translated if it contains CICS commands or references to the EXEC interface block (DFHEIBLK) or to the CICS communication area (DFHCOMMAREA).

Compilation

You must always use the NODYNAM compiler option (the default) when you compile a COBOL program that is to run with CICS, even if the program issues dynamic calls.

Link-editing

EXEC CICS LINK
The linked subprogram must be compiled and link-edited as a separate program.

Static COBOL call
The called subprogram must be link-edited with the calling program to form a single load module (but the programs can be compiled separately). This can produce large program modules, and it also stops two programs that call the same program from sharing a copy of that program.

Dynamic COBOL call
The called subprogram must be compiled and link-edited as a separate load module. It can reside in the link pack area or in a library that is shared with other CICS and non-CICS regions at the same time.

CICS CSD entries without program autoinstall

If you use program autoinstall, you do not require an entry in the CSD.

EXEC CICS LINK
The linked subprogram must be defined using RDO. If the linked subprogram is unknown or unavailable, even though autoinstall is active, the LINK fails with the PGMIDERR condition.

Static COBOL call
The calling program must be defined in the CSD. If program A calls program B and then program B attempts to call program A, COBOL issues a message and an abend (1015). The subprogram is part of the calling program so no CSD entry is required.

Dynamic COBOL call
The calling program must be defined in the CSD. If program A calls program B and then program B attempts to call program A, COBOL issues a message and an abend (1015). The called subprogram must be defined in the CSD. If the called subprogram cannot be loaded or is unavailable even though autoinstall is active, COBOL issues a message and abends (1029).

Recursive calls in COBOL

If program A calls program B and program B attempts to call program A, Language Environment issues message IGZ0064S to CEEMSG and an abend (4038).

If program A and program B have the RECURSIVE keyword on the PROGRAM-ID, recursive calls are allowed.

Passing parameters to a subprogram

Data can be passed by any of the standard CICS methods (COMMAREA, TWA, TCTUA, TS queues) if the called or linked subprogram is processed by the CICS translator.

EXEC CICS LINK
If the COMMAREA is used, its address must be passed in the LINK command. If the linked subprogram uses 24-bit addressing, and the COMMAREA is above the 16 MB line, CICS copies it to below the 16 MB line, and recopies it on return.

Static COBOL call
The CALL statement can pass DFHEIBLK and DFHCOMMAREA as the first two parameters, if the called program is to issue EXEC CICS requests, or the called program can issue **EXEC CICS ADDRESS** commands. The COMMAREA is optional but if other parameters are passed, a dummy COMMAREA must also be passed. The rules for nested programs can be different.

Dynamic COBOL call
The CALL statement can pass DFHEIBLK and DFHCOMMAREA as the first two parameters, if the called program is to issue EXEC CICS requests, or the called program can issue **EXEC CICS ADDRESS** commands. The COMMAREA is optional but if other parameters are passed, a dummy COMMAREA must also be passed. If the called subprogram uses 24-bit addressing and any parameter is above the 16MB line, COBOL issues a message and abends (1033) .

Return from a subprogram

EXEC CICS LINK
The linked subprogram must return using either **EXEC CICS RETURN** or a native language return command such as the COBOL statement **GOBACK**.

Static or dynamic COBOL call
The called subprogram must return using a native language return statement such as the COBOL statement **GOBACK** or **EXIT PROGRAM** . The use of **EXEC CICS RETURN** in the called subprogram terminates the calling program.

Storage

EXEC CICS LINK

On each entry to the linked subprogram, a new initialized copy of its WORKING-STORAGE SECTION is provided, and the run unit is reinitialized (in some circumstances, this can cause a performance degradation).

On each entry to the linked subprogram, a new initialized copy of its LOCAL-STORAGE section is provided.

Static or dynamic COBOL call

On the first entry to the called subprogram within a CICS logical level, a new initialized copy of its WORKING-STORAGE SECTION is provided. On subsequent entries to the called subprogram at the same logical level, the same WORKING STORAGE is provided in its last-used state, that is, no storage is freed, acquired, or initialized. If performance is unsatisfactory with LINK commands, COBOL calls might give improved results.

On every entry to the called subprogram in a CICS logical level, a new initialized copy of its LOCAL-STORAGE SECTION is provided.

CICS condition, AID, and abend handling

EXEC CICS LINK

On entry to the called subprogram, no abend or condition handling is active. Within the subprogram, the normal CICS rules apply. In order to establish an abend or condition handling environment, that exists for the duration of the subprogram, a new HANDLE command should be issued on entry to the subprogram. The environment so created remains in effect until either a further HANDLE command is issued, or the subprogram returns control to the caller.

Static or dynamic COBOL call

- If the dynamically called COBOL program abends, with Language Environment and CBLPSHPOP ON, on entry to the called subprogram, no abend or condition handling is active. Within the subprogram, the normal CICS rules apply. On entry to the called subprogram, COBOL issues a PUSH HANDLE to stack the calling program's condition or abend handlers. In order to establish an abend or condition handling environment that exists for the duration of the subprogram, a new HANDLE command should be issued on entry to the subprogram. The environment that this creates remains in effect until either a further HANDLE command is issued or the subprogram returns control to the caller. When control is returned to the calling program from the subprogram, COBOL unstacks the condition and abend handlers using a POP HANDLE.
- If the dynamically called COBOL program abends with CBLPSHPOP OFF, and condition, AID, or abend handling for the calling program is active, the program ends with abend code APC2.
- For a statically called COBOL program, condition, AID, and abend handling remain in effect, irrespective of the setting of CBLPSHPOP.

COBOL2 and COBOL3 translator options

In CICS Transaction Server for z/OS, Version 5 Release 6 , you can choose between the COBOL2 and COBOL3 CICS translator options for COBOL programs.

Modules compiled in earlier CICS releases with the OOCOBOL translator option cannot execute in CICS Transaction Server. The OOCOBOL translator option was used for the older SOM-based (System Object Manager-based) OO COBOL, and runtime support for this form of OO COBOL was withdrawn in z/OS V1.2. The newer Java-based OO COBOL, which is used in Enterprise COBOL, is not supported by the CICS translator.

The COBOL2 option is the default. COBOL2 instructs the translator to translate as COBOL3, but in addition to include declarations of temporary variables for use in EXEC CICS and EXEC DLI requests.

Choose the COBOL2 option if you are re-translating old programs which were written in such a way that they require the use of temporary variables. In particular, note that the use of temporary variables might circumvent errors that would normally occur when an argument value in a program is incorrectly defined.

If you are confident that your programs do not need the translator's temporary variables, you may use COBOL3, which results in smaller working storage. The COBOL3 option includes all features of the older COBOL2 and ANSI85 translator options, except for declarations of temporary variables.

Note: COBOL2 and COBOL3 are mutually exclusive. If you specify both options by different methods, the COBOL3 option is *always* used, regardless of where the two options have been specified. If this happens, the translator issues a warning message.

For general information about translating your program and preparing it for execution, see Chapter 12, "Translation and compilation," on page 577.

CICS translator actions for COBOL programs

These notes describe specific translator action that is taken when the COBOL3 option is used. Processing with the COBOL2 option is the same in all respects, except for declarations of temporary variables.

Literals intervening in blank lines

Blank lines can appear anywhere in a COBOL source program. A blank line contains nothing but spaces between columns 7 and 72 inclusive.

If blank lines occur within literals in a COBOL source program, the translator eliminates them from the translated output but includes them in the translated listing.

Lower case characters

Lower case characters can occur anywhere in any COBOL word, including user-defined names, system names, and reserved words. The translator listing and output preserve the case of COBOL text as entered.

In addition, the translator accepts mixed case in:

- Translator options
- EXEC CICS commands, both for keywords and for arguments to keywords
- CBL and PROCESS statements
- Compiler directives such as EJECT and SKIP1

The translator does not translate lower case text into upper case. Some names in COBOL text, for example file names and transaction IDs, must match with externally defined names. Such names must always be entered in the same case as the external definition.

If you specify the LINKAGE translator option, or allow it to default, a mixed-case version of the EIB structure (DFHEIBLC) is inserted into the LINKAGE SECTION.

Sequence numbers containing any character

In a COBOL source program, the sequence number field can contain any character in the computer's character set. The sequence number fields need not be in any order and need not be unique.

REPLACE statement

COBOL programs can include the REPLACE statement, which allows the replacement of identified text by defined substitution text. The text to be replaced and inserted can be pseudo-text, an identifier, a literal, or a COBOL word. REPLACE statements are processed after COPY statements.

If you process your COBOL source statements with the CICS-supplied translator, the translator accepts REPLACE statements but does not translate text between pseudo-text delimiters, with the exception of CICS built-in functions (DFHRESP and DFHVALUE), which are translated wherever they occur. CICS commands should not be placed between pseudo-text delimiters.

If you use the integrated translator, the translator accepts REPLACE statements and does translate text between pseudo-text delimiters. CICS commands can be placed between pseudo-text delimiters.

Reference modification

Reference modification supports a method of referencing a substring of a character data item by specifying the starting (leftmost) position of the substring in the data item and, optionally, the length of the substring. The acceptable formats are:

```
data-name (leftmost-character-position:)
data-name (leftmost-character-position: length)
```

data-name can be subscripted or qualified or both. *eftmost-character-position* and *length* can be arithmetic expressions. For more information about reference modification, qualification and subscripting, see the Enterprise COBOL for z/OS Language Reference.

The translator accepts reference modification wherever the name of a character variable is permitted in a COBOL program or in an EXEC CICS command.

Note: If a CICS command uses reference modification in defining a data value, it should include a LENGTH option to specify the data length, unless the NOLENGTH translator option is used. Otherwise the translator generates a COBOL call with a LENGTH register reference in the form:

```
LENGTH OF (reference modification)
```

This is rejected by the compiler.

Global variables

The GLOBAL clause is supported. A variable defined with the GLOBAL clause in a top-level program (see “Nested COBOL programs” on page 528) can be referred to in any of its nested programs, whether directly or indirectly contained.

The translator accepts the GLOBAL keyword.

Comma and semicolon as delimiters

A separator comma is a comma followed by a space. A separator semicolon is a semicolon followed by a space. A separator comma or a separator semicolon can be used as a separator wherever a space alone can be used.

The translator accepts the use in COBOL statements of a separator comma or a separator semicolon wherever a space can be used. For example, the translator accepts the statement:

```
IDENTIFICATION; DIVISION
```

The translator does not support the use of the separator comma and separator semicolon as delimiters in EXEC CICS commands. The only acceptable word delimiter in an EXEC CICS command continues to be a space.

Symbolic character definition

Symbolic characters can be defined in the SPECIAL-NAMES paragraph after the ALPHABET clause. A symbolic character is a program-defined word that represents a 1-character figurative constant.

The translator accepts the use of symbolic characters as specified in the standard.

Note: In general, the compiler does not accept the use of figurative constants and symbolic characters as arguments in CALL statements. For this reason, do not use figurative constants or symbolic constants in EXEC CICS commands, which are converted into CALL statements by the translator. There is one exception to this restriction: a figurative constant is acceptable in an EXEC CICS command as an argument to **pass** a value if it is of the correct data type. For example, a numeric figurative constant can be used in the LENGTH option.

Batch compilation for COBOL programs

Separate COBOL programs can be compiled together as one input file. An END PROGRAM header statement terminates each program and is optional for the last program in the batch. The translator accepts separate COBOL programs in a single input file, and interprets END PROGRAM header statements.

Translator options specified as parameters when invoking the translator are in effect for the whole batch, but can be overridden for a unit of compilation by options specified in the CBL or PROCESS card that initiates the unit.

The options for a unit of compilation are determined according to the following order of priority:

1. Options fixed as installation non-user-modifiable options.
2. Options specified in the CBL or PROCESS card that initiates the unit.
3. Options specified when the translator is invoked.
4. Default options.

For more information about compilation, see “Installing application programs” on page 674.

If you are using batch compilation, you must take some additional action to ensure that compilation and linkage editing are successful, as follows:

- Include the compiler NAME option as a parameter in the JCL statement that invokes the compiler or in a CBL statement for each top-level (non-nested) program. This causes the inclusion of a NAME statement at the end of each program. See Figure 131 on page 528 for more information.
- Edit the compiler output to add INCLUDE and ORDER statements for the CICS COBOL stub to each object module. These statements cause the linkage editor to include the stub at the start of each load module. These statements can be anywhere in the module, though by convention they are at the start. You might find it convenient to place them at the end of the module, immediately before each NAME statement. Figure 132 on page 528 shows the output from Figure 131 on page 528 after editing in this way.

For batch compilation you must vary the procedure described in “Installing application programs” on page 674 . The following is a suggested method:

1. Split the supplied cataloged procedure DFHYITVL into two procedures: PROC1 containing the translate and compilation steps (TRN and COB), and PROC2 containing the linkage editor steps COPYLINK and LKED.
2. In PROC1, add the NAME option to the parameters in the EXEC statement for the compiler, which then looks like this:

```
//COB EXEC PGM=IGYCRCTL,REGION=..,
// PARM='....,NAME,....',
```

3. In PROC1, change the name and disposition of the compiler output data set &&LOADSET. At least remove the initial && from the data set name and change the disposition to CATLG. The SYSLIN statement should then read:

```
//SYSLIN DD DSN=LOADSET,DISP=(NEW,CATLG),
// UNIT=&WORK,SPACE=(80,(250,100))
```

4. Run PROC1.

```
 ..................
....program a....
..................
NAME PROGA(R)
..................
..................
....program b....
..................
..................
NAME PROGB(R)
..................
....program c....
..................
NAME PROGC(R)
```

Figure 131. Compiler output before editing

5. Edit the compiler output in the data set LOADSET to add the INCLUDE and ORDER statements as shown in Figure 132 on page 528. If you use large numbers of programs in batches, you should write a simple program or REXX EXEC to insert the ORDER and INCLUDE statements.
6. In PROC2, add a DD statement for the library that includes the CICS stub. The standard name of this library is CICSTS56.CICS .SDFHLOAD. The INCLUDE statement for the stub refers to this library by the DD name. In Figure 132 on page 528 , it is assumed you have used the DD name SYSLIB (or concatenated this library to SYSLIB). The suggested statement is:

```
//SYSLIB DD DSN=
CICSTS56.CICS
.SDFHLOAD,
// DISP=SHR
```

7. In PROC2, replace the SYSLIN concatenation with the single statement:

```
//SYSLIN DD DSN=LOADSET,
// DISP=(OLD,DELETE)
```

   In this statement it is assumed that you have renamed the compiler output data set LOADSET.
8. Run PROC2.

```
 ....program a....
..................
INCLUDE SYSLIB(DFHELII)
ORDER DFHELII
NAME PROGA(R)
..................
..................
....program b....
..................
..................
INCLUDE SYSLIB(DFHELII)
ORDER DFHELII
NAME PROGB(R)
..................
....program c....
..................
INCLUDE SYSLIB(DFHELII)
ORDER DFHELII
NAME PROGC(R)
```

Figure 132. Linkage editor input

Note: You are recommended to use the DFHELII stub, but DFHECI is still supplied, and can be used.

Nested COBOL programs

COBOL programs can contain COBOL programs. Contained programs are included immediately before the END PROGRAM statement of the containing program. A contained program can also be a containing

program, that is, it can itself contain other programs. Each contained or containing program is terminated by an END PROGRAM statement.

For an explanation of valid calls to nested programs and of the COMMON attribute of a nested program, see the Enterprise COBOL for z/OS Customization Guide.

The CICS translator treats top-level and nested programs differently.

The translator translates a top-level program (a program that is not contained by any other program) in the normal way, with one addition. The translator assigns the GLOBAL attribute for all translator-generated variables in the WORKING-STORAGE SECTION.

The translator translates nested or contained programs in a special way as follows:

- A DATA DIVISION and LINKAGE SECTION are added if they do not already exist.
- Declarations for DFHEIBLK (EXEC interface block) and DFHCOMMAREA (communication area) are inserted into the LINKAGE SECTION.
- EXEC CICS commands and CICS built-in functions are translated.
- The PROCEDURE DIVISION header is not modified.
- No translator-generated temporary variables, used for pre-call assignments, are inserted in the WORKING-STORAGE SECTION.

The translator interprets that the input source starts with a top-level program if the first non-comment record is any of the following:

- IDENTIFICATION DIVISION statement
- CBL card
- PROCESS card

If the first record is none of these, the translator treats the input as part of the PROCEDURE DIVISION of a nested program. The first CBL or PROCESS card indicates the start of a top-level program and of a new unit of compilation. Any IDENTIFICATION DIVISION statements that are found before the first top-level program indicate the start of a new nested program.

The practical effect of these rules is that nested programs cannot be held in separate files and translated separately. A top-level program and all its directly and indirectly contained programs constitute a single unit of compilation and must be submitted together to the translator.

Comments in nested programs

The translator treats comments that follow an END PROGRAM statement as belonging to the next program in the input source. Comments that precede an IDENTIFICATION DIVISION statement appear in the listing after the IDENTIFICATION DIVISION statement.

To avoid confusion always place comments:

- After the IDENTIFICATION DIVISION statement that initiates the program to which they refer.
- Before the END PROGRAM statement that terminates the program to which they refer.

If you are using a separate translator

If you are using a separate translator, and not using the integrated CICS translator, for nested programs that contain EXEC CICS commands, you need to explicitly code EIB and COMMAREA on the USING phrases on CALL and on the PROCEDURE DIVISION, as described in this section.

If you are using the integrated CICS translator, this action is **not** necessary for nested programs that contain EXEC CICS commands. The compiler, in effect, declares DFHEIBLK and DFHCOMMAREA as global in the top-level program. This means that explicit coding is not required.

If you are using a separate translator:

1. In each nested program that contains EXEC CICS commands, CICS built-in functions, or references to the EIB or COMMAREA, code DFHEIBLK and DFHCOMMAREA as the first two parameters of the PROCEDURE DIVISION header as follows:

```
PROCEDURE DIVISION USING DFHEIBLK
DFHCOMMAREA PARM1 PARM2 ...
```

2. In every call to a nested program that contains EXEC CICS commands, CICS built-in functions, or references to the EIB or COMMAREA, code DFHEIBLK and DFHCOMMAREA as the first two parameters of the CALL statement as follows:

```
CALL 'PROGA' USING DFHEIBLK
DFHCOMMAREA PARM1 PARM2 ...
```

3. For every call that forms part of the control hierarchy between the top-level program and a nested program that contains EXEC CICS commands, CICS built-in functions, or references to the EIB or COMMAREA, code DFHEIBLK and DFHCOMMAREA as the first two parameters of the CALL statement. In the PROCEDURE DIVISION in the called programs code DFHEIBLK and DFHCOMMAREA. This is necessary to allow addressability to the EIB and COMMAREA to be passed to programs not directly contained by the top-level program.
4. If it is not necessary to insert DFHEIBLK and DFHCOMMAREA in the PROCEDURE DIVISION of a nested program for any of the reasons previously listed, calls to that program should not include DFHEIBLK and COMMAREA in the parameter list of the CALL statement.

An example of a nested program

A unit of compilation consists of a top-level program W and three nested programs, X, Y, and Z, all directly contained by W.

Program W
: During initialization and termination, calls Y and Z to do initial CICS processing and non-CICS file access. Calls X to do main processing.

Program X
: Calls Z for non-CICS file access and Y for CICS processing.

Program Y
: Issues CICS commands. Calls Z for non-CICS file access.

Program Z
: Accesses files in batch mode.

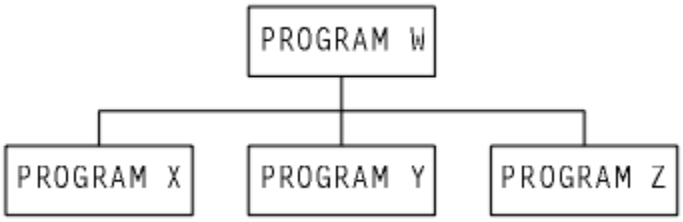


Figure 133. Nested program example—nesting structure

Applying the rules:

- Y must be COMMON to enable a call from X.
- Z must be COMMON to enable calls from X and Y.
- Y issues CICS commands, so if you are using a separate translator:
  - All calls to Y must have DFHEIBLK and a COMMAREA as the first two parameters.
  - Y's PROCEDURE DIVISION header must have DFHEIBLK and DFHCOMMAREA as the first two parameters.

- Though X does not access the EIB or the communication area, it calls Y, which issues CICS commands. Therefore if you are using a separate translator, the call to X must have DFHEIBLK and a COMMAREA as the first two parameters, and X's PROCEDURE DIVISION header must have DFHEIBLK and DFHCOMMAREA as its first two parameters.

Figure 134 on page 531 illustrates these points.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. W...
PROCEDURE DIVISION...
CALL Z...
CALL Y USING DFHEIBLK COMMAREA...
CALL X USING DFHEIBLK COMMAREA...
IDENTIFICATION DIVISION.
PROGRAM-ID. X...
PROCEDURE DIVISION USING DFHEIBLK DFHCOMMAREA..
CALL Z...
CALL Y USING DFHEIBLK COMMAREA...
END PROGRAM X.
IDENTIFICATION DIVISION.
PROGRAM-ID. Y IS COMMON...
PROCEDURE DIVISION USING DFHEIBLK DFHCOMMAREA...
CALL Z...
EXEC CICS.....
END PROGRAM Y.
IDENTIFICATION DIVISION.
PROGRAM-ID. Z IS COMMON...
PROCEDURE DIVISION...
END PROGRAM Z.
END PROGRAM W.
```

Figure 134. Nested program example: coding

Chapter 9. Programming languages and Language Environment

Language Environment, supplied as an element of z/OS, provides a common set of runtime libraries. Language Environment allows you to use only one runtime environment for your applications, regardless of the programming language or system resource needs, because most system dependencies have been removed.

Before the introduction of Language Environment, each high-level language (HLL) provided a separate runtime environment. The runtime libraries provided by Language Environment replace the runtime libraries that were provided with older compilers such as VS COBOL II, OS PL/I, and C/370. The common environment offers two significant advantages:

- You can mix all the languages supported by CICS in a single program.
- The same Language Environment callable services are available to all programs. For example:
  - A linked list created with storage obtained using Language Environment callable services in a PL/I program can be processed later and the storage freed using the callable services from a COBOL routine.
  - The currency symbol used on a series of reports can be set in an Assembler routine, even though the reports themselves are produced by COBOL programs.
  - System messages from programs written in different languages are all sent to the same output destination.

See the z/OS Language Environment Concepts Guide for more information. Because of these advantages, high-level language support under CICS depends upon Language Environment.

CICS supports application programs compiled by a variety of compilers; for a list of compilers that are supported in this release of CICS Transaction Server for z/OS, see Changes to CICS support for application programming languages.

Language Environment does not support AMODE(64) assembler language programs.

Most of the compilers supported by CICS and Language Environment are Language Environment-conforming compilers, meaning that programs compiled by these compilers can take advantage of all Language Environment facilities that are available to a CICS region. CICS and Language Environment also support programs compiled by some pre-Language Environment compilers, which do not conform with Language Environment. However, CICS does not support all the pre-Language Environment compilers that are supported by Language Environment.

Applications compiled and linked with pre-Language Environment compilers might run successfully using the runtime support provided by Language Environment. These applications might not require recompiling or re-link-editing. In some circumstances, you might need to adjust Language Environment runtime options so that the applications run correctly. See the z/OS Language Environment Runtime Application Migration Guide and the *Compiler and Run-Time Migration Guide* for the language in use.

The runtime libraries provided with pre-Language Environment compilers are not supported. Do not include any Language libraries other than the Language Environment libraries in your CICS startup JCL.

When modifying existing application programs, or writing new programs, you must use a compiler supported by Language Environment. Your application programs must be link-edited using the Language Environment SCEELKED library, which means that the resulting application load module can run only under Language Environment.

In CICS you can also create Assembler MAIN programs that conform to Language Environment. For more information about Assembler programs, see Chapter 6, “Developing Assembler language applications,” on page 449.

© Copyright IBM Corp. 1974, 2023

Language Environment callable services

Language Environment provides callable services, which can be accessed by programs running under CICS.

The callable services provided by Language Environment are classified in the following categories:

Storage services
These allow you to allocate and free storage from the Language Environment heaps.

Error handling services
These provide a common method of obtaining information to enable you to process errors.

Message services
These provide a common method of handling and issuing messages.

Date and time
These allow you to read, calculate, and write values representing the date and time. Language Environment offers unique pattern-matching capabilities that let you process almost any date and time format contained in an input record or produced by operating system services.

National language support
These allow you to customize Language Environment output (such as messages, RPTOPTS reports, RPTSTG reports, and dumps) for a given country.

Locale
These allow you to customize culturally-sensitive output for a given national language, country, and codeset by specifying a locale name.

General
These are a set of callable services that are not directly related to a specific Language Environment function, for example, dump.

Mathematical
These allow you to perform standard mathematical computations.

These services are normally **only** available to programs compiled with Language Environment-conforming compilers. As an exception, VS COBOL II programs can make dynamic calls to the date and time callable services, but they cannot make any other dynamic calls or any static calls to Language Environment callable services.

For further information about the details of these services, see the z/OS Language Environment Programming Guide. For information about the syntax required to call any of the services, see the z/OS Language Environment Programming Reference.

Message and dump services

When the Language Environment services CEEMOUT (dispatch a message) and CEE3DMP (generate dump) are running under CICS, both the messages and dumps are sent to a transient data queue called CESE, and not to their usual destinations.

The usual destinations for Language Environment messages and dumps are the *ddname* specified in the MSGFILE runtime option for messages and the *ddname* given in the *fname* argument of the CEE3DMP service for dumps. CICS ignores both of these *ddnames*.

Related concepts

Language Environment abend and condition handling
Language Environment abend handling depends on the use of CICS HANDLE ABEND. User-written condition handlers can be used when a CICS HANDLE ABEND is not active. Language Environment is not involved in the handling of CICS-defined exception conditions, or in the detection of attention identifiers (AIDs).

Dynamic Link Libraries (DLLs)

The z/OS dynamic link library (DLL) facility provides a mechanism for packaging programs and data into load modules (DLLs) that can be accessed from other separate load modules.

Related reference

Mixing languages in Language Environment
You can use Language Environment to build an application that is composed of programs that are written in different high-level source languages and assembler language.

Defining runtime options for Language Environment
Language Environment provides runtime options to control your program's processing. Under CICS , exactly which options apply to the execution of a particular program depends not only on the program, but also on how it is run.

CEEBINT exit for Language Environment
All programs running under Language Environment invoke a subroutine called CEEBINT at program initialization time, just after invocation of the CEEBXITA and CEECSTX exits. The runtime environment is fully operational at this point. Language Environment calls this program the High Level Language (HLL) user exit.

Related information

Language Environment storage
Language Environment uses storage obtained from CICS for each run-unit. When each program is first used, Language Environment tells CICS how much storage the run unit work area (RUWA) requires. The allocation of storage depends on the setting of the CICS system initialization parameter, RUWAPOOL.

Language Environment abend and condition handling

Language Environment abend handling depends on the use of CICS HANDLE ABEND. User-written condition handlers can be used when a CICS HANDLE ABEND is not active. Language Environment is not involved in the handling of CICS-defined exception conditions, or in the detection of attention identifiers (AIDs).

Abend handling

When a CICS application is running under Language Environment , the action taken when a task is scheduled for abnormal termination depends on whether a CICS HANDLE ABEND is active or not active:

- When a HANDLE ABEND is active, the action defined in the CICS HANDLE ABEND takes place. Language Environment condition handling does not gain control for any abends or program interrupts, and any user-written condition handlers that are established by CEEHDLR are ignored.
- When a CICS HANDLE ABEND is not active, Language Environment condition handling gains control for abends and program interrupts if the runtime option TRAP(ON) is specified. Normal Language Environment condition handling is then performed. If TRAP(OFF) is specified, no error handling occurs and the abend proceeds. For details of normal Language Environment condition handling, see the z/OS Language Environment Programming Guide.

User-written Language Environment condition handlers

You can use the Language Environment runtime option USRHDLR to register a user-written condition handler at the highest level. At a lower level, for example after a subroutine CALL, you can use the CEEHDLR service to register a condition handler for that level. This lower level handler is automatically unregistered on return from the lower level. You can explicitly unregister it by using the CEEHDLU service. For an explanation of stack levels, and for details of the USRHDLR runtime option and the CEEHDLR and CEEHDLU services, see the z/OS Language Environment Programming Guide.

If you create a user-written Language Environment condition handler (other than in COBOL), you can use most CICS commands, provided that they are coded with a NOHANDLE, RESP, or RESP2 option, to prevent further conditions being raised during execution of the condition handler. The only commands you cannot use are the following, which must not appear in either the condition handler or any program it calls:

- ABEND

- HANDLE ABEND
- HANDLE AID
- HANDLE CONDITION
- IGNORE CONDITION
- POP HANDLE
- PUSH HANDLE

Unless you use the NOLINKAGE translator option, do not use the CICS translator to translate a COBOL user-written condition handler that you have registered for a routine using the CEEHDLR service. This is because the CICS translator adds two extra arguments to the PROCEDURE DIVISION header of the COBOL program: the EXEC Interface Block (EIB) and the COMMAREA. These arguments do not match the arguments passed by Language Environment . A COBOL condition handler cannot, therefore, contain any CICS commands.

However, a user-written condition handler can call a subroutine to perform CICS commands (and this could be a COBOL routine). If you need to pass arguments to this subroutine, place two dummy arguments before them in the caller. The called subroutine must issue EXEC CICS ADDRESS EIB(DFHEIPTR) before executing any other CICS commands.

For an application to use a user-written Language Environment condition handler, the condition handler must be available at runtime (for example by using the STEPLIB concatenation or LPA). Define such condition handlers in the CICS system definition data set (CSD) for your CICS region, rather than using program autoinstall. This includes the sample user-written condition handler CEEWUCHA.

For full details of the required interface to any Language Environment condition handling routine, see the z/OS Language Environment Programming Guide.

CICS condition and attention identifier (AID) handling

Language Environment condition handling does not alter the behavior of applications that use CICS HANDLE CONDITION or HANDLE AID commands. Language Environment is not involved in the handling of CICS-defined exception conditions, which are raised and handled only by CICS . Similarly, AID detection is a CICS function unaffected by Language Environment.

Related concepts

Language Environment callable services
Language Environment provides callable services, which can be accessed by programs running under CICS.

Dynamic Link Libraries (DLLs)
The z/OS dynamic link library (DLL) facility provides a mechanism for packaging programs and data into load modules (DLLs) that can be accessed from other separate load modules.

Related reference

Mixing languages in Language Environment
You can use Language Environment to build an application that is composed of programs that are written in different high-level source languages and assembler language.

Defining runtime options for Language Environment
Language Environment provides runtime options to control your program's processing. Under CICS , exactly which options apply to the execution of a particular program depends not only on the program, but also on how it is run.

CEEBINT exit for Language Environment
All programs running under Language Environment invoke a subroutine called CEEBINT at program initialization time, just after invocation of the CEEBXITA and CEECSTX exits. The runtime environment is fully operational at this point. Language Environment calls this program the High Level Language (HLL) user exit.

Related information

Language Environment storage

Language Environment uses storage obtained from CICS for each run-unit. When each program is first used, Language Environment tells CICS how much storage the run unit work area (RUWA) requires. The allocation of storage depends on the setting of the CICS system initialization parameter, RUWAPOOL.

Language Environment storage

Language Environment uses storage obtained from CICS for each run-unit. When each program is first used, Language Environment tells CICS how much storage the run unit work area (RUWA) requires. The allocation of storage depends on the setting of the CICS system initialization parameter, RUWAPOOL.

If you specify RUWAPOOL=NO, at the start of each CICS link level, CICS issues a GETMAIN for this storage and passes it to Language Environment to use for its control blocks and for storage areas such as STACK, LIBSTACK, and HEAP. The storage is acquired in the default key specified on the transaction. The storage is freed (using FREEMAIN) when the program terminates.

If you specify RUWAPOOL=YES, the first run of a transaction is the same as with RUWAPOOL=NO, but CICS keeps a history of the total storage for RUWAs that is requested to run the transaction. This means that when the transaction is run again, CICS issues a single GETMAIN for the total storage (and a single FREEMAIN at task end), creating a RUWAPOOL. If the transaction follows the same path, CICS allocates the storage from the RUWAPOOL, and no further GETMAIN has to be issued. If more storage is required for RUWAs because of different or extra CICS links, CICS issues a GETMAIN and updates the history, so that next time the single GETMAIN (and FREEMAIN) is for the larger amount. For transactions that issue a large number of CICS LINK commands, the performance improvement can be considerable.

If you specify the CICS system initialization parameter AUTODST=YES, CICS indicates to Language Environment that it is able to support dynamic storage tuning.

If a program specifies a runtime option of ALL31(OFF) and Language Environment needs to use storage below the 16MB line, two areas of storage are allocated, one below 16MB and one above the 16MB line.

If necessary, any application can obtain CICSDATAKEY or USERDATAKEY storage by using a CICS GETMAIN command. However, a program with an EXECKEY of USER cannot use CICSDATAKEY storage.

Related concepts

Language Environment callable services
Language Environment provides callable services, which can be accessed by programs running under CICS.

Language Environment abend and condition handling
Language Environment abend handling depends on the use of CICS HANDLE ABEND. User-written condition handlers can be used when a CICS HANDLE ABEND is not active. Language Environment is not involved in the handling of CICS-defined exception conditions, or in the detection of attention identifiers (AIDs).

Dynamic Link Libraries (DLLs)
The z/OS dynamic link library (DLL) facility provides a mechanism for packaging programs and data into load modules (DLLs) that can be accessed from other separate load modules.

Related reference

Mixing languages in Language Environment
You can use Language Environment to build an application that is composed of programs that are written in different high-level source languages and assembler language.

Defining runtime options for Language Environment
Language Environment provides runtime options to control your program's processing. Under CICS , exactly which options apply to the execution of a particular program depends not only on the program, but also on how it is run.

CEEBINT exit for Language Environment
All programs running under Language Environment invoke a subroutine called CEEBINT at program initialization time, just after invocation of the CEEBXITA and CEECSTX exits. The runtime environment is

fully operational at this point. Language Environment calls this program the High Level Language (HLL) user exit.

Mixing languages in Language Environment

You can use Language Environment to build an application that is composed of programs that are written in different high-level source languages and assembler language.

Assembler language subroutines called from a high level language (HLL) program are fairly straightforward and common. A subroutine that is called from one HLL, but written in another, needs much more careful consideration and involves interlanguage communication (ILC). Language Environment defines an ILC application as one that is built from two or more HLLs and, optionally, assembler language. For more information, see z/OS Language Environment Writing Interlanguage Communication Applications.

In Language Environment , if there is any ILC in a run unit under CICS , each compile unit must be compiled with a Language Environment-conforming compiler. CICS supports the following HLLs:

- C/C++
- COBOL
- PL/I

The following sections describe the conditions for each pair of HLLs. If your application contains only two HLLs, consult the appropriate section. If your application contains all three HLLs, consult the sections that correspond to each interface in your application.

C/C++ and COBOL

The conditions under which Language Environment supports ILC between routines written in C/C++ and COBOL depend on the following factors:

- Whether the language is C or C++.
- Which COBOL compiler is used and whether DLL is specified as a compiler option.
- Whether the call is static or dynamic.
- Whether the function that is invoked is in the module, or exported from a DLL.
- Whether the program is reentrant.
- What, if any, #pragma linkage statement is in your C program.
- Whether your C program exports functions or variables.
- What, if any, extern statement is in your C++ program.

For information about the effect of these factors, see z/OS Language Environment Writing Interlanguage Communication Applications.

C/C++ and PL/I

Under CICS , if all the components of your C/C++ and PL/I application are reentrant, Language Environment supports ILC between routines compiled by C/C++ and PL/I as follows:

- C/C++ routines can statically call PL/I routines and PL/I routines can statically call C/C++ routines.
- C/C++ routines can fetch() PL/I routines that have OPTIONS(FETCHABLE) specified. If the called routine contains any CICS commands, C/C++ must pass the EIB and the COMMAREA as the first two parameters on the call statement.
- PL/I routines can FETCH only those C/C++ routines that were not processed by the CICS translator. This is because during the dynamic call, certain static fields created by the translator cannot be set correctly.

COBOL and PL/I

Under CICS , Language Environment supports ILC between routines compiled with Language Environment-supported COBOL and PL/I Compilers, as follows:

- COBOL routines can statically call PL/I routines, and PL/I routines can statically call COBOL routines.
- COBOL programs can dynamically call PL/I routines that have OPTIONS(FETCHABLE) specified, and PL/I routines can FETCH COBOL programs.

If the called routine contains any CICS commands, the calling routine must pass the EIB and the COMMAREA as the first two parameters on the CALL statement.

Assembler language

- You can make static or dynamic calls from any Language Environment-conforming HLL program to a Language Environment-conforming assembler language subroutine. Conversely, a Language Environment-conforming assembler language routine can make a static call to any Language Environment-conforming routine, and can dynamically load another routine, either assembler language or HLL, by using either of the Language Environment macros CEEFETCH or CEELOAD.
- You cannot delete (release) an ILC module that was loaded by using CEELOAD.
- You can use the CEERELES macro to release an ILC module that was fetched by using CEEFETCH.
- Use the language that fetched it to delete an assembler language routine. This can be done from C/C++, COBOL, and PL/I, only if there is no ILC with PL/I in the module being released.

Additionally, you can make static calls from any Language Environment-conforming HLL program or assembler language subroutine to a non-conforming assembler language subroutine. However, a non-conforming assembler language routine cannot make a static call to any Language Environment-conforming routine, and cannot fetch or load a conforming routine, because it cannot use the Language Environment macros.

For assembler language to call C or C++, you must include the following statement:

C

```
#pragma linkage(,OS)
```

C++

```
extern "OS"
```

DL/I

If you use DL/I in your ILC application under CICS , calls to DL/I, either by an EXEC DLI statement or by a CALL xxxTDLI, can be made only in programs with the same language as the main program.

Language Environment does not support CALL CEETDLI under CICS.

Related concepts

Language Environment callable services
Language Environment provides callable services, which can be accessed by programs running under CICS.

Language Environment abend and condition handling
Language Environment abend handling depends on the use of CICS HANDLE ABEND. User-written condition handlers can be used when a CICS HANDLE ABEND is not active. Language Environment is not involved in the handling of CICS-defined exception conditions, or in the detection of attention identifiers (AIDs).

Dynamic Link Libraries (DLLs)
The z/OS dynamic link library (DLL) facility provides a mechanism for packaging programs and data into load modules (DLLs) that can be accessed from other separate load modules.

Related reference

Defining runtime options for Language Environment

Language Environment provides runtime options to control your program's processing. Under CICS , exactly which options apply to the execution of a particular program depends not only on the program, but also on how it is run.

CEEBINT exit for Language Environment
All programs running under Language Environment invoke a subroutine called CEEBINT at program initialization time, just after invocation of the CEEBXITA and CEECSTX exits. The runtime environment is fully operational at this point. Language Environment calls this program the High Level Language (HLL) user exit.

Related information

Language Environment storage
Language Environment uses storage obtained from CICS for each run-unit. When each program is first used, Language Environment tells CICS how much storage the run unit work area (RUWA) requires. The allocation of storage depends on the setting of the CICS system initialization parameter, RUWAPOOL.

Dynamic Link Libraries (DLLs)

The z/OS dynamic link library (DLL) facility provides a mechanism for packaging programs and data into load modules (DLLs) that can be accessed from other separate load modules.

A DLL can export symbols representing routines that can be called from outside the DLL, and can import symbols representing routines or data or both in other DLLs, avoiding the need to link the target routines into the same load module as the referencing routine. When an application references a separate DLL for the first time, the system automatically loads the DLL into memory.

You should define all potential DLL executable modules as PROGRAM resources to CICS.

DLL support is available for applications under CICS where the code has been compiled using any of the compilers listed in the z/OS Language Environment Programming Guide. See that information for more information on building and using DLLs.

Related concepts

Language Environment callable services
Language Environment provides callable services, which can be accessed by programs running under CICS.

Language Environment abend and condition handling
Language Environment abend handling depends on the use of CICS HANDLE ABEND. User-written condition handlers can be used when a CICS HANDLE ABEND is not active. Language Environment is not involved in the handling of CICS-defined exception conditions, or in the detection of attention identifiers (AIDs).

Related reference

Mixing languages in Language Environment
You can use Language Environment to build an application that is composed of programs that are written in different high-level source languages and assembler language.

Defining runtime options for Language Environment
Language Environment provides runtime options to control your program's processing. Under CICS , exactly which options apply to the execution of a particular program depends not only on the program, but also on how it is run.

CEEBINT exit for Language Environment
All programs running under Language Environment invoke a subroutine called CEEBINT at program initialization time, just after invocation of the CEEBXITA and CEECSTX exits. The runtime environment is fully operational at this point. Language Environment calls this program the High Level Language (HLL) user exit.

Related information

Language Environment storage

Language Environment uses storage obtained from CICS for each run-unit. When each program is first used, Language Environment tells CICS how much storage the run unit work area (RUWA) requires. The allocation of storage depends on the setting of the CICS system initialization parameter, RUWAPOOL.

Defining runtime options for Language Environment

Language Environment provides runtime options to control your program's processing. Under CICS , exactly which options apply to the execution of a particular program depends not only on the program, but also on how it is run.

Java programs and programs initiated from the Web use the Language Environment preinitialization module, CEEPIPI. This has its own version of the CEEDOPT CSECT and such programs get their runtime options from this CSECT.

For normal CICS tasks, such as those started from a terminal, use any of the following methods listed to set the Language Environment runtime options. For more information about the full order of precedence for Language Environment runtime options see the z/OS Language Environment Programming Guide. The methods are shown in the order in which they are processed. Each setting could be overridden by a following one. This is, in effect, a reverse order of precedence.

1. The CEEDOPT CSECT built into CEECCICS contains the IBM Language Environment default runtime options. You can change these default runtime options by using the CEEWCOPT sample job located in SCEESAMP. This option is supported but using the CEEPRMxx parmlib member to specify runtime options is the preferred and easiest method.
2. The CEEPRMxx parmlib member provides support for the CEECOPT option group which is the preferred method for setting your default Language Environment runtime options for CICS.
3. In the CEEROPT CSECT, where the region-wide default options are located. This CSECT is link-edited into a load module of the same name and placed in a data set in the DFHRPL library concatenation for the CICS job.
4. The user replaceable program DFHAPXPO (applies to XPLINK programs only).
5. In the CEEUOPT CSECT, where user-supplied application program-level runtime options are located. This CSECT is linked with the application program itself.
6. In the application source code using the programming language options statements, as follows:
   - In C programs, through the #pragma runopts statement in the program source. For example:

     ```
     #pragma runopts(rptstg(on))
     ```

   - In PL/I programs, through the PLIXOPT declaration statement within the program. For example:

     ```
     DECLARE PLIXOPT CHARACTER(18) VARYING STATIC EXTERNAL
     INIT('RPTOPTS(ON) NOSTAE');
     ```

   Note: There is no source code mechanism that allows the setting of runtime options within COBOL programs or within C++ programs.
7. In the Language Environment options specified in a debugging profile. For more information, see “Debugging profiles” on page 667.

In most installations, the first method in the previous list is unavailable to application programmers, and the second is often unavailable. However, application programmers can use method 4 or method 5. Choose one method only; do not attempt to use both method 4 and method 5. For information about generating a CEEUOPT CSECT to link with your application, see z/OS Language Environment Customization.

Both CEEDOPT and CEEROPT are able to set any option so that it cannot be overridden by a later specification.

For more information about how to specify Language Environment runtime options and also for their meanings, see z/OS Language Environment Programming Reference.

Runtime options ignored under CICS

Under CICS many of the Language Environment runtime option settings are ignored. These are all the Fortran-only options plus the following:

- ABPERC
- AIXBLD
- CBLOPTS
- CBLQDA
- DEBUG
- EXECOPS
- INTERRUPT
- LIBRARY
- MSGFILE
- NONIPTSTACK
- PLITASKCOUNT
- POSIX (unless XPLINK or Java program)
- RTEREUS
- RTLS
- SIMVRD
- THREADHEAP
- VERSION

Determining which runtime options were used

If you want to know which Language Environment runtime options were in effect when your program ran, specify the option RPTOPTS(ON). When the program ends this produces a list of all the runtime options used. The list is written to the CESE TD queue. The list contains not only the actual settings of the options, but also their origin, that is, whether they are the default for the installation or the region, or whether they were set by the programmer or in one of the exits.

Note: Do not use RPTOPTS(ON) in a production environment. There is significant overhead and it causes a large amount of data to be written to the CESE queue.

Runtime options in child enclaves: performance considerations

Under CICS the execution of a CICS LINK command creates what Language Environment calls a Child Enclave. A new environment is initialized and the child enclave gets its runtime options. These runtime options are independent of those options that existed in the creating enclave.

Frequent use of EXEC CICS LINK, and the individual setting of many runtime options, could affect performance. A static or dynamic call does not incur these overheads. If you must use CEEUOPT to specify options, specifying only those options that are different from the defaults improves performance.

Something similar happens when a CICS XCTL command is executed. In this case we do not get a child enclave, but the existing enclave is terminated and then reinitialized with the runtime options determined for the new program. The same performance considerations apply.

Related concepts

Language Environment callable services
Language Environment provides callable services, which can be accessed by programs running under CICS.

Language Environment abend and condition handling

Language Environment abend handling depends on the use of CICS HANDLE ABEND. User-written condition handlers can be used when a CICS HANDLE ABEND is not active. Language Environment is not involved in the handling of CICS-defined exception conditions, or in the detection of attention identifiers (AIDs).

Dynamic Link Libraries (DLLs)
The z/OS dynamic link library (DLL) facility provides a mechanism for packaging programs and data into load modules (DLLs) that can be accessed from other separate load modules.

Related reference

Mixing languages in Language Environment
You can use Language Environment to build an application that is composed of programs that are written in different high-level source languages and assembler language.

CEEBINT exit for Language Environment
All programs running under Language Environment invoke a subroutine called CEEBINT at program initialization time, just after invocation of the CEEBXITA and CEECSTX exits. The runtime environment is fully operational at this point. Language Environment calls this program the High Level Language (HLL) user exit.

Related information

Language Environment storage
Language Environment uses storage obtained from CICS for each run-unit. When each program is first used, Language Environment tells CICS how much storage the run unit work area (RUWA) requires. The allocation of storage depends on the setting of the CICS system initialization parameter, RUWAPOOL.

CEEBXITA and CEECSTX user exits

These two Language Environment user exits can change some of the Language Environment runtime options.

- Setting the CEEAUE_A_OPTION return parameter of the CEEBXITA user exit can change options (apart from the LIBRARY, RTLS, STACK, and VERSION options).
- In the storage tuning user exit, CEECSTX, the options STACK, LIBSTACK, HEAP, ANYHEAP, and BELOWHEAP can be set.

The exits are called in the order in which they are listed above.

The storage tuning exit CEECSTX, like the CEEROPT CSECT, is region-wide, but CEEBXITA is linked into every program.

Language Environment calls CEEBXITA the assembler exit, because, like CEECSTX, it is invoked before the environment is fully established, and must therefore be coded in assembler language.

Language Environment supplies a sample source version of CEEBXITA in the SCEESAMP library (it returns to its caller for whatever reason it is called). You can use this as it is or modify it for use as the installation default version. However, you can link-edit a specifically tailored version of CEEBXITA with any application program and this version is then used instead of the installation default version. Take great care if you choose this method, because CEEBXITA is invoked for up to five different reasons during the course of program execution, and an application-specific version of CEEBXITA must be capable of handling all these invocations.

If you write your own version of CEEBXITA, you must write it in assembler language . You can use all CICS commands except the ones listed here, provided you specify the NOHANDLE, RESP or RESP2 option, to prevent conditions being raised during the execution of the exit. These are the commands that cannot be used within CEEBXITA, or any routines called by CEEBXITA:

- ABEND
- HANDLE ABEND
- HANDLE AID
- HANDLE CONDITION
- IGNORE CONDITION

- POP HANDLE
- PUSH HANDLE

For more details on both CEEBXITA and CEECSTX, see z/OS Language Environment Customization.

CICSVAR: CICS environment variable

CICS provides an environment variable called CICSVAR to allow the CONCURRENCY and API program attributes to be closely associated with the application program itself. You can specify this environment variable by using the Language Environment runtime option ENVAR.

CICSVAR can be used in a CEEDOPT CSECT to set an installation default, but it is most useful when it is set in a CEEUOPT CSECT that is link-edited with an individual program, or set by a #pragma statement in the source of a C or C++ program, or set by a PLIXOPT statement in a PL/I program. For example, when a program is coded to threadsafe standards, it can be defined as such without changing a PROGRAM resource definition, or it can adhere to an installation-defined naming standard to allow a program autoinstall exit to install it with the correct attributes.

CICSVAR can be used for Language Environment-conforming assembler language, for PL/I, for COBOL, and for C and C++ programs (both those compiled with the XPLINK option, and those compiled without it), if the programs were compiled using a Language Environment-conforming compiler. CICSVAR cannot be used for assembler language programs that are not Language Environment-conforming, or for Java programs.

The use of CICSVAR overrides the settings on a PROGRAM resource definition installed through the standard RDO interfaces, or through program autoinstall. Before the program is run for the first time, an INQUIRE PROGRAM command shows the keyword settings from the program definition. When the application has run once, an INQUIRE PROGRAM command shows the settings with any CICSVAR overrides applied.

Valid values for CICSVAR are QUASIRENT, THREADSAFE, REQUIRED, and OPENAPI.

CICSVAR=QUASIRENT
Results in a program with the attributes CONCURRENCY(QUASIRENT) and APIST(CICSAPI).

CICSVAR=THREADSAFE
Results in a program with the attributes CONCURRENCY(THREADSAFE) and APIST(CICSAPI).

CICSVAR=REQUIRED
Results in a program with the attributes CONCURRENCY(REQUIRED) and APIST(CICSAPI).

CICSVAR=OPENAPI
Results in a program with the attributes CONCURRENCY(REQUIRED) and APIST(OPENAPI).

The following example shows the Language Environment runtime option ENVAR coded in a CEEUOPT CSECT:

```
CEEUOPT CSECT
CEEUOPT AMODE ANY
CEEUOPT RMODE ANY
CEEXOPT ENVAR=('CICSVAR=THREADSAFE')
END
```

This code can be assembled and link-edited into a load module, and then the CEEUOPT load module can be link-edited together with any language program supported by Language Environment.

Alternatively, for C and C++ programs, add the following statement at the start of the program source before any other C statements:

```
#pragma runopts(ENVAR(CICSVAR=THREADSAFE))
```

For PL/I programs, add the following statement following the PL/I MAIN procedure statement:

```
DCL PLIXOPT CHAR(25) VAR STATIC EXTERNAL
INIT('ENVAR(CICSVAR=THREADSAFE)');
```

CEEBINT exit for Language Environment

All programs running under Language Environment invoke a subroutine called CEEBINT at program initialization time, just after invocation of the CEEBXITA and CEECSTX exits. The runtime environment is fully operational at this point. Language Environment calls this program the High Level Language (HLL) user exit.

Language Environment provides a module containing this program in the SCEELKED library (it returns to its caller) and this is, therefore, the installation default version. However, you can also write and link-edit a tailored version in to any program to replace the default.

Ordinary Language Environment coding rules apply to CEEBINT, and you can write it in C, C++, PL/I, or Language Environment-conforming assembler language. CEEBINT applies to COBOL programs just as any others, but it cannot be written in COBOL or call COBOL programs. If CEEBINT introduces a second HLL to a program, the rules for mixing HLLs described in "Mixing languages in Language Environment" on page 538 apply.

For more information on CEEBINT, see the z/OS Language Environment Programming Guide.

Related concepts

Language Environment callable services
Language Environment provides callable services, which can be accessed by programs running under CICS.

Language Environment abend and condition handling
Language Environment abend handling depends on the use of CICS HANDLE ABEND. User-written condition handlers can be used when a CICS HANDLE ABEND is not active. Language Environment is not involved in the handling of CICS-defined exception conditions, or in the detection of attention identifiers (AIDs).

Dynamic Link Libraries (DLLs)
The z/OS dynamic link library (DLL) facility provides a mechanism for packaging programs and data into load modules (DLLs) that can be accessed from other separate load modules.

Related reference

Mixing languages in Language Environment
You can use Language Environment to build an application that is composed of programs that are written in different high-level source languages and assembler language.

Defining runtime options for Language Environment
Language Environment provides runtime options to control your program's processing. Under CICS , exactly which options apply to the execution of a particular program depends not only on the program, but also on how it is run.

Related information

Language Environment storage
Language Environment uses storage obtained from CICS for each run-unit. When each program is first used, Language Environment tells CICS how much storage the run unit work area (RUWA) requires. The allocation of storage depends on the setting of the CICS system initialization parameter, RUWAPOOL.

Chapter 10. Developing PL/I applications

Use this information to help you code, translate, and compile PL/I programs that you want to use as CICS application programs.

Changes to CICS support for application programming languages lists the PL/I compilers that are supported by CICS Transaction Server for z/OS, Version 5 Release 6 , and their service status on z/OS.

All references to PL/I in CICS Transaction Server for z/OS, Version 5 Release 6 documentation imply the use of a supported Language Environment-conforming compiler, unless stated otherwise.

OPTIONS(MAIN) specification

If you specify the OPTIONS(MAIN) option in a PL/I application program, that program can be the first program of a transaction, or control can be passed to it with a LINK or XCTL command.

In PL/I application programs where the OPTIONS(MAIN) option is not specified, it cannot be the first program in a transaction, nor can it have control passed to it by an LINK or XCTL command, but it can be link-edited to a main program.

FLOAT compiler option

For Enterprise PL/I, specifying the FLOAT option controls the use of the additional floating point registers.

- If your program makes little or no use of floating point, specify the FLOAT(NOAFP) option. The program uses the traditional four floating point registers, and has less work to do when saving registers.
- If your program makes significant use of floating point, specify the FLOAT(AFP) option or the FLOAT(NOVOLATILE) option. The program can use all 16 floating point registers, and CICS preserves the floating point registers used by the program.
- If you specify the FLOAT(AFP(VOLATILE)) option, both CICS and PL/I preserve the floating point registers. Extra code is generated and performance can be affected as a result.

PL/I programming restrictions and requirements

Some restrictions and requirements apply to a PL/I program that is used as a CICS application program.

Functions and statements that cannot be used

- You cannot use the following multitasking built-in functions:

  ```
  COMPLETION
  PRIORITY
  STATUS
  ```

- You cannot use the following multitasking options:

  ```
  EVENT
  PRIORITY
  TASK
  ```

- You should not use the following PL/I statements:

  ```
  CLOSE
  DELAY
  DELETE
  DISPLAY
  EXIT
  GET
  ```

© Copyright IBM Corp. 1974, 2023

```
HALT
LOCATE
OPEN
PUT
READ
REWRITE
STOP
WRITE
UNLOCK
```

The FETCH and RELEASE statements are supported. EXEC CICS commands are provided for the storage and retrieval of data, and for communication with terminals. However, you can use CLOSE, PUT, and OPEN for SYSPRINT.

- You cannot use PL/I Sort/Merge.
- You cannot use static storage (except for read-only data).

Coding requirements

- If you declare a variable with the STATIC attribute and EXTERNAL attribute, you must also include the INITIAL attribute. If you do not, such a declaration generates a common CSECT that cannot be handled by CICS.
- Do not define variables or structures with variable names that are the same as variable names generated by the translator. These begin with DFH. Use care with the LIKE keyword to avoid implicitly generating such variable names.
- All PROCEDURE statements must be in uppercase, except for the PROCEDURE name, which can be in lowercase.
- The suboptions of the XOPTS option of the *PROCESS statement must be in uppercase.
- You cannot use the PL/I 48-character set option in EXEC CICS statements.
- If a CICS command uses the SUBSTR built-in function in defining a data value, it should include a LENGTH option to specify the data length, unless the translator option NOLENGTH is specified. If it does not, the translator generates a PL/I call including an invocation of the CSTG built-in function in the following form:

```
CSTG(SUBSTR(..,..,..))
```

This call is rejected by the compiler.

64-bit addressing

64-bit addressing mode is not supported for PL/I programs.

64-bit residency

CICS does not support 64-bit residency mode (RMODE(64)) and treats any RMODE(64) programs as RMODE(31). That is, RMODE(64) programs are loaded into 31-bit (above-the-line) storage, not 64-bit (above-the-bar) storage.

Language Environment coding requirements for PL/I applications

All PL/I programs are executed under the runtime support provided by Language Environment . There are some additional coding requirements compared to pre-Language Environment PL/I programs.

Language Environment runtime options, if needed, can be specified in a **plixopt** character string. See “ Defining runtime options for Language Environment ” on page 541 and the z/OS Language Environment Programming Reference for information about customizing runtime options.

If you are converting a PL/I program that was previously compiled with a non- Language Environment -conforming compiler, you must ensure that NOSTAE and NOSPIE are not specified in a **plixopt** string, because either of these will cause Language Environment to set TRAP (OFF). TRAP (ON) must be in effect for applications to run successfully.

Entry point

CEESTART is the only entry point for PL/I applications running under Language Environment . This entry point is set for programs compiled using Language Environment-conforming compilers.

You can re-link object modules produced by non-Language Environment-conforming compilers for running under Language Environment by using the following linkage-editor statements:

```
 INCLUDE SYSLIB(CEESTART)
INCLUDE SYSLIB(CEESG010)
INCLUDE SYSLIB(DFHELII)
REPLACE PLISTART
CHANGE PLIMAIN(CEEMAIN)
INCLUDE mainprog
INCLUDE subprog1
.......
.......
ORDER CEESTART
ENTRY CEESTART
NAME progname(R)
```

The INCLUDE statement for the object modules must come immediately after the CHANGE statement. There is also a requirement under Language Environment that the main program must be included before any subroutines. (This requirement did not exist for modules produced by non-conforming compilers.)

For Enterprise PL/I programs that are compiled with OPTIONS(FETCHABLE), the binder ENTRY statement must be the name of the PROCEDURE.

Re-link utility for PL/I

If you have only the load module for a CICS program compiled by a non-conforming compiler, there is a file of linkage editor input, IBMWRLKC, specifically for CICS programs, located in the sample library SCEESAMP. This input file replaces OS PL/I library routines in a non-conforming executable program with Language Environment routines.

For more information about using IBMWRLKC, see the relevant version of the *Compiler and Runtime Migration Guide* in Enterprise PL/I for z/OS product information.

Communicating between conforming and non-conforming PL/I routines

Language Environment-conforming PL/I programs can CALL a program that appears in a FETCH or RELEASE statement and can RELEASE it subsequently.

You can link-edit non-Language Environment-conforming PL/I subroutines with a Language Environment-conforming main program.

Static calls are supported from any version of PL/I, but dynamic calls are supported only from Language Environment-conforming procedures.

Called subroutines can issue CICS commands if the address of the EIB is available in the subroutine. You can achieve this either by passing the address of the EIB to the subroutine, or by coding EXEC CICS ADDRESS EIB(DFHEIPTR) in the subroutine before issuing any other CICS commands.

Abend handling

If a CICS PL/I program abends under Language Environment , your CICS abend handlers are given a Language Environment abend code, rather than a PL/I abend code.

To avoid changing your programs, you can modify the sample user condition handler, CEEWUCHA, supplied by Language Environment in the SCEESAMP library. This user condition handler can be made

to return PL/I abend codes instead of the Language Environment codes. Use the USRHDLR runtime option to register it to do this. For details of this option see the z/OS Language Environment Programming Guide.

Ensure that the sample user condition handler, CEEWUCHA, is available at runtime (for example by using the STEPLIB concatenation or LPA). Define the condition handler in the CICS system definition data set (CSD) for your CICS region, rather than using program autoinstall.

Fetched PL/I routines

To enable a PL/I procedure to be fetched, code the option FETCHABLE in the OPTIONS on the PROCEDURE statement.

The FETCHABLE option indicates that the procedure can only be invoked dynamically. An OPTIONS(MAIN) procedure cannot be fetched; FETCHABLE and MAIN are mutually exclusive options.

For Enterprise PL/I programs that are compiled with OPTIONS(FETCHABLE), the binder ENTRY statement must be the name of the PROCEDURE.

Treat the FETCHABLE procedure like a normal CICS program: that is, link-edited with any required subroutines, placed in the CICS application program library, defined, and installed as a program, either in the CSD or using program autoinstall.

Language Environment-conforming PL/I programs can CALL a program that appears in a FETCH or RELEASE statement and can RELEASE it subsequently.

There were some restrictions on the PL/I for MVS & VM statements that could be used in a fetched procedure; many of the restrictions were removed with VisualAge® PL/I. See the see the relevant version of the *Compiler and Runtime Migration Guide* in Enterprise PL/I for z/OS product information.

No special considerations apply to the use of FETCH when both the fetching and the fetched programs have the same AMODE attribute. Language Environment, however, also supports the fetching of a load module that has an AMODE attribute different to the program issuing the FETCH. In this case, Language Environment performs the AMODE switch, and the following constraints apply:

- If any fetched module is to execute in 24-bit addressing mode, the fetching module must have the RMODE(24) attribute regardless of its AMODE attribute.
- Any variables passed to a fetched routine must be addressable in the AMODE of the fetched procedure.

Chapter 11. Developing for recovery

When you are planning aspects of recovery, you must consider your applications, system definitions, internal documentation, and test plans.

Application design considerations

Think about recoverability as early as possible during the application design stages. This topic covers a number of aspects of design planning to consider.

Questions relating to recovery requirements

For ease of presentation, the following questions assume a single application.

Note: If a new application is added to an existing system, the effects of the addition on the whole system need to be considered.

Question 1: Does the application update data in the system?
If the application is to perform **no** updating (that is, it is an inquiry-only application), recovery and restart functions are not needed within CICS. (But you should take backup copies of non-updated data sets in case they become unreadable.) The remaining questions assume that the application does perform updates.

Question 2: Does this application update data sets that other online applications access?
If yes, does the business require updates to be made **online** , and then to be immediately available to other applications—that is, as soon as the application has made them? This could be a requirement in an online order entry system where it is vital for inventory data sets (including databases) to be as up-to-date as possible for use by other applications at all times.

Alternatively, can updates be stored temporarily and used to modify the data set(s) later—perhaps using offline batch programs? This might be acceptable for an application that records only data not needed immediately by other applications.

Question 3: Does this application update data sets that batch applications access?
If yes, establish whether the batch applications are to access the data sets concurrently with the online applications. If accesses made by the batch applications are limited to read-only, the data sets can be shared between online and batch applications, although read integrity may not be guaranteed. If you intend to update data sets concurrently from both online and batch applications, consider using DL/I or Db2, which ensure both read and write integrity.

Question 4: Does the application access any confidential data?
Files that contain confidential data, and the applications having access to those files, must be clearly identified at this stage. You may need to ensure that only authorized users may access confidential data when service is resumed after a failure, by asking for re-identification in a sign-on message.

Question 5: If a data set becomes unusable, should all applications be terminated while recovery is performed?
If degraded service to any application must be preserved while recovery of the data set takes place, you will need to include procedures to do this.

Question 6: Which of the files to be updated are to be regarded as vital?
Identify any files that are so vital to the business that they must always be recoverable.

Question 7: How important is data integrity, compared to availability?
Consider how long the business can afford to wait for a record that is locked, and weigh this against the risks to data integrity if the normal resynchronization process is overridden.

The acceptable waiting time will vary depending on the value of the data, and the number of users whom you expect to be affected. If the data is very valuable or infrequently accessed, the acceptable

[1] In the context of these questions, the term “data sets” includes databases.

© Copyright IBM Corp. 1974, 2023

waiting time will be longer than if the data is of low value or accessed by many business-critical processes.

Question 8: How long can the business tolerate being unable to use the application in the event of a failure?

Indicate (approximately) the maximum time that the business can allow the system to be out of service after a failure. Is it minutes or hours? The time allowed may have to be negotiated according to the types of failure and the ways in which the business can continue without the online application.

Question 9: How is the user to continue or restart entering data after a failure?

This is an important part of a recovery requirements statement because it can affect the amount of programming required. The terminal user's restart procedure will depend largely on what is feasible—for example:

- Must the user be able to continue business by other means—for example, manually?
- Does the user still have source material (papers, documents) that allow the continued entry (or reentry) of data? If the source material is transitory (received over the telephone, for example), more complex procedures may be required.
- Even if the user still has the source material, does the quantity of data preclude its reentry?

Such factors define the point where the user restarts work. This could be at a point that is as close as possible to the point reached before the system failure. The best point could be determined with the aid of a progress transaction, or it could be at some point earlier in the application—even at the start of the transaction. Note: a progress transaction here means one that enables users to determine the last actions performed by the application on their behalf.

These considerations should be in the external design statement.

Question 10: During what periods of the day do users expect online applications to be available?

This is an important consideration when applications (online and batch) require so much of the available computer time that difficulties can arise in scheduling precautionary work for recovery (taking backup copies, for example). See The RLS quiesce and unquiesce functions.

Validate the recovery requirements statement

After considering the questions relevant to your recovery requirements, produce a formal statement of application and recovery requirements.

Before any design or programming work begins, all interested parties should agree on the statement—including:

- Those responsible for **business management**
- Those responsible for **data management**
- Those who are to **use** the application—including the end users, and those responsible for computer and online system operation

Designing the user's restart procedure

Decide how the user is to restart work on the application after a system failure.

About this task

Points to consider are:

- The need for users to re-identify themselves to the system in a sign-on message (dictated by security requirements, as discussed under question 4 in "Questions relating to recovery requirements" on page 551).
- The availability of appropriate information for users, so that they know what work has and has not been done. Consider the possibility of a progress transaction.

- How much or how little rekeying will be needed when resuming work (dictated by the feasibility of rekeying data, as discussed under question 9 in "Questions relating to recovery requirements" on page 551).

When designing the user's restart procedure (including the progress transaction, if used) include precautions to ensure that each input data item is processed once only.

User's standby procedures

Decide how application work might continue in the event of a prolonged failure of the system.

For example, for an order-entry application, it might be practical (for a limited time) to continue taking orders offline—by manual methods. If you plan such an approach, specify how the offline data is to be subsequently entered into the system; it might be necessary to provide a catch-up function.

Note: If the user is working with an intelligent workstation or a terminal attached to a programmable controller, it may be possible to continue gathering data without access to a CICS region on an MVS host system.

Communications between application and user

For each application, specify the type of terminal the user is to work with.

Decide if you will provide special procedures to overcome communication problems; for example:

- Allow the user to continue work on an alternative terminal (but with appropriate security precautions, such as signing on again).
- In cases where the user's terminal is attached to a programmable controller, determine the recovery actions that controller (or the program in it) is capable of providing.
- If a user's printer becomes unusable (because of hardware or communication problems), consider the use of alternatives, such as the computer center's printer, as a standby.

Security

Decide the security procedures for an emergency restart or a break in communications. For example, when confidential data is at risk, specify that the users should sign on again and have their passwords rechecked.

Bear in mind the security requirements when a user needs to use an alternative terminal if a failure is confined to one terminal (or to a few terminals).

Note: The sign-on state of a user is not retained after a persistent sessions restart.

System definitions for recovery-related functions

You are recommended to use a number of system definitions to ensure that your CICS regions provide the required recovery and restart support for your CICS applications.

System recovery table (SRT)

You are recommended to specify a system recovery table on the **SRT** system initialization parameter. If you do not specify a table suffix on this system initialization parameter, it defaults to YES, which means that CICS tries to load an unsuffixed table (which probably won't exist in your load libraries). There is a pregenerated sample table, DFHSRT1$, supplied in CICSTS56.CICS .SDFHLOAD, and if this is adequate for your needs, specify SRT=1$. This table adds some extra system abend codes to the built-in list that CICS handles automatically, even if you define only the basic DFHSRT TYPE=INITIAL and TYPE=FINAL macros.

If you want to add additional system or user entries of your own, modify the sample table. For information about modifying an SRT, see System recovery table (SRT) .

Resource definitions for recovery

Ensure that your CICS region includes DFHLIST as one of the lists specified on the **GRPLIST** system initialization parameter. Included in DFHLIST are the following groups, which provide basic recovery functions:

- DFHRSEND
- DFHSTAND
- DFHVTAM

For information about the contents of these groups, see Supplied resource definitions, groups, and lists . For information about individual resource recoverability, see Configuring for recovery of CICS-managed resources.

System log streams and general log streams

Defining system log streams for each CICS region is the most fundamental of all the recovery requirements. A system log is mandatory to maintain data integrity in the event of transaction or system abends, and to enable CICS to perform warm and emergency restarts.

CICS uses the services of the MVS system logger for all its system and general logging requirements. The CICS log manager writes system log and general log data to log streams defined to the MVS system logger. For more information, see Logging and journaling.

Files

For VSAM files defined to be accessed in RLS mode, define the recovery attributes in the ICF catalog, using IDCAMS. For VSAM files defined to be accessed in non-RLS mode, you can define the recovery attributes in the CSD file resource definition, or in the ICF catalog, providing your level of DFSMS supports this. For BDAM files, you define the recovery attributes in the FCT.

Transient data queues

If you want your intrapartition queues to be recoverable, specify the recovery option in the definition for each queue. See SESSIONS attributes for information about defining the recovery option, RECOVSTATUS.

Temporary storage queues

If you want your temporary storage queues to be recoverable, specify the RECOVERY(YES) option in their TSMODEL resource definitions. See TSMODEL attributes for information about defining the recovery option.

Program list table (PLT)

Use the DFHPLT macro to create program list tables that name each program executed during initialization or controlled shutdown of CICS.

You specify the **PLTPI** system initialization parameter for the PLT to be used at initialization, and the **PLTSD** system initialization parameter for the PLT to be used at shutdown. You can also specify the shutdown PLT on the CICS shutdown command.

If you have global user exit programs that are invoked for recovery purposes, they must be enabled during the second stage of CICS initialization. You can enable these global user exit recovery programs in application programs specified in the first part of the PLTPI.

See Program list table (PLT) for information about defining program list tables.

Transaction list table (XLT)

There are two ways you can specify transactions that can be initiated from a terminal during the first quiesce stage of normal shutdown:

- Use the DFHXLT macro to create a transaction list table that names the transactions.
- Specify the SHUTDOWN(ENABLED) attribute on the transaction resource definition.

Documentation and test plans

During internal design, consider how to document and test the defined recovery and restart programs, exits, and procedures.

Recovery and restart programs and procedures usually relate to exceptional conditions, and can therefore be more difficult to test than those that handle normal conditions. However, they should be tested as far as possible, to ensure that they handle the functions for which they are designed.

CICS facilities, such as the execution diagnostic facility (CEDF) and command interpreter (CECI), can help to create exception conditions and to interpret program and system reactions to those conditions.

The installed CICS system, application programs, operators, and terminal users must be able to cope with exception conditions. Therefore, the designer and implementer need to forecast the exceptional conditions that can be expected, and then document the operator and user actions in the process of recovery. Documentation should include escape procedures for problems or errors that persist.

Conditions that need documented procedures include the following:

- Power® failure in the processor
- Failure of CICS
- Physical failure of data sets
- Transaction abends
- Communication failures, such as the loss of telephone lines, or a printer that is out of service
- Shunted units of work that are caused by communication or backout failures

It is essential that recovery and restart procedures are tested and rehearsed in a controlled environment by everyone who might have to cope with a failure. This is especially important in installations that have temporary operators.

Programming for recovery

When you are designing your application programs, you can include recovery facilities that are provided by CICS; for example, you can use global user exits for backout recovery.

Designing applications for recovery

In this context, **application** refers to a set of one or more **transactions** designed to fulfill the particular needs of the user organization. A transaction refers to a set of actions within an application which the designer chooses to regard as an entity. It corresponds to a unit of execution, which is typically started in a CICS region by invoking the transaction by its identifier from a terminal attached to CICS.

As an application designer, you must decide how (if at all) to subdivide an application into transactions, and whether the transactions should consist of just one unit of work, or more than one.

Ideally, but not necessarily, a transaction would correspond to a unit of work. Dividing the business application into units of work that correspond to transactions simplifies the entire recovery process.

An example of a typical business application is an order-entry system. A typical order-entry application includes all the processes needed to handle one order from a customer, designed as a set of processing units, as follows:

1. Check the customer’s name and address and allocate an order number.
2. Record the details of ordered items and update inventory files.
3. Print the invoice and shipping documents.

Depending on the agreed recovery requirements statement, you could design noting details of ordered items and updating files either as one large transaction or as several transactions, with one transaction for each item within the order.

Splitting the application into transactions

Specify how to divide the application into transactions.

Procedure

1. Name each transaction, and describe its function in terms that the terminal user can understand.

   Your application could include transactions to recover from failures, such as:

   - *Progress transaction* , to check on progress through the application. Such a function could be used after a transaction failure or after an emergency restart, as well as at any time during normal operation. For example, it could be designed to find the correct restart point at which the terminal user should recommence the interrupted work. This would be particularly relevant in a pseudo-conversation.
   - *Catch-up function* , for entering data that the user might have been forced to accumulate by other means during a system failure.
2. Specify the files and databases that can be accessed in each processing unit.

   Of the files and databases that can be accessed, specify those that are to be updated as distinct from those that are only to be read.
3. Specify how to apply the updates for those files and databases that can be updated by an application processing unit.

   Factors to consider here include the consistency and the immediacy of updates.

   a) Specify which, if any, updates must happen in step with each other to ensure integrity of data. For example, in an order-entry application, it might be necessary to ensure that a quantity subtracted from the inventory file is, at the same time, added to the to-be-shipped file.

   b) Specify when newly entered data must or can be applied to the files or databases:

      - The application processing unit updates the files and databases as soon as the data is accepted from the user.
      - The application processing unit accumulates updates for later action; for example, by a later processing unit within the same application or a batch application that runs overnight. If you choose the batch option, make sure that there is enough time for the batch work to complete the number of updates.

   Use this information when deciding on the internal design of application processing units.
4. Specify what data needs to be passed from one application processing unit to another.
   For example, in an order-entry application, one processing unit might accumulate order items. Another separate processing unit might update the inventory file. Clearly, there is a need here for the data accumulated by the first processing unit to be passed to the second.

   Use this information when deciding what resources are required by each processing unit.

Relationships between processing units

Specify what data needs to be passed from one application processing unit to another.

For example, in an order-entry application, one processing unit may accumulate order items. Another, separate, processing unit may update the inventory file. Clearly, there is a need here for the data accumulated by the first processing unit to be passed to the second.

This information is needed when deciding what resources are required by each processing unit (see “Mechanisms for passing data between transactions” on page 558).

SAA-compatible applications

The resource recovery element of the Systems Application Architecture® (SAA) common programming interface (CPI) provides an alternative to the standard CICS application program interface (API) if you need to implement SAA-compatible applications.

The resource recovery facilities provided by the CICS implementation of the SAA resource recovery interface are the same as those provided by CICS API. Therefore, you have to change from CICS API to SAA resource recovery commands only if your application needs to be SAA-compatible.

To use the SAA resource recovery interface, you need to include SAA resource recovery commands in your applications in place of **EXEC CICS SYNCPOINT** commands. This book refers only to CICS API resource recovery commands; for information about the SAA resource recovery interface, see Systems Application Architecture Common Programming Interface Resource Recovery Reference.

Program design

This section tells you how to design your programs to use the CICS recovery facilities effectively.

Dividing transactions into units of work

You must decide how to implement application processing units in terms of transactions, units of work, and programs.

About this task

You are recommended to plan your application processing units using the following advice:

Procedure

1. In programs that support a dialog with the user, consider implementing each unit of work to include only a single terminal read and a single terminal write.

   Using this approach can simplify the user restart procedures (see also “Processing dialogs with users” on page 558).

   Short units of work are preferable for several reasons:

   - Data resources are locked for a shorter time. This reduces the chance of other tasks having to wait for the resource to be freed.
   - Backout processing time (in dynamic transaction backout or emergency restart) is shortened.
   - The user has less to re-enter when a transaction restarts after a failure.

     In applications for which little or no re-keying is feasible (discussed in question 9 under “Questions relating to recovery requirements” on page 551), short units of work are essential so that all entered data is **committed** as soon as possible.
2. Consider the recovery/restart implications when deciding whether to divide a transaction into many units of work.

   CICS functions such as dynamic transaction backout and transaction restart work most efficiently for transactions that have only one unit of work. But there can be situations in which multiple-unit of work transactions are necessary, for example if a set of file or database updates must be irrevocably committed in **one** unit of work, but the transaction is to continue with one or more units of work for further processing.
3. Where file or database updates must be kept in step, make sure that your application does them in the same unit of work.

   This approach ensures that those updates will all be committed together or, in the event of the unit of work being interrupted, the updates will back out together to a consistent state.

Processing dialogs with users

An application may require several interactions (input and output) with the user.

CICS provides the following basic techniques for program design for use in such situations:

- Conversational processing
- Pseudo-conversational processing

Conversational processing

With conversational processing, the transaction continues to run as a task across all terminal interactions, including the time it takes for the user to read output and enter input.

While it runs, the task retains resources that may be needed by other tasks. For example:

- The task occupies storage, and locks database records, for a considerable period of time. Also, in the event of a failure and subsequent backout, all the updates to files and databases made up to the moment of failure have to be backed out (unless the transaction has been subdivided into units of work).
- If the transaction uses DL/I, and the number of scheduled PSBs reaches the maximum allowed, tasks needing to schedule further PSBs have to wait.

Conversational processing is not generally favored, but may be required where several file or database updates made by several interactions with the user must be related to each other—that is, they must all be committed together, or all backed out together, in order to maintain data integrity.

Pseudoconversational processing

In pseudoconversational processing, successive terminal interactions with the user are processed as separate tasks, usually consisting of one unit of work each.

This approach can result in a requirement to communicate between tasks or transactions (see “Mechanisms for passing data between transactions” on page 558) and the application programming can be a little more complex than for conversational processing.

However, at the end of each task, the updates are committed, and the resources associated with the task are released for use by other tasks. For this reason, pseudoconversational transactions are generally preferred to the conversational type.

When several terminal interactions with the user are related to each other, data for updates must accumulate on a recoverable resource and then be applied to the database in a single task; for example, in the last interaction of a conversation. In the event of a failure, emergency restart or dynamic transaction backout would back out only the updates made during that individual step; the application is responsible for restarting at the appropriate point in the conversation, which might involve recreating a screen format.

However, other tasks might try to update the database between the time when update information is accepted and the time when it is applied to the database. Design your application to ensure that no other application can update the database at a time when it would corrupt the updating by your own application.

Mechanisms for passing data between transactions

In those applications where one transaction needs to access working data created by a previous transaction, you must decide which mechanism will pass the data between the transactions.

You have two options for passing data between transactions:

- Main storage areas
- CICS recoverable resources

Main storage areas

The advantages of main storage areas are realized only where recovery is not important, or when passing data between programs servicing the same task.

Main storage areas that you can use to pass data between transactions include:

- The communication area (COMMAREA)
- The common work area (CWA)
- Temporary storage (main)
- The terminal control table user area (TCTUA)

CICS does not log changes to these areas (except as noted later in this section). Therefore, in the event of an uncontrolled shutdown, data stored in any of these areas is lost, which makes them unsuitable for applications needing to retain data between transactions across an emergency restart. Also, some of these storage areas can cause inter-transaction affinities, which are a hindrance to dynamic transaction routing. To avoid inter-transaction affinities, use either a COMMAREA or the TCTUA. For information about intertransaction affinities, see Affinity .

Design programs so that they do not rely on the presence or absence of data in the COMMAREA to indicate whether or not control has been passed to the program for the first time (for example, by testing for a data length of zero). Consider the abend of a transaction where dynamic transaction backout and automatic restart are specified. After the abend, a COMMAREA could be passed to the next transaction from the terminal, even though the new transaction is unrelated. Similar considerations apply to the terminal control table user area (TCTUA).

CICS recoverable resources

You can use the following resources that are recoverable by backout for communication between transactions:

Temporary storage (auxiliary)

You can use a temporary storage item to communicate between transactions. For this purpose, the temporary storage item needs to be unique to the terminal ID. If the terminal becomes unavailable, the transaction sequence is interrupted until the terminal is again available.

The named temporary storage queue can be read and reread, but the application program must delete it when it is no longer needed to communicate between a sequence of transactions.

Transient data queues

Transient data (intrapartition) is similar to temporary storage (auxiliary) for communicating between transactions. The main difference is that you can read each record in a transient data queue only once, after which the record is no longer available.

Transient data must be specified as **logically** recoverable to achieve backout to the start of any in-flight unit of work.

User files and DL/I and Db2 databases

You can dedicate files or database segments to communicating data between transactions.

Transactions can record the completion of certain functions on the dedicated file or database segment. A progress transaction (whose purpose is to tell the user what updates have and have not been performed) can examine the dedicated file or segment.

In the event of physical damage, user VSAM files, DL/I, and Db2 databases can be forward recovered.

Data tables (user-maintained)

User-maintained data tables (UMTs), which are recoverable after a unit of work failure can be a useful means of passing data between transactions. However, they are not forward recoverable, and not recoverable after a CICS restart.

Coupling facility data tables

Coupling facility data tables updated using the locking model, and which are recoverable after a unit of work failure, can be a useful means of passing data between transactions.

Unlike UMTs, coupling facility data tables are recoverable in the event of a CICS failure, CFDT server failure , or an MVS failure. However, they are not forward recoverable.

CICS can return all these resources to their status at the beginning of an in-flight unit of work if a task ends abnormally.

Designing to avoid transaction deadlocks

You must design your program to avoid transaction deadlocks. There are a number of techniques that you can use in your program to avoid this situation.

About this task

Consider using the following techniques:

Procedure

- Arrange for all transactions to access files in a sequence agreed in advance. This could be a suitable subject for installation standards. Be extra careful if you allow updates through multiple paths.
- Enforce explicit installation enqueueing standards so that all applications do the following:
  1. Enqueue by the same character string.
  2. Use those strings in the same sequence.
- Always access records within a file in the same sequence.

  For example, where you update several file or database records, ensure that you access them in ascending sequence.

  How to do this?

  You can use any of the following three methods. Note that the second (sorting data items into an ascending sequence by programming) is most widely accepted.

  1. The terminal operator always enters data in the existing data set sequence.

     This method requires special terminal operator action, which may not be practical within the constraints of the application. (For example, orders may be taken by telephone in random product number sequence.)
  2. The application program first sorts the input transaction contents so that the sequence of data items matches the sequence on the data set.

     This method requires additional application programming, but imposes no external constraints on the terminal operator or the application.
  3. The application program issues a SYNCPOINT command after processing each data item entered in the transaction.

     This method requires less additional programming than the second method. However, issuing a synchronization point implies that previously processed data items in the transaction are not to be backed out if a system or transaction failure occurs before the entire transaction ends. This may not be valid for the application, and raises the question as to which data items in the transaction were processed and which were backed out by CICS . If the entire transaction must be backed out, synchronization points should not be issued, or only one data item should be entered per transaction.

If you allow updates on a data set through the base and one or more alternate index paths, or through multiple alternate index paths, sequencing record updates may not provide protection against transaction deadlock. You are not protected because the different base key sequences will probably not all be in ascending (or descending) order. If you do allow updates through multiple paths, and if you need to perform several record updates, always use a single path or the base. Define such a procedure in your installation standards.

- Be aware that a transaction deadlock might occur when a CICS transaction calls two or more external resources such as IMS and Db2 through different interfaces.

  For example, the application issues a DBCTL API call, and then issues a call to Db2, which drives a Db2 Stored Procedure calling DBCTL itself. From a CICS point of view, this is a single unit of recovery, but from an IMS point of view, these are two independent units of recovery. This can cause a deadlock involving an IMS syncpoint latch if an IMS system checkpoint occurs during the time CICS makes an **ISSUE PREPARE** request for the DBCTL unit of recovery and then calls Db2 (and hence RRS and IMS) for the Db2 Stored Procedure unit of recovery.

Implications of interval control START requests

Interval control START requests initiate another task—for example, to perform updates accumulated by the START-issuing task; this allows the user to continue accumulating data without waiting for the updates to be applied.

The PROTECT option on a START request ensures that, if the task issuing the START fails during the unit of work, the new task will not be initiated, even though its start time may have passed. (See Recovery of START requests for more information about the PROTECT option.)

Consider also the possibility of a started task that fails. Unless you include abend processing in the program, only the main terminal will know about the failure. The abend processing should analyze the cause of failure as far as possible, and restart the task if appropriate. Ensure that either the user or the main terminal operator can take appropriate action to repeat the updates. You could, for example, allow the user to reinitiate the task.

An alternative solution is for the started transaction to issue a START command specifying its **own** TRANSID. Immediately before issuing the RETURN command, the transaction should cancel the START command. The effect of this will be that, if a started task fails, it will automatically restart. (If the interval specified in the START command is too short, the transaction could be invoked again while the first invocation is still running. Ensure that the interval is long enough to prevent this.)

Implications of automatic task initiation (TD trigger level)

Specifying the TRANSID operand in the resource definition for an intrapartition transient data destination starts the named transaction when the trigger level is reached. Designate such a destination as **logically** recoverable. This ensures that the transient data records are committed before the task executes and uses those records.

Implications of presenting large amounts of data to the user

Ideally, a transaction that updates files or databases should defer confirmation (to the user) until such updates are committed (by user syncpoint or end of task).

In cases where the application requires the reply to consist of a large amount of data that cannot all be viewed at one time (such as data required for browsing), several techniques are available, including:

- Terminal paging through BMS
- Using transient data queues

Terminal paging through BMS

The application program (using the **SEND PAGE BMS** commands) builds pages of output data on a temporary storage queue for subsequent display using operator page commands.

Such queues should, of course, be specified as recoverable, as described in Recovery for temporary storage.

The application program should then send a committed output message to the user to say that the task is complete, and that the output data is available in the form of terminal pages.

If an uncontrolled termination occurs while the user is viewing the pages of data, those pages are not lost (assuming that temporary storage for BMS is designated as recoverable). After emergency restart, the user can resume terminal paging by using the CSPG CICS-supplied transaction and terminal paging commands. (For more information about CSPG, see CSPG - page retrieval.)

Using transient data queues

When a number of tasks direct large amounts of data to a single terminal (for example, a printer receiving multipage reports initiated by the users), it may be necessary to queue the data (on disk) until the terminal is ready to receive it.

About this task

Such queuing can be done on a transient data queue associated with a terminal. A special transaction, triggered when the terminal is available, can then format and present the data.

For recovery and restart purposes:

- The transient data queue should be specified as logically recoverable.
- If the transaction that presents the data fails, dynamic transaction backout will be called.

  If the terminal at which the transaction runs is a printer, however, dynamic transaction backout (and a restart of the transaction by whatever means) may cause a partial duplication of output—a situation that might require special user procedures. The best solution is to ensure that each unit of work corresponds to a printer page or form.

Managing transaction and system failures

To help you manage transaction failures and uncontrolled shutdowns of the system, a number of facilities are available to help you.

About this task

These facilities ensure that:

1. Files and databases remain in a coordinated and consistent state.
2. Diagnostic and warning information is produced if a program fails.
3. Communication between transactions is not affected by the failure.

The actions taken by CICS are described under Unit of work recovery and abend processing and Processing operating system abends and program checks.

Transaction failures

When a transaction fails, you can invoke CICS facilities during and after the abend process.

These facilities include:

- CICS condition handling
- HANDLE ABEND commands, and user exit code
- The SYNCPOINT ROLLBACK command
- Dynamic transaction backout (DTB)

- Transaction restart after DTB
- The program error program (DFHPEP)

You can use these facilities individually or together. During the internal design phase, specify which facilities to use and determine what additional (application or systems) programming might be involved.

The RESP option on a command returns a condition ID that can be tested. Alternatively, a HANDLE CONDITION command is used in the local context of a transaction program to name a label where control is passed if certain conditions occur.

For example, if file input and output errors occur (where the default action is merely to abend the task), you might want to inform the main terminal operator, who can decide to terminate CICS , especially if one of the files is critical to the application.

Your installation might have standards relating to the use of RESP options or HANDLE CONDITION commands. Review these for each new application.

HANDLE ABEND commands

A **HANDLE ABEND** command can pass control to a routine within a transaction, or to a separately compiled program when the task abends.

The kind of things you might do in abend-handling code include:

- Capturing diagnostic information (in addition to that provided by CICS) before the task abends, and sending messages to the main terminal and user
- Executing clean up actions, such as canceling start requests (if the PROTECT option has not been used)
- Writing journal records to reverse the effects of explicit journaling performed before the abend.

Your installation might have standards relating to the use of HANDLE ABEND commands; review these for each new application.

EXEC CICS SYNCPOINT ROLLBACK command

ROLLBACK might be useful within your transaction if, for instance, the transaction discovers logically inconsistent input after some database updates have been initiated, but before they are committed by the syncpoint.

Before deciding to use it, however, consider the following:

- Rollback backs out updates to recoverable resources performed in the current unit of work only and not in the task as a whole.
- The SYNCPOINT command, with or without the ROLLBACK option, causes a new unit of work to start.
- If you have a transaction abend, and you do not want the transaction to continue processing, issue an **EXEC CICS ABEND** and allow dynamic transaction backout to back out the updates and ensure data integrity. Use rollback only if you want the application to regain control after nullifying the effects of a unit of work.

  For programming information about the SYNCPOINT command, see SYNCPOINT.

Dynamic transaction backout

Dynamic transaction backout (DTB) occurs automatically for all transactions that have recoverable resources. It is not an option you can specify on a transaction resource definition. The actions of DTB are described under Transaction backout.

Remember that:

- For transactions that access a recoverable resource, DTB helps to preserve logical data integrity.

- Resources that are to be updated should be made recoverable.
- DTB takes place only after program level abend exits (if any) have attempted clean up or logical recovery.

Transaction restart after DTB

For each transaction where DTB is specified, consider also specifying automatic transaction restart. For example, for transactions that access DL/I databases (and are subject to program isolation deadlock), automatic transaction restart is usually specified.

Even if transaction restart is specified, a task will restart automatically only under certain default conditions (listed under Abnormal termination of a task). These conditions can be changed, if absolutely necessary, by modifying the restart program DFHREST.

Use of the program error program (DFHPEP)

Decide whether or not to include your own functions, examples of which are given in The CICS-supplied PEP . (DFHPEP is invoked during abnormal task termination as described at Abnormal termination of a task .)

System failures

Specify how an application is to be restarted after an emergency restart.

Depending on how far you want to automate the restart process, application and system programming could provide the following functions:

- User exits for transaction backout processing to handle:
  - Logically deleting records added to BDAM or VSAM-ESDS files (see Exit XFCLDEL, file control logical delete exit for details of the XFCLDEL global user exit point)
  - Backing out file control log records (see Exit XFCBOUT, file control backout exit for details of the XFCBOUT global user exit point)
  - File errors during transaction backout (see Exit XFCBFAIL, file control backout failure exit for details of the XFCBFAIL global user exit point)
  - User recovery records read from the system log during emergency restart (see Exit XRCINPT for details of the XRCINPT global user exit point).
- A progress transaction to help the user discover what updates have and have not been performed. For this purpose, application code can be written to search existing files or databases for the latest record or segment of a particular type.

Handling abends and program level abend exits

You can write program-level abend exit code so that the program proceeds differently, depending on the abend that occurs.

For example, you might want the program to proceed in one of the following ways. However, it is advisable to keep abend exit code to a minimum.

- Record application-dependent information that relates to that task, in case the task terminates abnormally.

  If you want to initiate a dump, do this in the exit code at the same program level as the abend. If you initiate the dump at a higher program level than where the abend occurred, you might lose valuable diagnostic information.
- Attempt local recovery, and then continue to run the program.
- Send a message to the terminal operator. For example, you might do this if you think that the cause of the abend is bad input data.

Information available to a program-level exit routine or program

The following table shows information that the ASSIGN commands provide to a program-level exit routine or program.

Command	Information provided
ADDRESS TWA	The address of the transaction work area (TWA)
ASSIGN ABCODE	The current CICS abend code
ASSIGN ABOFFSET	The offset of the latest ASRA, ASRB, or ASRD abend
ASSIGN ABPROGRAM	The name of the failing program for the latest abend
ASSIGN ASRAINTRPT	The instruction length code (ILC) and program interrupt code (PIC) data for the latest AICA, ASRA, ASRB, ASRD or ASRE abend
ASSIGN ASRAKEY	The execution key at the time of the last AEYD, AEYF, AICA, ASRA, or ASRB abend, if any
ASSIGN ASRAPSW	The program status word (PSW) for the latest AICA, ASRA, ASRB, ASRD or ASRE abend
ASSIGN ASRAPSW16	The 16 byte PSW for the latest AICA, ASRA, ASRB, ASRD or ASRE abend
ASSIGN ASRAREGS	The general-purpose registers for the latest AICA, ASRA, ASRB, ASRD or ASRE abend
ASSIGN ASRAREGS64	The 64-bit general-purpose registers for the latest AICA, ASRA, ASRB, ASRD or ASRE abend
ASSIGN ASRASPC	The type of space in control at the time of the last AEYD, AEYF, AICA, ASRA, or ASRB abend, if any
ASSIGN ASRASTG	The type of storage being addressed at the time of the last AEYD, AEYF, AICA, ASRA, or ASRB abend, if any
ASSIGN ORGABCODE	Original abend code in cases of repeated abends

Considerations

If an abend occurs during the invocation of a CICS service, issuing a further request for the same service might cause unpredictable results, because the reinitialization of pointers and work areas, and the freeing of storage areas in the exit routine, might not be completed. In addition, ASPx abends, which are task abends while in syncpoint processing, cannot be handled by an application program.

For transactions that are to be dynamically backed out if an abend occurs, beware of writing exit code that ends with a RETURN command. This would indicate to CICS that the transaction had ended normally and would therefore prevent dynamic transaction backout and automatic transaction restart where applicable.

Exit programs can be coded in any supported language, but exit routines must be in the same language as the program of which they are a part.

Learn more

For the transaction abend codes for abnormal terminations that CICS initiates, their meanings, and the recommended actions, see Transaction abend codes.

For programming information relating to the coding of program-level exit code (such as addressability and use of registers), see Creating a program-level abend program or routine. For background information, see Abnormal termination recovery.

Processing the IOERR condition

Any program that attempts to process an IOERR condition for a recoverable resource must not issue a RETURN or SYNCPOINT command, but must terminate by issuing an ABEND command. A RETURN or SYNCPOINT command causes the recovery manager to complete the unit of work and commit changes to recoverable resources.

START TRANSID commands

In a transaction that uses the **START TRANSID** command to start other transactions, you must maintain logical data integrity.

You can maintain data integrity by following these guidelines:

1. Always use the PROTECT option of the **START TRANSID** command. This ensures that if the START-issuing task is backed out, the new task will not start.
2. If you pass data to the started transaction (on one of the data options FROM, RTERMID, RTRANSID, or QUEUE), ensure you define the associated temporary storage queue as recoverable in its TSMODEL resource definition. You can select a name for the temporary storage queue by specifying the REQID option of the START command when using any of the data options. If you do not use REQID, the temporary storage queue name generated by CICS is in the format 'DFRxxx'.

Defining the temporary storage queue as recoverable ensures that data being passed to another task is deleted from the temporary storage queue if the START-issuing task fails and is backed out. If a system failure occurs after the START-issuing task has completed its syncpoint, the START command is preserved. CICS starts the transaction specified on a recoverable START command after an emergency restart, when the expiry time is reached and provided the terminal specified the by TERMID option is available.

Note: Consider using **EXEC CICS RETURN TRANSID** with the IMMEDIATE option if the purpose is to start the next transaction in a sequence on the same terminal. This does not unlock the terminal, and in a dynamic transaction routing (DTR) environment, the transaction is eligible for DTR.

PL/I programs and error handling

ON-units are a standard method of error-handling in PL/I programs. If the execution-time option STAE is specified, CICS program control services set up an exit routine that activates the PL/I ON-units.

This exit routine can handle:

- All PL/I errors
- CICS abends that occur in the PL/I program and in associated CICS services
- Program checks

Note that, under CICS, PL/I execution-time options can be specified only by the PLIXOPT character string.

For details of PL/I coding restrictions in a CICS environment, see the appropriate PL/I programmer's guide for your compiler.

Locking (enqueuing on) resources in application programs

This topic describes locking (enqueuing) functions provided by CICS (and access methods) to protect data integrity.

About this task

There are two forms of locking:

1. The implicit locking functions performed by CICS (or the access method) whenever your transactions issue a request to change data. See the following topics:
   - "Implicit locking for nonrecoverable files" on page 567

- “Implicit locking for recoverable files” on page 569
- “Implicit enqueuing on logically recoverable TD destinations” on page 570
- “Implicit enqueuing on recoverable temporary storage queues” on page 571
- “Implicit enqueuing on DL/I databases with DBCTL” on page 571

2. The explicit enqueuing function that you request by means of an **EXEC CICS** command. This is described under “Explicit enqueuing (by the application programmer)” on page 571.

Note: Locking (implicit or explicit) data resources protects data integrity in the event of a failure, but can affect performance if several tasks attempt to operate on the same data resource at the same time. The effect of locking on performance, however, is minimized by implementing applications with short units of work, as discussed under “Dividing transactions into units of work” on page 557.

Implicit locking for nonrecoverable files

For BDAM files that are nonrecoverable (that is, LOG=NO is specified in the FCT entry), CICS does not lock records that are being updated.

By default, you get BDAM exclusive control, which operates on a physical block, is system wide, but lasts only until the update is complete. If a transaction reads a record for update under BDAM exclusive control, and the transaction subsequently decides not to change the data, it must release the BDAM exclusive control. To do this, issue an **EXEC CICS UNLOCK** command, which causes CICS to issue a RELEX macro.

If you don’t want BDAM exclusive control, specify SERVREQ=NOEXCTL on the file entry in the FCT.

For nonrecoverable VSAM files accessed in non-RLS mode, VSAM exclusive control locks the control interval during an update. For nonrecoverable VSAM files accessed in RLS mode, SMSVSAM locks the record during the update.

Figure 135 on page 568 illustrates the extent of locking for nonrecoverable files. This figure illustrates two tasks updating the same record or control interval. Task A is given a lock on the record or control interval between the READ UPDATE and WRITE commands. During this period, task B waits.

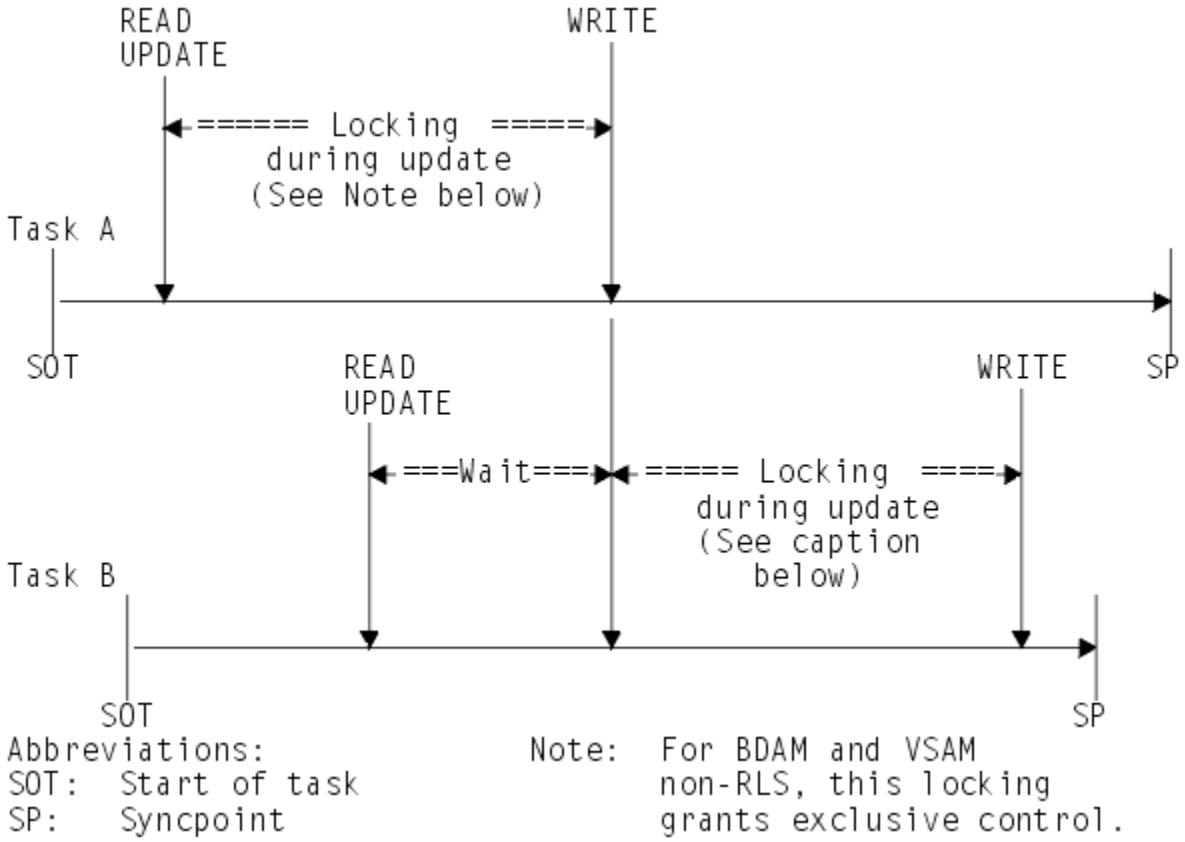


Figure 135. Locking during updates to nonrecoverable files

Figure 136 on page 569 illustrates the extent of locking (enqueuing on a resource) for nonrecoverable files. This figure illustrates two tasks updating the same record or control interval. Task A is given an exclusive lock on the record until the update is committed (at the end of the UOW). During this period, task B waits.

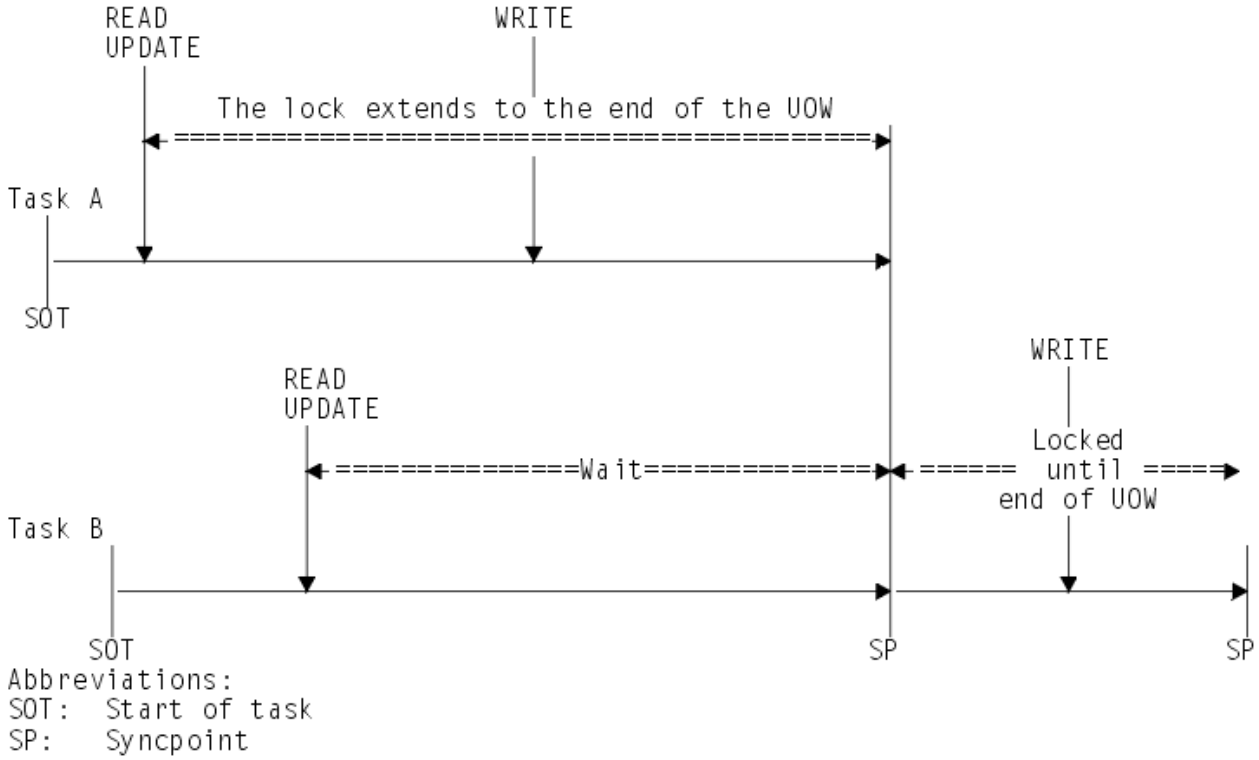


Figure 136. Locking (enqueuing on a resource) during updates to recoverable files

Implicit locking for recoverable files

For VSAM or BDAM files designated as recoverable, the duration of the locking action is extended. For VSAM files, the extended locking is on the updated record only, not the whole control interval.

The extended period of locking is needed to avoid an update committed by one task being backed out by another. (Consider what could happen if the nonextended locking action shown in Figure 135 on page 568 was used when updating a **recoverable** file. If task A abends just after task B has reached sync point and has thus committed its changes, the subsequent backout of task A returns the file to the state it was in at the beginning of task A, and task B's committed update is lost.)

To avoid this problem, whenever a transaction issues a command that changes a recoverable file (or reads from a recoverable file before update), CICS automatically locks the updated record until the change is committed (that is, until the end of the unit of work). Thus in the preceding example, Task B would not be able to access the record until Task A had committed its change at the end of the unit of work. Hence, it becomes impossible for Task B's update to be lost by a backout of Task A. For files opened in non-RLS mode, CICS provides this locking using the enqueue domain. For files opened in RLS mode, SMSVSAM provides the locking, and the locks are released at the completion of the unit of work at the request of CICS.

The file control commands that invoke automatic locking in this way are:

- READ (for UPDATE)
- WRITE
- DELETE

Note:

1. Enqueuing as previously described can lead to **transaction deadlock** (see “Possibility of transaction deadlock” on page 572).
2. The scope of locks varies according to the access method, the type of access, and who obtains the lock:
   - BDAM exclusive control applies to the physical block

- Non-RLS VSAM exclusive control applies to the control interval
- CICS locks for BDAM (with NOEXCTL specified) apply to the record only
- CICS locks for non-RLS VSAM apply to the record only
- SMSVSAM locks for RLS apply to the record only

3. **VSAM exclusive control**. The CICS enqueuing action on recoverable files opened in non-RLS mode lasts until the end of the unit of work. When a transaction issues a READ UPDATE command, VSAM's exclusive control of the control interval containing the record is held only long enough for CICS to issue an ENQ on the record. CICS then notifies VSAM to release the exclusive control of the CI, and reacquires this only when the task issues a REWRITE (or UNLOCK, DELETE, or SYNCPOINT) command. Releasing the VSAM exclusive CI lock until the task is ready to rewrite the record minimizes the potential for transaction deadlock
4. For recoverable files, do not use unique key alternate indexes to allocate unique resources (represented by the alternate key). If you do, backout can fail in the following set of circumstances:

   a. A task deletes or updates a record (through the base or another alternate index) and the alternate index key is changed.
   b. Before the end of the first task's unit of work, a second task inserts a new record with the original alternate index key, or changes an existing alternate index key to that of the original one.
   c. The first task fails and backout is attempted.

   The backout fails because a duplicate key is detected in the alternate index indicated by message DFHFC4701, with a failure code of X'F0'. There is no locking on the alternate index key to prevent the second task taking the key before the end of the first task's unit of work. If there is an application requirement for this operation, you should use the CICS enqueue mechanism to reserve the key until the end of the unit of work.
5. To ensure that the data being read is up-to-date, the application program should:
   - For files accessed in non-RLS mode, issue a READ UPDATE command (rather than a simple READ), thus locking the data until the end of the unit of work
   - For files accessed in RLS mode, use the consistent read integrity option.

Implicit enqueuing on logically recoverable TD destinations

CICS provides an enqueuing protection facility for logically recoverable (as distinct from physically recoverable) transient data destinations in a similar way to that for recoverable files.

There is one minor difference, however: CICS regards each recoverable destination as two separate recoverable resources—one for writing and one for reading.

Transient data control commands that invoke implicit enqueuing are:

- **`WRITEQ TD`**
- **`READQ TD`**
- **`DELETEQ TD`**

Thus, for example:

- If a task issues a WRITEQ TD command to a particular destination, the task is enqueued upon that write destination until the end of the task (or unit of work). While the task is thus enqueued:
  - Another task attempting to write to the same destination is suspended.
  - Another task attempting to read from the same destination is allowed to read only committed data (not data being written in a currently incomplete unit of work).
- If a task issues a READQ TD command to a particular destination, the task is enqueued upon that read destination until the end of task (or unit of work). While the task is thus enqueued:
  - Another task attempting to read from the same destination is suspended.

- Another task attempting to write to the same destination is allowed to do so and will itself enqueue on that write destination until end of task (or unit of work).
- If a task issues a DELETEQ TD request, the task is enqueued upon both the read and the write destinations. While the task is thus enqueued, no other task can read from, or write to, the queue.

Implicit enqueuing on recoverable temporary storage queues

CICS provides the enqueuing protection facility for recoverable temporary storage queues in a similar way to that for recoverable files on VSAM data sets.

There is one minor difference, however: CICS enqueuing is not invoked for **READQ TS** commands, thereby making it possible for one task to read a temporary storage queue record while another is updating the same record. To avoid this, use explicit enqueuing on temporary storage queues where concurrently executing tasks can read and change queue(s) with the same temporary storage identifier. (See "Explicit enqueuing (by the application programmer)" on page 571.)

Temporary storage control commands that invoke implicit enqueuing are:

- WRITEQ TS
- DELETEQ TS

Implicit enqueuing on DL/I databases with DBCTL

IMS ™ program isolation scheduling ensures that, when a task accesses a segment by a DL/I database call, it implicitly enqueues on all segments in the same database record as the accessed segment.

How long it is enqueued depends on the access method being used:

Direct methods (HDAM, HIDAM)
If an ISRT, DLET, or REPL call is issued against a segment, that segment, with all its child segments (and, for a DLET call, its parent segments as well), remains enqueued upon until a DL/I TERM call is issued. The task dequeues from all other segments in the database record by accessing a segment in a different database record.

Sequential methods (HSAM, HISAM, SHISAM)
If the task issues an ISRT, DLET, or REPL call against any segment, the entire database record remains enqueued upon until a DL/I TERM call is issued. If no ISRT, DLET, or REPL call is issued, the task dequeues from the database record by accessing a segment in a different database record.

The foregoing rules for program isolation scheduling can be overridden using the 'Q' command code in a segment search argument (this command extends enqueuing to the issue of a DL/I TERM call), or by using PROCOPT=EXCLUSIVE in the PCB (this macro gives exclusive control of specified segment types throughout the period that the task has scheduled the PSB).

Explicit enqueuing (by the application programmer)

CICS provides enqueuing commands that can be useful in applications when you want to protect data and prevent transaction deadlocks.

CICS provides the following explicit enqueuing commands:

- **EXEC CICS ENQ RESOURCE**
- **EXEC CICS DEQ RESOURCE**

You can use these commands to perform the following functions:

- Protect data written into the common work area (CWA), which is not automatically protected by CICS.
- Prevent transaction deadlock by enqueuing on records that might be updated by more than one task concurrently.
- Protect a temporary storage queue from being read and updated concurrently.

To be effective, however, all transactions must adhere to the same convention. A transaction that accesses the CWA without using the agreed ENQ and DEQ commands is not suspended, and protection is violated.

After a task has issued an ENQ RESOURCE(*data-area*) command, any other task that issues an ENQ RESOURCE command with the same data-area parameter is suspended until the task issues a matching DEQ RESOURCE(*data-area*) command, or until the unit of work ends.

Note: Enqueueing on more than one resource concurrently might create a deadlock between transactions.

Possibility of transaction deadlock

The enqueuing and program isolation scheduling mechanisms, which protect resources against double updating, can cause a situation known as *transaction deadlock*.

As shown in Figure 137 on page 572, transaction deadlock means that two (or more) tasks cannot proceed because each task is waiting for the release of a resource that is enqueued upon by the other. (The enqueuing, DL/I program isolation scheduling action, or VSAM RLS locking action protects resources until the next synchronization point is reached.)

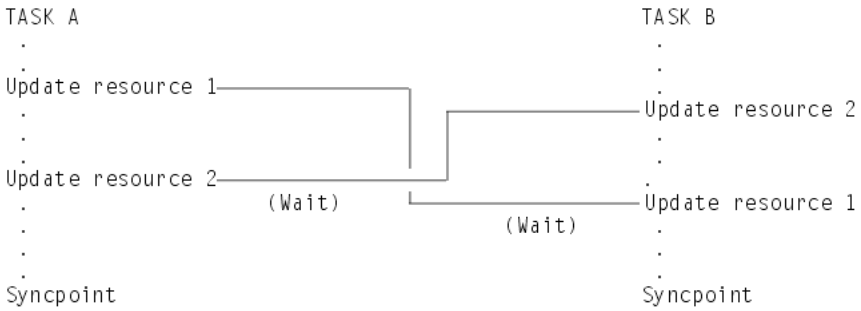


Figure 137. Transaction deadlock (generalized)

If transaction deadlock occurs, one task abends and the other proceeds.

- Transaction deadlock detection is controlled by the DTIMOUT transaction attribute. If the task remains suspended (inactive) for the specified interval (and SPURGE(YES) is also specified), CICS initiates an abnormal termination of the task. The CICS supplied mirror transactions, CSMI, CSM1, CSM2, CSM3 and CSM5 have the default settings of DTIMOUT(NO) and SPURGE(NO).
- If both deadlocked resources are VSAM RLS resources, deadlock detection is performed by VSAM. If VSAM detects an RLS deadlock condition, it returns a deadlock exception condition to CICS, causing CICS file control to abend the transaction with an AFCW abend code. CICS also writes messages and trace entries that identify the members of the deadlock chain.

  Note: VSAM cannot detect a cross-resource deadlock (for example, a deadlock arising from use of RLS and Db2 resources) where another resource manager is involved. VSAM resolves a cross-resource deadlock when the timeout period expires, as defined by either the DTIMOUT or FTIMEOUT parameters, and the waiting request is timed out. In this situation, VSAM cannot determine whether the timeout is caused by a cross-resource deadlock, or by a timeout caused by another transaction acquiring an RLS lock and not releasing it.
- If the resources are both DL/I databases, DL/I itself detects the potential deadlock as a result of the tasks issuing their scheduling calls. In this case, DL/I causes CICS to abend the task that has the least update activity (with abend code ADCD).
- If both deadlocked resources are CICS resources (but not both VSAM resources), or one is CICS and the other DL/I, CICS abends the task whose DTIMOUT period elapses first. It is possible for both tasks to time out simultaneously. If neither task has a DTIMOUT period specified, they both remain suspended indefinitely, unless one of them is canceled by a main terminal command.

The abended task may then be backed out by dynamic transaction backout, as described in Transaction backout. (Under certain conditions, the transaction can be restarted automatically, as described under Abnormal termination of a task. Alternatively, the terminal operator may restart the abended transaction.)

For more information, see "Designing to avoid transaction deadlocks" on page 560.

User exits for transaction backout

You can include your own logic in global user exit programs that run during dynamic transaction backout and during backout at an emergency restart.

There are exits in the file control recovery control program, and in the user log record recovery program (which is driven at emergency restart only). Transient data and temporary storage backouts do not have any exits.

Where you can add your own code

At emergency restart, you can add your own code in post-initialization programs that you nominate in the program list table.

About this task

You might want to include the following function in global user exit programs that run during an emergency restart:

- Process file control log records containing the details of file updates that are backed out
- Deal with the backing-out of additions to data set types that do not support deletion (VSAM ESDS and BDAM), by flagging the records as logically deleted
- Handle file error conditions that arise during transaction backout
- Process user recovery records (in the XRCINPT exit of the user log record recovery program during emergency restart)
- Deal with the case of a non-RLS batch program having overridden RLS retained locks (this should not occur often, if at all)

You can use the following exits:

1. XRCINIT—started at the beginning and end of the user log record recovery program
2. XRCINPT—started whenever a user log record is read from the system log
3. XFCBFAIL—file backout failure exit
4. XFCLDEL—file logical delete exit
5. XFCBOVER—file backout non-RLS override exit
6. XFCBOUT—file backout exit

You can use any of these exits to add your own processing if you do not want the default action, but do not set the UERCPURG return code for these exits because the exit tasks cannot be purged. Use the following procedure to use these exits at emergency restart.

Procedure

- Enable the exits in PLT programs in the first part of PLT processing.
- Specify the exits on the system initialization parameter, **TBEXITS**.

  This takes the form TBEXITS=(*name1,name2,name3,name4,name5,name6*), where *name1, name2, name3, name4, name5* , and *name6* are the names of your global user exit programs for the XRCINIT, XRCINPT, XFCBFAIL, XFCLDEL, XFCBOVER, XFCBOUT exit points.

What to do next

For programming information about the generalized interface for exits, how to write exit programs, and the input parameters and return codes for each exit, see Global user exit programs.

XRCINIT exit

XRCINIT is invoked at warm and emergency restart:

1. When the first user log record is read from the system log, and before it is passed to the XRCINPT exit
2. After the last user log record has been read from the system log and passed to XRCINPT

The XRCINIT exit code must always end with a return code of UERCNORM. No choice of processing options is available to this exit.

CICS passes information in the global user exit parameter list about user-written log records presented at the XRCINPT exit. The parameters includes a flag byte that indicates the disposition of the user log records. This can indicate the state of the unit of work in which the user log records were found, or that the record is an activity keypoint record. The possible values indicate:

- The record is an activity keypoint record
- The UOW was committed
- The UOW was backed out
- The UOW was in-flight
- The UOW was indoubt

XRCINPT exit

XRCINPT is invoked at warm and emergency restart, once for each user log record read from the system log.

The default action at this exit is to do nothing.

If you want to ignore the log record, return with return code UERCBYP. This frees the record area immediately and reads a new record from the system log. Take care that this action does not put data integrity at risk.

XFCBFAIL global user exit

XFCBFAIL is invoked whenever an error occurs during backout of a unit of work.

An XFCBFAIL global user exit program can decide whether to bypass, or invoke, CICS backout failure control, The processing performed by CICS backout failure control is described under Backout-failed recovery.

XFCLDEL global user exit

XFCLDEL is invoked when backing out a unit of work that performed a write operation to a VSAM ESDS, or a BDAM data set.

XFCBOVER global user exit

XFCBOVER is invoked whenever CICS is about to decide not to backout an uncommitted update, because the record could have been updated by a non-RLS batch program.

This situation can occur after a batch program has opened a data set, even though it has retained locks, by overriding the RLS data set protection.

XFCBOUT global user exit

XFCBOUT is invoked when CICS is about to backout a file update.

Coding transaction backout exits

You have access to all CICS services, except terminal control services, during exit execution.

About this task

However, consider the following restrictions:

- The exit programs must be written in assembler code.
- They must be quasi-reentrant. They may use the exit programming interface (XPI) and issue **EXEC CICS** commands.
- If an exit program acquires an area as a result of a file control request, it is the responsibility of the exit to release that area.
- An exit must not attempt to make any file control requests to a file referring to a VSAM data set opened in non-RLS mode with a string number of 1, unless no action is specified for that file during the initialization exit.
- Task-chained storage acquired in an exit should be released by the exit as soon as its contents are no longer needed.
- If an exit is not used, the default actions are taken.
- We strongly recommend that emergency restart global user exits do not change any *recoverable* resource. If you do try to use temporary storage, transient data, or file control, these resource managers may also be in a state of recovery. Access to these services will, therefore, at best cause serialization of the recovery tasks and, at worst, cause a deadlock.

Chapter 12. Translation and compilation

Some older compilers (and assemblers) cannot process CICS commands directly. An additional step is needed to convert your program into executable code. This step is called *translation* , and consists of converting CICS commands into the language in which the rest of the program is coded, so that the compiler (or assembler) can understand them.

Most compilers use the integrated CICS translator approach, where the compiler interfaces with CICS at compile time to interpret CICS commands and convert them automatically to calls to CICS service routines. If you use the integrated CICS translator approach, many of the translation tasks are done at compile time and you do not need to complete the additional translation step. For more information about the task in the translation step, see “The translation process” on page 579.

This section describes:

- “The integrated CICS translator ” on page 577
- “The translation process” on page 579
- “The CICS-supplied translators” on page 581
- “Using a CICS translator” on page 590
- “Defining translator options” on page 592
- “Using COPY statements” on page 594
- “The CICS-supplied interface modules” on page 594
- “Using the EXEC interface modules for AMODE(24) and AMODE(31) applications” on page 595

The integrated CICS translator

Using the integrated translator, you can translate and compile your high-level source code in a single step. Compilers that support the integrated translator scan the application source and call the integrated translator at relevant points. The integrated translator converts **EXEC CICS** commands into comments and generates CALL statements appropriate to the language.

There are versions of the integrated translator for the following languages when compiling CICS online programs:

C
C++
COBOL
PL/I

In addition, you can use the integrated translator when compiling COBOL, C, C++, and PL/I batch programs that use the External CICS Interface (EXCI) command level API.

With the integrated translator, application development is faster because there is no separate translation step. It is also made easier because there is only one listing; the original source statements and the CICS error messages are included in the compiler listing. The CICS-supplied separate translators change the line numbers in source programs, which means that with translator-generated calls you need an intermediate listing that must be used when debugging an application program.

With the integrated translator, the process of translating and compiling is also less error-prone because it is no longer necessary to translate included members separately.

For COBOL programs, it is recommended to use the integrated translator because the separate CICS translator has not been updated for newer COBOL language such as floating comment delimiters, JSON GENERATE and JSON PARSE, and compiler directives. When you migrate COBOL applications to use the integrated CICS translator, follow the migration instructions in the *Enterprise COBOL for z/OS Migration Guide*.

© Copyright IBM Corp. 1974, 2023

The Language Environment-conforming language compilers that support the integrated translator scan the application source and call the integrated CICS translator at relevant points.

The releases of the language compilers that support the integrated translator are listed in Changes to CICS support for application programming languages. If you use any other compiler, including Assembler, you must translate your program before compiling it.

Using the integrated CICS translator

The language compilers provide various procedures that you can use with the integrated CICS translator. They are documented in the Programming Guides for Enterprise COBOL for z/OS , z/OS XL C/C++ , and for Enterprise PL/I for z/OS.

About this task

The procedure that you use needs to have CICSTS56.CICS .SDFHLOAD added to the STEPLIB concatenation for the compilation step and the link-edit step should include the interface module DFHELII at the start of the step.

To use the integrated CICS translator for PL/I, you must specify the compiler option `SYSTEM(CICS)`.

To use the integrated CICS translator for COBOL, the compiler options CICS, NODYNAM, and RENT must be in effect. NODYNAM is not a restriction specific to the integrated translator. DYNAM is not supported for code that is separately translated and compiled. Do not use SIZE(MAX), because storage must remain in the user region for integrated CICS translator services. Instead, use a value such as SIZE(4000K), which should work for most programs.

Note: The compiler LIB option is no longer required for COBOL 5 and later, and has been removed. For earlier versions of COBOL, this option must be manually reinstated.

To use the integrated CICS translator for C and C++, use the `CICS` option.

If you are running DB2® Version 7 or later and preparing a COBOL program using a compiler with integrated translator, the compiler also provides an SQL statement coprocessor (which produces a DBRM), so you do not need to use a separate Db2 precompiler. See CICS Db2 program preparation and Programming for Db2 for z/OS in Db2 for z/OS product documentation for more information on using the SQL statement coprocessor.

Specifying CICS translator options

You can specify CICS translator options when you use the PL/I, COBOL, XL C, or C++ compiler.

About this task

For a description of all the translator options, see “Defining translator options” on page 592.

Many translator options do not apply when using the integrated CICS translator, for example those associated with translator listings. If these options are specified, they are ignored. The EXCI option is not supported for PL/I, but is supported for batch COBOL, C and C++ programs that use the EXCI command level API.

The following translator options can be used effectively with the integrated CICS translator:

- APOST or QUOTE
- CPSM or NOCPSM
- CICS
- DBCS
- DEBUG or NODEBUG
- DLI
- EDF or NOEDF
- FEPI or NOFEPI

- GRAPHIC
- LENGTH or NOLENGTH
- LINKAGE or NOLINKAGE
- NATLANG
- SP
- SYSEIB

Procedure

- To specify CICS translator options when using the PL/I compiler, specify the compiler option, PP(CICS), with the translator options enclosed in apostrophes and inside parentheses.

  For example:

  ```
  PP(CICS('opt1 opt2 optn ...'))
  ```

  For more information about specifying PL/I compiler options, see the Enterprise PL/I for z/OS Programming Guide.
- To specify CICS translator options when using the COBOL compiler, specify the compiler option, CICS, with the translator options enclosed in apostrophes and inside parentheses.

  For example:

  ```
  CICS('opt1 opt2 optn ...')
  ```

  Note: The XOPTS translator option must be changed to the CICS compiler option. XOPTS is not accepted when using the integrated CICS translator.

  For more information about specifying COBOL compiler options, see the Enterprise COBOL for z/OS Programming Guide.
- To specify CICS translator options when using the XL C and C++ compiler specify the compiler option, CICS, with the translator options inside parentheses and separated by commas.

  For example:

  ```
  CICS(opt1,opt2,optn ...)
  ```

  Alternatively, you can specify translator options on a `#pragma` statement in the program source on the XOPTS or CICS keyword.

  For more information about specifying C and C++ compiler options, see z/OS XL C/C++ User's Guide.

The translation process

For compilers without integrated translators, CICS provides a translator program for each of the languages in which you may write, to handle EXEC CICS, EXEC CPSM and EXEC DLI statements. For compilers with integrated translators, the compilers call the CICS translator directly to handle the translation of these commands.

A language translator reads your source program and creates a new one; most normal language statements remain unchanged, but CICS commands are translated into CALL statements of the form required by the language in which you are coding. The calls invoke CICS-provided "EXEC" interface modules, which later get link-edited into your load module, and these in turn invoke the requested services at execution time.

There are three steps: translation, compilation (assembly), and link-edit. Figure 138 on page 580 shows these 3 steps.

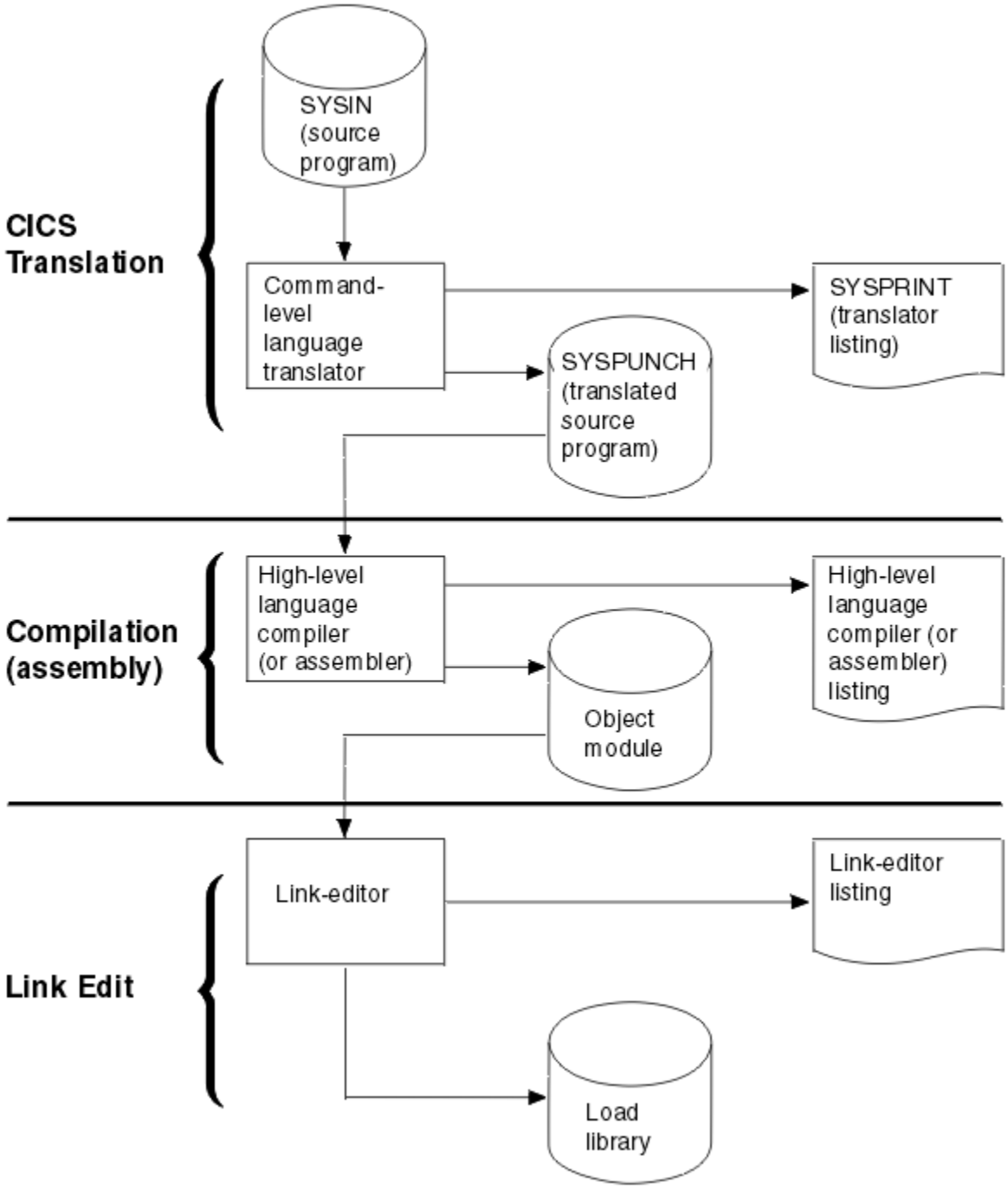


Figure 138. Preparing an application program

If your system administrator has defined rules that identify restricted CICS API and SPI commands in the restricted commands parmlib member DFHAPIR, during translation the CICS translator also checks your source program against the restricted commands and keywords. In case of violation, the CICS translator will issue warning or error messages, and the translation might fail. For more information, see Controlling the use of specific CICS API and SPI commands.

The translators for all languages use one input and two output files:

SYSIN
(Translator input) is the file that contains your source program.

If the SYSIN file is defined as a fixed blocked data set, the maximum record length that the data set can possess is 80 bytes. Passing a fixed blocked data set with a record length of greater than 80 bytes

to the translator results in termination of translator execution. If the SYSIN file is defined as a variable blocked data set, the maximum record length that the data set can possess is 100 bytes. Passing a variable blocked data set with a record length greater than 100 bytes to the translator causes the translator to stop with an error.

SYSPUNCH
(Translated source) is the translated version of your source code, which becomes the input to the compilation (assemble) step. In this file, your source has been changed as follows:

- The EXEC interface block (EIB) structure has been inserted.
- EXEC CICS , EXEC CPSM and EXEC DLI commands have been turned into function call statements.
- CICS DFHRESP, EYUVALUE, and DFHVALUE built-in functions have been processed.
- A data interchange block (DIB) structure and initialization call have been inserted if the program contains EXEC DLI statements.

The CICS commands that get translated still appear in the source, but as comments only. Generally the non-CICS statements are unchanged. The output from the translator always goes to an 80 byte fixed-record length data set.

SYSPRINT
(Translator listing) shows the number of messages produced by the translator, and the highest severity code associated with any message. The options used in translating your program also appear, unless these have been suppressed with the NOOPTIONS option.

For COBOL, C, C++, and PL/I programs, SYSPRINT also contains the messages themselves. In addition, if you specify the SOURCE option of the translator, you also get an annotated listing of the source in SYSPRINT. This listing contains almost the same information as the subsequent compilation listing, and therefore many installations elect to omit it (the NOSOURCE option). One item you may need from this listing which is not present in the compilation listing, however, is the line numbers, if the translator is assigning them. Line numbers are one way to indicate points in the code when you debug with the execution diagnostic facility (EDF). If you specify the VBREF option, you also get a list of the commands in your program, cross-referenced by line number, and you can use this as an alternative to the source listing for EDF purposes.

For assembler language programs, SYSPRINT contains only the translator options, the message count and maximum severity code. The messages themselves are inserted into the SYSPUNCH file as comments after the related statement. This causes the assembler to copy them through to the assembler listing, where you can check them. You may also see MNOTEs that are generated by the assembler as the result of problems encountered by the translator.

Note: If you use EXEC SQL, you need additional steps to translate the SQL statements and bind.

CICS provides a procedure to execute these steps in sequence for each of the languages it supports. “Using the CICS-supplied procedures to install application programs” on page 690 describes how to use these procedures, and exactly what they do.

You can control the translation process by specifying a number of **options**. For example, if your program uses EXEC DLI calls, you need to tell the translator.

The translator may produce error messages, and it is as important to check these messages as it is to check the messages produced by the compiler and link-editor. See “The CICS-supplied translators” on page 581 for the location of these messages.

EXEC commands are translated into CALL statements that invoke CICS interface modules. These modules get incorporated into your object module in the link-edit step, and you see them in your link-edit output listing. You can read more about these modules in “The CICS-supplied interface modules” on page 594.

The CICS-supplied translators

The CICS-supplied separate translators are installed in the CICSTS56.CICS.SDFHLOAD library.

The following table lists the separate translators in this library.

Table 56. CICS-supplied separate translator by language	
Language	**Translator**
Assembler	DFHEAP1$
C	DFHEDP1$
COBOL	DFHECP1$ **Note:** The separate CICS translator has not been updated for newer COBOL language such as floating comment delimiters, JSON GENERATE and JSON PARSE, and compiler directives. To use the latest features of the COBOL compiler, use the integrated CICS translator. See Migrating from the separate CICS translator to the integrated translator in Enterprise COBOL for z/OS Migration Guide.
PL/I	DFHEPP1$

Dynamic invocation of the separate translator

You can invoke the command-level language translator dynamically from a batch assembler-language program using an ATTACH, CALL, LINK, or XCTL macro; or from a C, PL/I, or COBOL program using CALL.

About this task

If you use ATTACH, LINK, or XCTL, use the appropriate translator load module, DFHExP1$, where x=A for assembler language, x=C for COBOL, x=D for C, or x=P for PL/I.

If you use CALL, specify PREPROC as the entry point name to call the translator.

In all cases, pass the following address parameters to the translator:

- The address of the translator option list
- The address of a list of DD names used by the translator (this is optional)

These addresses must be in adjacent fullwords, aligned on a fullword boundary. Register 1 must point to the first address in the list, and the high-order bit of the last address must be set to one, to indicate the end of the list. This is true for both one or two addresses.

Translator options

The command-level language translator has a list of options that you can select from to translate your COBOL, C, C++, PL/I, and Assembler programs. Each translator option is explained and lists the languages that you can use it with.

APOST
(COBOL only)

APOST indicates that literals are delineated by the apostrophe or a quotation mark. QUOTE is the alternative option, which indicates double quotation marks.

This translator option determines the appropriate value for all delimiters in COBOL source code, including those in copybooks, CICS statements and COBOL literals in the source. Although COBOL allows delimiters in COBOL source code to be apostrophes, quotation marks, or a mix of both in the same program, if the CICS stand-alone translator is used, then all beginning and ending delimiters in the COBOL program must match the CICS option specified. Literals continued across multiple lines must also start and end with the specified delimiter.

Note that the CICS-supplied COBOL copybooks have a single quotation mark. If you are using any CICS-supplied copybooks in your application to interface to a CICS component, ensure that you use the APOST option, not the QUOTE option, and code the delimiters in your program accordingly.

C5

When specified, C5 causes the translator to generate COMP-5 rather than COMP by changing EIBCALEN, EIBCPOSN, and DFHEIGDI from PIC S9(4) COMP to PIC S9(4) COMP-5. This results in values greater than 9999 not being truncated.

CBLCARD

(COBOL only) Abbreviation: CBL

CBLCARD specifies that the translator is to generate a CBL statement. CBLCARD is the default; the alternative is NOCBLCARD.

The COBOL compiler LIB option is not required for COBOL Version 5 and later. The compiler options that result from the CBLCARD translator option do not include the LIB option. When translating source to be processed by COBOL compilers earlier than Version 5, you must specify the LIB option without relying on CBLCARD.

CICS

CICS specifies that the translator is to process **EXEC CICS** commands. It is the default specification in the translator. CICS is also an old name for the XOPTS keyword for specifying translator options, which means that you can specify the CICS option explicitly either by including it in your XOPTS list, or by using it in place of XOPTS to name the list. The only way to indicate that there are no CICS commands is to use the XOPTS keyword without the option CICS . You must do this in a batch DL/I program using EXEC DLI commands. For example, to translate a batch DL/I program written in assembler language, specify:

```
*ASM XOPTS(DLI)
```

To translate a batch program written in COBOL, containing EXEC API commands, specify:

```
CBL XOPTS(EXCI)
```

COBOL2

(COBOL only) Abbreviation: CO2

COBOL2 specifies that the translator is to generate temporary variables for use in the translated EXEC statements. In all other respects, the program is translated in the same manner as with the COBOL3 option. COBOL2 and COBOL3 are mutually exclusive. COBOL2 is the default for COBOL.

Note: If you specify COBOL2 and COBOL3 by different methods, the COBOL3 option is always used, regardless of where the two options have been specified. If you specify both COBOL2 and COBOL3 , the translator issues a warning message.

COBOL3

(COBOL only) Abbreviation: CO3

COBOL3 specifies that the translator is to translate programs that are Language Environment-conforming. COBOL3 and COBOL2 options are mutually exclusive. For information about which Language Environment-conforming compilers are available, see Chapter 9, “ Programming languages and Language Environment ,” on page 533.

CPP

(C++ only)

CPP specifies that the translator is to translate C++ programs for compilation by a supported C++ compiler.

CPSM

CPSM specifies that the translator is to process EXEC CPSM commands. The alternative is NOCPSM , which is the default.

DBCS

(COBOL only)

DBCS specifies that the source program might contain double-byte characters. The DBCS option causes the translator to treat hexadecimal codes X'0E' and X'0F' as shift-out (SO) and shift-in (SI) codes, respectively, wherever they appear in the program.

For more detailed information about how to program in COBOL using DBCS, see the section on DBCS character strings in Enterprise COBOL for z/OS Language Reference.

DEBUG
(COBOL, C, C++, and PL/I only)

DEBUG instructs the translator to produce code that passes the line number through to CICS for use by the execution diagnostic facility (EDF). DEBUG is the default; NODEBUG is the alternative.

DLI

DLI specifies that the translator is to process EXEC DLI commands. You must specify it with the XOPTS option, that is, XOPTS(DLI).

EDF

EDF specifies that the execution diagnostic facility is to apply to the program. EDF is the default; the alternative is NOEDF.

EPILOG
(Assembler language only)

EPILOG specifies that the translator is to insert the macro DFHEIRET at the end of the program being translated. DFHEIRET returns control from the issuing program to the program which invoked it. If you want to use any of the options of the **RETURN** command, use RETURN and specify NOEPILOG.

EPILOG is the default; the alternative NOEPILOG prevents the translator from inserting the macro DFHEIRET. (See for programming information about the DFHEIRET macro.)

EXCI

EXCI specifies that the translator is to process EXEC API commands for the External CICS Interface (EXCI). EXCI API commands must be used only in batch programs, and so the EXCI translator option is mutually exclusive to the CICS translator option, or any translator option that implies the CICS option. An error message is produced if both CICS and EXCI are specified, or EXCI and a translator option that implies CICS are specified.

The EXCI option is also mutually exclusive to the DLI option. EXEC API commands for the External CICS Interface cannot be coded in batch programs using EXEC DLI commands. An error message is produced if both EXCI and DLI translator commands are specified.

The EXCI translator option is specified by XOPTS , that is, XOPTS(EXCI).

FEPI

FEPI allows access to the FEPI API commands of the CICS Front End Programming Interface (FEPI). The alternative is NOFEPI. For more information about FEPI, see Developing with the FEPI API.

FLAG (I, W, E, or S)
(COBOL, C, C++, and PL/I only) Abbreviation: F

FLAG specifies the minimum severity of error in the translation that requires a message to be listed.

I
All messages.

W
(Default) All except information messages.

E
All except warning and information messages.

S
Only severe and unrecoverable error messages.

GDS
(C, C++, and assembler language only)

GDS specifies that the translator is to process CICS GDS (generalized data stream) commands. For programming information about these commands, see CICS API commands.

GRAPHIC
(PL/I only)

GRAPHIC specifies that the source program might contain double-byte characters. The GRAPHIC option causes the translator to treat hexadecimal codes X'0E' and X'0F' as shift-out (SO) and shift-in (SI) codes, respectively, wherever they appear in the program.

It also prevents the translator from generating parameter lists that contain the shift-out and shift-in values in hexadecimal form. Wherever these values would ordinarily appear, the translator expresses them in binary form, so that there are no unintended DBCS delimiters in the data stream that the compiler receives.

If the compiler you are using supports DBCS, you need to prevent unintended shift-out and shift-in codes, even if you are not using double-byte characters. Prevent unintended shift-out and shift-in codes by specifying the GRAPHIC option for the translator, so that it does not create them, or by specifying NOGRAPHIC on the compile step, so that the compiler does not interpret them as DBCS delimiters.

For more detailed information about how to program in PL/I using DBCS, see the relevant language reference manual.

LEASM
(Assembler language only)

LEASM instructs the translator to generate code for a Language Environment-conforming assembler MAIN program.

If the LEASM option is specified, the DFHEISTG, DFHEIENT, DFHEIRET, and DFHEIEND macros expand differently to create a Language Environment-conforming assembler MAIN program, instead of the form of macro expansion used for assembler subroutines in a CICS environment. The LEASM option allows customer programs that have used NOPROLOG and NOEPILOG and coded their own DFHEIENT and other macros to take advantage of Language Environment support without changing their program source. For example, all programs that require more than one code base register are in this category because the translator does not support multiple code base registers.

For an example of an assembler program translated using the LEASM option, see "Example assembler language program with LEASM" on page 597.

LENGTH
(COBOL, assembler language and PL/I only)

LENGTH instructs the translator to generate a default length if the LENGTH option is omitted from a CICS command in the application program. The alternative is NOLENGTH.

LINECOUNT(n)
Abbreviation: LC

LINECOUNT specifies the number of lines to be included in each page of translator listing, including heading and blank lines. The value of *n* must be an integer in the range 1 through 255; if *n* is less than 5, only the heading and one line of listing are included on each page. The default is 60.

LINKAGE
(COBOL only) Abbreviation: LIN

LINKAGE requests the translator to modify the LINKAGE SECTION and PROCEDURE DIVISION statements in top-level programs according to the existing rules.

This means that the translator inserts a USING DFHEIBLK DFHCOMMAREA statement in the PROCEDURE DIVISION, if one does not exist, and ensures that the LINKAGE SECTION (creating one if necessary) contains definitions for DFHEIBLK and DFHCOMMAREA.

LINKAGE is the default; the alternative is NOLINKAGE.

MAIN
(C only)

MAIN specifies the program contains a main procedure and so the translator will include structures and declarations for the CICS API. This is the default.

MARGINS(m,n[,c])
(C, C++, and PL/I only) Abbreviation: MAR

MARGINS specifies the columns of each line or record of input that contain language or CICS statements. The translator does not process data that is outside these limits, though it does include it in the source listings.

The MARGINS option can also specify the position of an American National Standard printer control character to format the listing produced when the SOURCE option is specified; otherwise, the input records are listed without any intervening blank lines. The margin parameters are as follows:

m
Column number of left margin.

n
Column number of right margin. It must be greater than m.

Note: When used as a C or C++ compiler option, the asterisk (*) is allowable for the second argument on the MARGIN option. For the translator, however, a numeric value between 1 and 100 inclusive must be specified. When the input data set has fixed-length records, the maximum value allowable for the right margin is 80. When the input data set has variable-length records, the maximum value allowable is 100.

c
Column number of the American National Standard printer control character. It must be outside the values specified for m and n. A zero value for c means no printer control character. If c is nonzero, only the following printer control characters can appear in the source:

(blank)
Skip one line before printing.

0
Skip two lines before printing.

-
Skip three lines before printing.

+
No skip before printing.

1
New page.

The default for C and C++ is MARGINS(1,72,0) for fixed-length records, and for variable-length records it is the same as the record length (1,record length,0). The default for PL/I is MARGINS(2,72,0) for fixed-length records, and MARGINS(10,100,0) for variable-length records.

NATLANG(EN or KA)
NATLANG specifies which language is to be used for the translator message output:

CS
Simplified Chinese.

EN
(Default) English.

KA
Kanji.

(Take care not to confuse this option with the NATLANG API option.)

NOCBLCARD
(COBOL only)

NOCBLCARD specifies that the translator is not to generate a CBL statement. The compiler options that CICS requires are specified by the DFHYITVL procedure. Ensure that RENT and NODYNAM are specified.

NOCPSM
NOCPSM specifies that the translator is not to process EXEC CPSM commands. NOCPSM is the default, the alternative is CPSM.

NODEBUG
(COBOL, C, C++, and PL/I only)

NODEBUG instructs the translator not to produce code that passes the line number through to CICS for use by the execution diagnostic facility (EDF).

NOEDF
NOEDF specifies that the execution diagnostic facility is not to apply to the program. There is no performance advantage in specifying NOEDF , but the option can be useful to prevent commands in well-debugged subprograms appearing on EDF displays.

NOEPILOG
(Assembler language only)

NOEPILOG instructs the translator not to insert the macro DFHEIRET at the end of the program being translated. DFHEIRET returns control from the issuing program to the program which invoked it. If you want to use any of the options of the **EXEC CICS RETURN** command, use **EXEC CICS RETURN** and specify NOEPILOG. NOEPILOG prevents the translator inserting the macro DFHEIRET. The alternative is EPILOG , which is the default.

NOFEPI
NOFEPI disallows access to the FEPI API commands of the CICS Front End Programming Interface (FEPI). NOFEPI is the default; the alternative is FEPI.

NOLENGTH
(COBOL, assembler language and PL/I only)

NOLENGTH instructs the translator not to generate a default length if the LENGTH option is omitted from a CICS command in the application program. The default is LENGTH.

NOLINKAGE
(COBOL only)

NOLINKAGE requests the translator not to modify the LINKAGE SECTION and PROCEDURE DIVISION statements to supply missing DFHEIBLK and DFHCOMMAREA statements, or insert a definition of the EIB structure in the LINKAGE section.

The NOLINKAGE option means that you can provide COBOL copybooks to define a COMMAREA and use the **EXEC CICS ADDRESS** command.

LINKAGE is the default.

NOMAIN
(C only)

NOMAIN specifies the program does not contain a main procedure and so the translator will not include structures for the CICS API. The DFHEIPTR pointer is declared as external. This allows multiple C programs to be link-edited together.

NONUM
(COBOL only)

NONUM instructs the translator not to use the line numbers appearing in columns one through six of each line of the program as the line number in its diagnostic messages and cross-reference listing, but to generate its own line numbers. NONUM is the default, the alternative is NUM.

NOOPSEQUENCE
(C, C++, and PL/I only) Abbreviation: NOS

NOOPSEQUENCE specifies the position of the sequence field in the translator output records. The default for C and C++ is OPSEQUENCE(73,80) for fixed-length records and NOOPSEQUENCE for variable-length records. For PL/I, the default is OPSEQUENCE(73,80) for both types of records.

NOOPTIONS
Abbreviation: NOP

NOOPTIONS instructs the translator not to include a list of the options used during this translation in its output listing.

NOPROLOG
(Assembler language only)

NOPROLOG instructs the translator not to insert the macros DFHEISTG, DFHEIEND, and DFHEIENT into the program being assembled. These macros define local program storage and execute at program entry.

NOSEQ
(COBOL only)

NOSEQ instructs the translator not to check the sequence field of the source statements, in columns 1-6. The alternative, SEQ, is the default. If SEQ is specified and a statement is not in sequence, it is flagged.

NOSEQUENCE
(C, C++, and PL/I only) Abbreviation: NSEQ

NOSEQUENCE specifies that statements in the translator input are not sequence numbered and that the translator must assign its own line numbers.

The default for fixed-length records is SEQUENCE(73,80). For variable-length records in C and C++, the default is NOSEQUENCE and for variable-length records in PL/I, the default is SEQUENCE(1,8).

NOSOURCE
NOSOURCE instructs the translator not to include a listing of the translated source program in the translator listing.

NOSPIE
NOSPIE prevents the translator from trapping unrecoverable errors; instead, a dump is produced. Use NOSPIE only when requested to do so by the IBM support center.

NOVBREF
(COBOL, C, C++ and PL/I only)

NOVBREF instructs the translator not to include a cross-reference of commands with line numbers in the translator listing. (NOVBREF used to be called NOXREF; for compatibility, NOXREF is still accepted.) NOVBREF is the default, the alternative is VBREF.

NUM
(COBOL only)

NUM instructs the translator to use the line numbers appearing in columns one through six of each line of the program as the line number in its diagnostic messages and cross-reference listing. The alternative is NONUM , which is the default.

OPMARGINS(m,n[,c])
(C, C++ and PL/I only) Abbreviation: OM

OPMARGINS specifies the translator output margins, that is, the margins of the input to the following compiler. Normally these margins are the same as the input margins for the translator. For a definition of input margins and the meaning of *m*, *n*, and *c*, see MARGINS. The default for C and C++ is OPMARGINS(1,72,0) and for PL/I, the default is OPMARGINS(2,72,0).

The maximum *n* value allowable for the OPMARGINS option is 80. The output from the translator is always of a fixed-length record format.

If the OPMARGINS option is used to set the output from the translator to a certain format, it might be necessary to change the input margins for the compiler being used. If the OPMARGINS value is allowed to default, it is not necessary to change the input margins for the compiler being used.

OPSEQUENCE(m,n)
(C, C++, and PL/I only) Abbreviation: OS

OPSEQUENCE specifies the position of the sequence field in the translator output records. For the meaning of *m* and *n* , see SEQUENCE . The default for C and C++ is OPSEQUENCE(73,80) for fixed-length records and NOOPSEQUENCE for variable-length records. For PL/I, the default is OPSEQUENCE(73,80) for both types of records.

OPTIONS
Abbreviation: OP

OPTIONS instructs the translator to include a list of the options used during this translation in its output listing.

PROLOG
(Assembler language only)

PROLOG instructs the translator to insert the macros DFHEISTG, DFHEIEND, and DFHEIENT into the program being assembled. These macros define local program storage and execute at program entry.

QUOTE
(COBOL only) Abbreviation: Q

QUOTE indicates that literals are delineated by the double quotation mark (”). The same value must be specified for the translator step and the following compiler step. Although COBOL allows delimiters in COBOL source code to be apostrophes, quotation marks, or a mix of both in the same program, if the CICS stand-alone translator is used, then all beginning and ending delimiters in the COBOL program must match the CICS option specified. Literals continued across multiple lines must also start and end with the specified delimiter.

Note that the CICS-supplied COBOL copybooks have a single quotation mark. If you are using any CICS-supplied copybooks in your application to interface to a CICS component, ensure that you use the APOST option, not the QUOTE option, and code the delimiters in your program accordingly.

SEQ
(COBOL only)

SEQ instructs the translator to check the sequence field of the source statements, in columns 1-6. SEQ is the default; the alternative is NOSEQ . If a statement is not in sequence, it is flagged.

SEQUENCE(m,n)
(C, C++, and PL/I only) Abbreviation: SEQ

SEQUENCE specifies that statements in the translator input are sequence numbered and the columns in each line or record that contain the sequence field. The translator uses this number as the line number in error messages and cross-reference listings. No attempt is made to sort the input lines or records into sequence. If no sequence field is specified, the translator assigns its own line numbers. The SEQUENCE parameters are:

m
Leftmost sequence number column.

n
Rightmost sequence number column.

The sequence number field must not exceed eight characters and must not overlap the source program (as specified in the MARGINS option).

The default for fixed-length records is SEQUENCE(73,80). For variable-length records in C and C++, the default is NOSEQUENCE and for variable-length records in PL/I, the default is SEQUENCE(1,8).

SOURCE
Abbreviation: S

SOURCE instructs the translator to include a listing of the translated source program in the translator listing. SOURCE is the default; the alternative is NOSOURCE.

SP

SP must be specified for application programs that contain special (SP) CICS commands or they are rejected at translate time. These commands are ACQUIRE, COLLECT, CREATE, DISABLE, DISCARD, ENABLE, EXTRACT, INQUIRE, PERFORM, RESYNC, and SET. Theses commands are used by system programmers. For programming information about these commands, see System commands.

SPACE(1 or 2 or 3)
(COBOL only)

SPACE indicates the type of spacing to be used in the output listing: SPACE(1) specifies single spacing, SPACE(2) double spacing, and SPACE(3) triple spacing. SPACE(3) is the default.

SPIE
SPIE specifies that the translator is to trap unrecoverable errors. SPIE is the default; the alternative is NOSPIE.

SYSEIB
SYSEIB indicates that the program is to use the system EIB instead of the application EIB. The SYSEIB option allows programs to execute CICS commands without updating the application EIB, making that aspect of execution transparent to the application. Use this option only in special situations because it imposes restrictions on the programs using it. A program translated with the SYSEIB option must:

- Execute in AMODE(31), because the system EIB is assumed to be located in TASKDATALOC(ANY) storage.
- Obtain the address of the system EIB using the ADDRESS EIB command (if the program is translated with the SYSEIB option, this command automatically returns the address of the system EIB).
- The use of the SYSEIB option implies the use of the NOHANDLE option on all CICS commands issued by the program. Commands can use the RESP option as required.

VBREF
(COBOL, C, C++, and PL/I only)

VBREF specifies whether the translator is to include a cross-reference of commands with line numbers in the translator listing. VBREF used to be called XREF , and is still accepted.)

Data definition (DD name) list

The DD name list must begin on a halfword boundary. The first two bytes contain a binary count of the number of bytes in the list (excluding the count field). Each entry in the list must occupy an 8-byte field.

The sequence of entries is:

Entry	Standard DD name	Entry	Standard DD name	Entry	Standard DD name
1	not applicable	3	not applicable	5	SYSIN
2	not applicable	4	not applicable	6	SYSPRINT
				7	SYSPUNCH

If you omit an applicable entry, the translator uses the standard DD name. If you use a DD name less than 8 bytes long, fill the field with blanks on the right. You can omit an entry by placing X'FF' in the first byte. You can omit entries at the end of the list entirely.

Using a CICS translator

A language translator reads your source program and creates a new one. Most normal language statements remain unchanged, but CICS commands are translated into CALL statements of the form required by the language in which you are coding.

The calls invoke CICS-provided "EXEC" interface modules, which are later link-edited into your load module, and these in turn invoke the requested services at execution time.

You can control the translation process by specifying translator options. See “Defining translator options” on page 592.

You can specify your choices in the following ways:

- List them as suboptions of the XOPTS option on the statement that the compiler (assembler) provides for specifying options.

Table 57. Statement provided for each language	
Language	**Statement**
COBOL	CBL [Note]
COBOL	PROCESS
C	#pragma
C++	#pragma
PL/I	* PROCESS
Assembler	*ASM or *PROCESS [Note]

- List your options in the PARM operand of the EXEC job control statement for the translate step.

  Most installations use cataloged procedures to translate, compile (assemble) and link CICS programs, and therefore you specify this PARM field in the EXEC job control statement that invokes the procedure.

  For example, if the name of the procedure for COBOL programs is DFHYITVL, and the name of the translate step in the procedure is TRN, the following example statement sets translator options for a COBOL program:

```
// EXEC
DFHYITVL,PARM.TRN=(VBREF,QUOTE,SPACE(2),NOCBLCARD)
```

If you use one method to specify an option and use the other method to specify the same option, or an option that conflicts, the specifications in the language statement override those in the EXEC statement. Similarly, if you specify multiple values for a single option or options that conflict on either type of statement, the last setting takes precedence. Except for COBOL programs, these statements must precede each source program; there is no way to batch the processing of multiple programs in other languages.

Translator options can appear in any order, separated by one or more blanks, or by a comma. If you specify them on the language statement for options, they must appear in parentheses following the XOPTS parameter, because other options are ignored by the translator and passed through to the compiler. The following COBOL example shows both translator and compiler options being passed together:

```
CBL RENT NODYNAM XOPTS(QUOTE SPACE(2))
```

The following examples show translator options being passed alone:

```
#pragma XOPTS(FLAG(W) SOURCE);
* PROCESS XOPTS(FLAG(W) SOURCE);
*ASM XOPTS(NOPROLOG NOEPILOG)
```

If you use the PARM operand of the EXEC job control statement to specify options, the XOPTS keyword is unnecessary, because the only options permitted here are translator options. However, you can use XOPTS, with or without its associated parentheses. If you use XOPTS with parentheses, be sure to enclose all of the translator options. For example, the following forms are valid:

```
PARM=(op1 op2 .. opn)
PARM=(XOPTS op1 op2 .. opn)
PARM=XOPTS(op1 op2 .. opn)
```

The following form is not valid:

```
PARM=(XOPTS(op1 op2) opn)
```

For compatibility with previous releases, the keyword CICS can be used as an alternative to XOPTS, except when you are translating batch EXEC DLI programs. Remember, if you alter the default margins for

C or C++ #pragma card processing using the PARM operand, the sequence margins should be altered too. You can do this using the NOSEQUENCE option.

Notes:

***ASM and *PROCESS**

1. For assembler language programs, *ASM statements contain translator options only. They are treated as comments by the assembler. *PROCESS statements can contain translator or assembler options for the High Level assembler, HLASM.
2. Translator and assembler options must not coexist on the same *PROCESS statement.
3. *PROCESS and *ASM statements must be at the beginning of the input and no assembler statements must appear before them. This includes comments and statements such as "PRINT ON" and "EJECT". Both *PROCESS and *ASM statements can be included, in any order.
4. *PROCESS statements containing only translator options contain information for the translator only and are not passed to the assembler.
5. *PROCESS statements containing assembler options are placed in the translated program.

CBL

The COBOL compiler LIB option is not required for COBOL Version 5 and later. The compiler options that result from the CBLCARD translator option do not include the LIB option. When translating source to be processed by COBOL compilers earlier than Version 5, you must specify the LIB option without relying on CBLCARD.

Defining translator options

You can specify the translator options that apply to all languages except where stated otherwise.

Table 58 on page 592 lists all the translator options, the program languages that apply, and any valid abbreviations.

If your installation uses the CICS-provided procedures in the distributed form, the default options are used. These are explicitly noted in the following option descriptions. You can tell which options get used by default at your installation by looking at the SYSPRINT translator listing output from the translate step (see “The CICS-supplied translators” on page 581). If you want an option that is not the default, you must specify it, as described in “Using a CICS translator” on page 590.

In Table 58 on page 592, the translator options are shown, in tabular form, along with the program languages to which they apply.

Table 58. Translator options applicable to programming language

Translator option	COBOL	C	C++	PL/I	Assembler
APOST or QUOTE	X				
CBLCARD or NOCBLCARD	X				
CICS	X	X	X	X	X
COBOL2	X				
COBOL3	X				
CPP			X		
CPSM or **NOCPSM**	X	X	X	X	X

Table 58. Translator options applicable to programming language (continued)

Translator option	COBOL	C	C++	PL/I	Assembler
DBCS	X				
DEBUG or NODEBUG	X	X	X	X	
DLI	X	X	X	X	X
EDF or NOEDF	X	X	X	X	X
EPILOG or NOEPILOG					X
EXCI	X	X	X	X	X
FEPI or **NOFEPI**	X	X	X	X	X
FLAG(I or **W** or E or S)	X	X	X	X	
GDS		X	X		X
GRAPHIC				X	
LEASM					X
LENGTH or NOLENGTH	X			X	X
LINECOUNT(n)	X	X	X	X	X
LINKAGE or NOLINKAGE	X				
MAIN or NOMAIN		X			
MARGINS(m,n)		X	X	X	
NATLANG	X	X	X	X	X
NUM or **NONUM**	X				
OPMARGINS(m,n[,c])		X	X	X	
OPSEQUENCE (m,n) or NOOPSEQUENCE		X	X	X	
OPTIONS or NOOPTIONS	X	X	X	X	X
PROLOG or NOPROLOG					X

Table 58. Translator options applicable to programming language (continued)					
Translator option	**COBOL**	**C**	**C++**	**PL/I**	**Assembler**
QUOTE or **APOST**	X				
SEQ or NOSEQ	X				
SEQUENCE(m,n) or NOSEQUENCE		X	X	X	
SOURCE or NOSOURCE		X	X	X	
SP	X	X	X	X	X
SPACE(1 or 2 or **3**)	X				
SPIE or NOSPIE	X	X	X	X	X
SYSEIB	X	X	X	X	X
VBREF or **NOVBREF**	X	X	X	X	

Using COPY statements

The compiler (or assembler) reads the translated version of your program as input, rather than your original source. This affects what you see on your compiler (assembler) listing. It also means that COPY statements in your source code must not bring in untranslated CICS commands, because it is too late for the translator to convert them.

About this task

If you are using a separate translator and the source within any copybook contains CICS commands, you must translate it separately before translation and compilation of the program in which it will be included. If you are using the integrated CICS translator and the source within any copybook contains CICS commands, you do not have to translate it separately before compilation of the program in which it will be included.

The external program must always be passed through the CICS translator, or compiled with a compiler that has an integrated CICS translator, even if all the CICS commands are in included copybooks.

The CICS-supplied interface modules

Each of your application programs to run under CICS requires one or more interface modules (also known as *stubs*) to use CICS facilities.

Application programs require a stub to use the following facilities:

- The EXEC interface
- The CPI Communications facility
- The SAA Resource Recovery facility
- The CICSPlex SM application programming interface.

The EXEC interface modules

Each of your CICS application programs must contain an interface to CICS . This takes the form of an EXEC interface module, used by the CICS high-level programming interface. The module, installed in the CICSTS56.CICS .SDFHLOAD library, must be link-edited with your application program to provide communication between your code and the EXEC interface program, DFHEIP.

The CPI Communications interface module

Each of your CICS application programs that uses the Common Programming Interface for Communications (CPI Communications) must contain an interface to CPI Communications. This takes the form of an interface module, used by the CICS high-level programming interface, common to all the programming languages. The module, DFHCPLC, that is installed in the CICSTS56.CICS .SDFHLOAD library, must be link-edited with each application program that uses CPI Communications.

The SAA Resource Recovery interface module

Each of your CICS application programs that uses SAA Resource Recovery must contain an interface to SAA Resource Recovery. This takes the form of an interface module, used by the CICS high-level programming interface, common to all the programming languages. The module, DFHCPLRR, that is installed in the CICSTS56.CICS .SDFHLOAD library, must be link-edited with each application program that uses the SAA Resource Recovery facility.

Using the EXEC interface modules for AMODE(24) and AMODE(31) applications

For AMODE(24) and AMODE(31) applications, the CALL statements that the language translators generate invoke EXEC interface modules that provide communication between your code and the CICS EXEC interface program, DFHEIP.

A language translator reads your source program and creates a new one. Normal language statements remain unchanged, but CICS commands are translated into CALL statements of the form required by the language in which you are coding. The calls invoke CICS-provided "EXEC" interface modules or *stubs* ; that is, function-dependent sections of code used by the CICS high-level programming interface. The stub, which is provided in the SDFHLOAD library, must be link-edited with your application program. These stubs are invoked during execution of EXEC CICS and EXEC DLI commands. Stubs are provided for each programming language.

Table 59. Interface modules (stubs)

Language	**Interface module name**
Assembler	DFHELII and DFHEAI0
All HLL languages and Assembler MAIN programs that use the LEASM option	DFHELII

The CICS-supplied stub routines work with an internal programming interface, the CICS command-level interface, which is never changed in an incompatible way. Consequently, these stub modules are compatible with earlier and later versions, and CICS application modules never need to be relinked to include a later level of any of these stubs.

Except for DFHEAI0, these stubs all provide the same function, which is to provide a linkage from EXEC CICS commands to the required CICS service. To do this, the stubs provide various entry points that are called from the translated EXEC CICS commands, and then execute a sequence of instructions that pass control to the EXEC interface function of CICS.

DFHELII contains multiple entry points, most of which provide compatibility for old versions of the CICS PL/I translator. It contains the entries DFHEXEC (for C and C++ application programs), DFHEI1 (for COBOL and Assembler), and DFHEI01 (for PL/I).

Each stub begins with an 8 byte eyecatcher in the form DFHY *xnnn* , where *x* indicates the language supported by the stub (A indicates Assembler, and I indicates that the stub is language independent), and *nnn* indicates the CICS release from which the stub was included. The letter Y in the eyecatcher indicates that the stub is read-only. Stubs supplied with very early releases of CICS contained eyecatchers in the form DFHE *xxxx* , in which the letter E denotes that the stub is not read-only. The eyecatcher for DFHELII in CICS Transaction Server for z/OS, Version 5 Release 6 is DFHYI 730.

The eyecatcher can be helpful to determine the CICS release at which a CICS application load module was most recently linked.

COBOL

Each EXEC command is translated into a COBOL CALL statement that refers to the entry DFHEI1.

The following example shows the output generated by the translator when processing a simple **EXEC CICS RETURN** command:

```
 *EXEC CICS RETURN END-EXEC
Call 'DFHEI1' using by content x'0e0800006000001000'
end-call.
```

The reference to DFHEI1 is resolved by the inclusion of the DFHELII stub routine in the linkage editor step of the CICS-supplied procedures such as DFHYITVL or DFHZITCL.

PL/I

When translating PL/I programs, each EXEC command generates a call to the entry point DFHEI01. This is done using a variable entry point DFHEI0 that is associated with the entry DFHEI01. The translator enables this by inserting the following statements near the start of each translated program:

```
 DCL DFHEI0 ENTRY VARIABLE INIT(DFHEI01) AUTO;
DCL DFHEI01 ENTRY OPTIONS(INTER ASSEMBLER);
```

The translator creates a unique entry name based on DFHEI0 for each successfully translated EXEC command. The following example shows the output generated by the translator when it processes a simple **EXEC CICS RETURN** command:

```
 /* EXEC CICS RETURN TRANSID(NEXT) */
DO;
DCL DFHENTRY_B62D3C38_296F2687 BASED(ADDR(DFHEI0)) OPTIONS(INTER ASSEM
BLER) ENTRY(*,CHAR(4));
CALL DFHENTRY_B62D3C38_296F2687('xxxxxxxxxxxxxxxx' /* '0E 08 80 00 03
00 00 10 00 F0 F0 F0 F0 F0 F0 F1 F0 'X */, NEXT);
END;
```

In the previous example, DFHENTRY_B62D3C38_296F2687 is based on the entry variable DFHEI0 that is associated with the real entry DFHEI01. This technique allows the translator to create a PL/I data descriptor list for each variable entry name. The PL/I compiler can then check that variable names referenced in EXEC commands are defined with attributes that are consistent with the attributes defined by the translator in the data descriptor list. In this example, ENTRY(*,CHAR(4)) specifies that the variable (named NEXT) associated with the TRANSID option should be a character string with a length of four.

The reference to DFHEI01 is resolved by the inclusion of the DFHELII stub routine in the linkage editor step of one of the CICS-supplied procedures such as DFHYITPL.

C and C++

In a C and C++ program, each EXEC CICS command is translated by the command translator into a call to the function DFHEXEC.

The translator enables this by inserting the following statements near the start of each translated program:

```
#pragma linkage(DFHEXEC,OS) /* force OS linkage */
void DFHEXEC(); /* function to call CICS */
```

The following example shows the output generated by the translator when processing a simple **EXEC CICS RETURN** command:

```
 /* EXEC CICS RETURN */
{
DFHEXEC(
"\x0E\x08\x00\x2F\x00\x00\x10\x00\xF0\xF0\xF0\xF0\xF1\xF8\xF0\xF0");
}
```

The reference to DFHEXEC is resolved by the inclusion of the DFHELII stub routine in the linkage editor step of one of the CICS-supplied procedures such as DFHYITDL, DFHZITDL, DFHZITEL, DFHZITFL, or DFHZITGL.

Assembler language

Each EXEC command is translated into an invocation of the DFHECALL macro.

The following example shows the output generated by the translator when processing a simple **EXEC CICS RETURN** command:

```
* EXEC CICS RETURN
DFHECALL =X'0E0800000800001000'
```

The assembly of the DFHECALL macro invocation as shown generates code that builds a parameter list addressed by register 1, loads the address of entry DFHEI1 in register 15, and issues a BALR instruction to call the stub routine.

```
 DS 0H
LA 1,DFHEITPL
LA 14,=x'0E08000008001000'
ST 14,0(,1)
OI 0(1),x'80'
L 15,=V(DFHEI1)
BALR 14,15
```

The reference to DFHEI1 is resolved by the inclusion of the DFHEAI stub routine in the linkage editor step of one of the CICS-supplied procedures such as DFHEITAL. The eyecatcher for DFHEAI in CICS Transaction Server for z/OS, Version 5 Release 6 is DFHYA 730 , with the release numbers indicating this stub was supplied with CICS Transaction Server for z/OS, Version 5 Release 6 .

The DFHEAI0 stub for assembler application programs is included by the automatic call facility of the linkage editor or binder utility. It is called by code generated by the DFHEIENT and DFHEIRET macros to obtain and free, respectively, an assembler application program's dynamic storage area. This stub is required only in assembler application programs; there are no stubs required or supplied to provide an equivalent function for programs written in the high-level languages.

For information about including DFHEAI, see DFHEAI.

Example assembler language program with LEASM

An example CICS assembler program and the corresponding code after it is translated demonstrate the result of translating an assembler program using the LEASM option. This translator option generates code for a Language Environment-conforming assembler MAIN program.

Figure 139 on page 598 shows a simple CICS assembler language program.

```
*ASM     XOPTS(LEASM)
DFHEISTG DSECT
OUTAREA  DS    CL200               DATA OUTPUT AREA
*
TESTLE   DFHEIENT CODEREG=R3
         MVC  OUTAREA(40),MSG1
         MVC  OUTAREA(4),EIBTRMID
         EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(43) FREEKB ERASE
         EXEC CICS RECEIVE
         MVC  OUTAREA(13),MSG2
         EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(13) FREEKB ERASE
         EXEC CICS RETURN
*
MSG1     DC   C'xxxx: ASM program invoked. ENTER TO END.'
MSG2     DC   C'PROGRAM ENDED'
         DFHREGS
CODEREG  EQU  R3
         END
```

Figure 139. A simple CICS assembler language program

The code below shows the result after the program is translated and assembled:

```
                                     1 *ASM
XOPTS(LEASM)
                                     2          DFHEIGBL ,,,LE            INSERTED BY
TRANSLATOR
                                     3+*,&DFHEIDL SETB 0  1 MEANS EXEC DLI IN
PROGRAM                           01-DFHEI
                                     4+*,&DFHEIDB SETB 0  1 MEANS BATCH
PROGRAM                               01-DFHEI
                                     5+*,&DFHEIRS SETB 0  1 MEANS
RSECT                                       01-DFHEI
                                     6+*,&DFHEILE SETB 1  1 MEANS LE
MAIN                                    01-DFHEI
000000                00000 002C8    7 DFHEISTG
DSECT
                                     8          DFHEISTG                  INSERTED BY
TRANSLATOR

10+*************************************************************************
                                    11+*              EXEC INTERFACE DYNAMIC
STORAGE                          *

12+*************************************************************************
000000                00000 002C8    13+DFHEISTG DSECT                     EXEC INTERFACE
STORAGE      @BBAC81A 01-DFHEI
               R:D  00000          14+         USING *,DFHEIPLR          ESTABLISH
ADDRESSABILITY    @BBAC81A 01-DFHEI

16+*

17+*************************************************************************
                                    18+*   D Y N A M I C   S T O R A G E   A R E A    ( D S
A )            *

19+*************************************************************************

20+*
000000                               21+CEEDSA   DS    0D         Just keep the same label for
formulae         02-CEEDS

22+*
000000                               23+CEEDSAFLAGS DS XL2                  DSA
flags                            02-CEEDS
                      01000          24+CEEDSALNGC EQU X'1000'               C library
DSA                     02-CEEDS
                      00800          25+CEEDSALNGP EQU X'0800'               PL/I library
DSA                   02-CEEDS
                      00008          26+CEEDSAEXIT EQU X'0008'               An Exit
DSA                       02-CEEDS
000002                               27+CEEDSAMEMD DS  XL2                  Member
defined                      02-CEEDS
000004                               28+CEEDSABKC DS   A                    Addr of DSA of
caller                02-CEEDS
000008                               29+CEEDSAFWC DS   A                    Addr of DSA of last
called rtn       02-CEEDS
00000C                               30+CEEDSAR14 DS   F                    Save area for
register 14           02-CEEDS
```

```
000010                                31+CEEDSAR15 DS   F                     Save area for
register 15           02-CEEDS
000014                                32+CEEDSAR0 DS    F                     Save area for
register 0            02-CEEDS
000018                                33+CEEDSAR1 DS    F                     Save area for
register 1            02-CEEDS
00001C                                34+CEEDSAR2 DS    F                     Save area for
register 2            02-CEEDS
000020                                35+CEEDSAR3 DS    F                     Save area for
register 3            02-CEEDS
000024                                36+CEEDSAR4 DS    F                     Save area for
register 4            02-CEEDS
000028                                37+CEEDSAR5 DS    F                     Save area for
register 5            02-CEEDS
00002C                                38+CEEDSAR6 DS    F                     Save area for
register 6            02-CEEDS
000030                                39+CEEDSAR7 DS    F                     Save area for
register 7            02-CEEDS
000034                                40+CEEDSAR8 DS    F                     Save area for
register 8            02-CEEDS
000038                                41+CEEDSAR9 DS    F                     Save area for
register 9            02-CEEDS
00003C                                42+CEEDSAR10 DS   F                     Save area for
register 10           02-CEEDS
000040                                43+CEEDSAR11 DS   F                     Save area for
register 11           02-CEEDS
000044                                44+CEEDSAR12 DS   F                     Save area for
register 12           02-CEEDS
000048                                45+CEEDSALWS DS   A                     Addr of PL/I
Language Working Space 02-CEEDS
00004C                                46+CEEDSANAB DS   A                     Addr of next
available byte         02-CEEDS
000050                                47+CEEDSAPNAB DS  A                     Addr of end-of-
prolog NAB           02-CEEDS
000054                                48+           DS
4F                                                    02-CEEDS
000064                                49+CEEDSATRAN DS  0A                    HPL TxArea
or                       02-CEEDS
000064                                50+CEEDSARENT DS  A                     Program reentry
address-IPAT        02-CEEDS
000068                                51+CEEDSACILC DS  A                     C to Fortran ILC
save area          02-CEEDS
00006C                                52+CEEDSAMODE DS  A                     Return address of
module that       02-CEEDS
                                      53+*                                  caused the last
mode switch
000070                                54+           DS
2F                                                    02-CEEDS
000078                                55+CEEDSARMR  DS  A                     Addr of language
specific           02-CEEDS
                                      56+*                                  exception
handler

57+*
```

```
00007C                                58+           DS  F
Reserved                             02-CEEDS
000080                                59+CEEDSAAUTO DS  0D                    Automatic storage
starts here       02-CEEDS
000080                                60+CEEDSAEND  DS  0D                    End of
DSA                        02-CEEDS
                     00080            61+CEEDSASZ EQU   CEEDSAEND-CEEDSA     Size of
DSA                           02-CEEDS
                     00000            62+CEEDSA_STDCEEDSA  EQU   X'0000'     flag values of
standard CEE DSA     02-CEEDS

63+*

64+*

65+*
000080                                66+DFHEISA  DS    18F                 SAVE AREA R14-R12 AT
12 OFF @BBAC81A 01-DFHEI
0000C8                                67+DFHEILWS DS    F
RESERVED                     @BBAC81A 01-DFHEI
0000CC                                68+DFHEINAB DS    F
RESERVED                     @BBAC81A 01-DFHEI
0000D0                                69+DFHEIRS0 DS    F
RESERVED                     @BBAC81A 01-DFHEI
0000D4                                70+DFHEIR13 DS    F                   REGISTER
13                  @BBAC81A 01-DFHEI
```

```
0000D8                                71+DFHEIRS1 DS    F
RESERVED                    @BBAC81A 01-DFHEI
0000DC                                72+DFHEIBP  DS    F                   EIB POINTER
(NOT USED IF BATCH) 01-DFHEI
0000E0                                73+DFHEICAP DS    F                   COMMAREA POINTER
(NOT USED IF BATCH) 01-DFHEI
0000E4                                74+DFHEIV00 DS    H                   HALFWORD TEMP USED
BY DFHECALL       01-DFHEI
0000E6                                75+DFHEIRS2 DS    H
RESERVED                    @BBAC81A 01-DFHEI
0000E8                                76+DFHEIPL  DS    13F                 PARAMETER
LIST                  @05C 01-DFHEI
00011C                                77+         DS    51F                 ALLOW 64 PARAMETERS
FOR DLI     @L2A 01-DFHEI
                                      78+*                                AND IN XA2 ON, FOR EXEC
CICS ALSO
0001E8                                79+DFHEIRS3 DS    F                FULLWORD TEMP USED BY
DFHECALL     @06C 01-DFHEI
0001EC                                80+DFHEIRS4 DS    F
RESERVED                         @L2A 01-DFHEI
0001F0                                81+DFHEITP1 DS    F                   TEMPORARY POINTER
1             @L2A 01-DFHEI
0001F4                                82+DFHEITP2 DS    F                   TEMPORARY POINTER
2             @L2A 01-DFHEI
0001F8                                83+DFHEITP3 DS    F                   TEMPORARY POINTER
3             @L2A 01-DFHEI
0001FC                                84+DFHEITP4 DS    F                   TEMPORARY POINTER
4             @L2A 01-DFHEI

85+*************************************************************************
                                      86+*              START DEFINITION OF USER DYNAMIC
STORAGE              *

87+*************************************************************************
000200                                88+DFHEIUSR DS    0D                  ALIGN USER DYNAMIC
STORAGE  @BBAC81A 01-DFHEI

89+*
000200                                90 OUTAREA  DS    CL200               DATA OUTPUT
AREA
                                      91
*
                                      92 TESTLE   DFHEIENT
CODEREG=R3
0002C8                               186+         DC    0SL2(((R3-12)/
(R3-12))-1)                                X01-DFHEI
                                       +                Assembly error if register 12 used for
CODEREG     @PBA
0002C8                               187+         DC    0SL2(((R3-2)/
(R3-2))-1)                                 X01-DFHEI
                                       +                Assembly error if register 2 used for
CODEREG      @PBA

188+*************************************************************************


189+*                                                                      *
                                     190+* CONTROL BLOCK NAME =
DFHEIBLK                                       *

191+*                                                                      *
                                     192+* NAME OF MATCHING PL/AS CONTROL BLOCK =
None                          *

193+*                                                                      *
                                     194+* DESCRIPTIVE NAME = CICS TS  EXEC Interface
Block.                    *

195+*                                                                      *

196+*                                                                      *

197+*                                                                      *
                                     198+*        Licensed Materials - Property of
IBM                           *

199+*                                                                      *
                                     200+*        "Restricted Materials of
IBM"                                 *

201+*                                                                      *
                                     202+*        5655-
```

```
Y04                                                    *
203+*                                                                    *
```

```
                                204+*        (C) Copyright IBM Corp. 1990,
1993"                        *
205+*                                                                    *
206+*                                                                    *
207+*                                                                    *
208+*                                                                    *
                                209+* STATUS =
7.3.0                                                    *
210+*                                                                    *
                                211+* FUNCTION = EXEC Interface
Block.                                  *
212+*                                                                    *
                                213+*        The exec interface block contains information
on the         *
                                214+*        transaction identifier, the time and date, and
the cursor    *
                                215+*        position on a display device. Some of the
other fields are   *
                                216+*        set indicating the next action that a program
should take    *
                                217+*        in certain
circumstances.                                   *
                                218+*        DFHEIBLK also contains information that will
be helpful      *
                                219+*        when a dump is being used to debug a
program.                *
                                220+*        This control block is included automatically
by an           *
                                221+*        application program using the command-level
interface.       *
                                222+*        EISEIBA in the EIS addresses the
EIB.                       *
223+*                                                                    *
224+*                                                                    *
225+*                                                                    *
                                226+*
NOTES :                                                          *
                                227+*    DEPENDENCIES =
S/370                                           *
                                228+*    MODULE TYPE = Control block
definition                       *
                                229+*    PROCESSOR =
Assembler                                           *
230+*                                                                    *
231+*------------------------------------------------------------------- *
232+*                                                                    *
                                233+* CHANGE
ACTIVITY :                                                  *
                                234+*   $SEG(DFHEIBLK),COMP(COMMAND),PROD(CICS
TS ) :                     *
235+*                                                                    *
                                236+*     PN= REASON REL YYMMDD HDXXIII :
REMARKS                        *
                                237+*    $L1= 550    321 900515 HDFSPC  : Add an EIB length
equate        *
                                238+*    $D1= I05119 410 930226 HDDHDMA : Correct comments
for date field *
                                239+*    $P1= M60581 320 900116 HDAEGB  : Change for PLXMAP
to data areas *
240+*                                                                    *
241+*********************************************************************
```

```
                                    242+*              EXEC INTERFACE
BLOCK                                   *

243+*************************************************************************

000000                00000 00055   244+DFHEIBLK DSECT                     EXEC INTERFACE
BLOCK        @BBAC81A 01-DFHEI
                 R:B  00000         245+         USING
*,DFHEIBR                                      @BBAC81A 01-DFHEI
000000                              246+EIBTIME  DS    PL4                 TIME IN 0HHMMSS
FORMAT      @BBAC81A 01-DFHEI
000004                              247+EIBDATE  DS    PL4                 DATE IN 0CYYDDD+
FORMAT,        @D1C 01-DFHEI
                                    248+*                                  where C is the
century          @D1A
                                    249+*                                  indicator (0=1900,
1=2000),     @D1A
                                    250+*                                  YY is the year, DDD
is the      @D1A
                                    251+*                                  day number and '+'
is the       @D1A
                                    252+*                                  sign byte
(positive)            @D1A
000008                              253+EIBTRNID DS    CL4                 TRANSACTION
IDENTIFIER      @BBAC81A 01-DFHEI
00000C                              254+EIBTASKN DS    PL4                 TASK
NUMBER                 @BBAC81A 01-DFHEI
000010                              255+EIBTRMID DS    CL4                 TERMINAL
IDENTIFIER         @BBAC81A 01-DFHEI
000014                              256+EIBRSVD1 DS    H
RESERVED                    @BBAC81A 01-DFHEI
000016                              257+EIBCPOSN DS    H                   CURSOR
POSITION             @BBAC81A 01-DFHEI
000018                              258+EIBCALEN DS    H                   COMMAREA
LENGTH             @BBAC81A 01-DFHEI
```

```
00001A                              259+EIBAID   DS    CL1                 ATTENTION
IDENTIFIER        @BBAC81A 01-DFHEI
00001B                              260+EIBFN    DS    CL2                 FUNCTION
CODE               @BBAC81A 01-DFHEI
00001D                              261+EIBRCODE DS    CL6                 RESPONSE
CODE               @BBAC81A 01-DFHEI
000023                              262+EIBDS    DS    CL8                 DATASET
NAME                @BBAC81A 01-DFHEI
00002B                              263+EIBREQID DS    CL8                 REQUEST
IDENTIFIER          @BBAC81A 01-DFHEI
000033                              264+EIBRSRCE DS    CL8                 RESOURCE
NAME               @BBDIA0U 01-DFHEI
00003B                              265+EIBSYNC  DS    C                   X'FF' SYNCPOINT
REQUESTED   @BBDIA0U 01-DFHEI
00003C                              266+EIBFREE  DS    C                   X'FF' FREE
REQUESTED        @BBDIA0U 01-DFHEI
00003D                              267+EIBRECV  DS    C                   X'FF' RECEIVE
REQUIRED      @BBDIA0U 01-DFHEI
00003E                              268+EIBSEND  DS    C
RESERVED                    @BM13417 01-DFHEI
00003F                              269+EIBATT   DS    C                   X'FF' ATTACH
RECEIVED       @BBDIA0U 01-DFHEI
000040                              270+EIBEOC   DS    C                   X'FF' EOC
RECEIVED          @BBDIA0U 01-DFHEI
000041                              271+EIBFMH   DS    C                   X'FF' FMHS
RECEIVED         @BBDIA0U 01-DFHEI
000042                              272+EIBCOMPL DS    C                   X'FF' DATA
COMPLETE                  01-DFHEI
000043                              273+EIBSIG   DS    C                   X'FF' SIGNAL
RECEIVED                01-DFHEI
000044                              274+EIBCONF  DS    C                   X'FF' CONFIRM
REQUESTED              01-DFHEI
000045                              275+EIBERR   DS    C                   X'FF' ERROR
RECEIVED                 01-DFHEI
000046                              276+EIBERRCD DS    CL4                 ERROR CODE
RECEIVED                  01-DFHEI
00004A                              277+EIBSYNRB DS    C                   X'FF' SYNC ROLLBACK
REQ'D            01-DFHEI
00004B                              278+EIBNODAT DS    C                   X'FF' NO APPL DATA
RECEIVED          01-DFHEI
00004C                              279+EIBRESP  DS    F                   INTERNAL CONDITION
NUMBER            01-DFHEI
000050                              280+EIBRESP2 DS    F                   MORE DETAILS ON SOME
RESPONSES       01-DFHEI
000054                              281+EIBRLDBK DS    CL1                 ROLLED
```

```
BACK                          01-DFHEI

282+*

                      00055         283+EIBLENG  EQU   *-EIBTIME           Length of
EIB                   @L1A 01-DFHEI

284+*************************************************************************

                                    285+*              END OF EXEC INTERFACE
BLOCK                            *

286+*************************************************************************

                      0000B         287+DFHEIBR  EQU   11                  EIB
REGISTER                @BA02936 01-DFHEI
```

```
290+*************************************************************************
                                    291+*              PROLOG CODE FOR EXEC
INTERFACE                         *

292+*************************************************************************
                                    293+*&DFHEICS CEEENTRY
PPA=DFHPPA,MAIN=YES,PLIST=OS,
                                    294+*
BASE=&CODEREG,
                                    295+*               AUTO=(DFHEIEND-
DFHEISTG)
000000                00000 0020C   297+TESTLE
CSECT ,                                                    02-CEEEN
                                    298+TESTLE  RMODE
ANY                                               @D2A  02-CEEEN
                                    299+TESTLE  AMODE
ANY                                               @D2A  02-CEEEN
                                    300+           ENTRY
TESTLE                                               02-CEEEN
                                    301+         PUSH
USING                                                  02-CEEEN
                                    302+
DROP  ,                                                @02A  02-CEEEN
               R:F  00000           303+         USING
*,15                                                   02-CEEEN
000000 47F0 F014            00014   304+         B
CEEZ0009                                               02-CEEEN
000004 00C3C5C5                     305+         DC
X'00C3C5C5'                                      @D3C  02-CEEEN
                      002C8         306+CEEY0009    EQU (((DFHEIEND-DFHEISTG)+7)/
8)*8                          X02-CEEEN
                                       +                                . Size of automatic
storage.    @P1A
000008 000002C8                     307+         DC    A(CEEY0009)        . Size of automatic
storage.    @P1C  02-CEEEN
00000C 00000080                     308+         DC    A(DFHPPA-TESTLE)   . Address of PPA for
this program     02-CEEEN
000010 47F0 F001            00001   309+         B
1(,15)                                                 02-CEEEN
                      00014         310+CEEZ0009    EQU
*                                                       02-CEEEN
000014 90EC D00C            0000C   311+         STM   14,12,CEEDSAR14-
CEEDSA(13)                                  02-CEEEN
000018 5820 F050            00050   312+         L     2,CEEINPL0009
R2=addr(CEEINPL)        5@01D @01C  02-CEEEN
00001C 58F0 F054            00054   313+         L     15,CEEINT0009
R15=addr(CEEINT)              @01C  02-CEEEN
                                    314+         DROP
15                                                @01A  02-CEEEN
000020 05EF                         315+         BALR  14,15                Call CEEINT to init
LE               02-CEEEN
000022 1821                         316+         LR    2,1                  Save input R1 value
temporarily      02-CEEEN
000024 58E0 C2F0            002F0   317+         L     14,752(,12)          Get EDB
address                     02-CEEEN
000028 9680 E008      00008         318+         OI    8(14),X'80'          Turn CEEEDBMAINI
flag ON             02-CEEEN
00002C 5810 D04C            0004C   319+         L     1,CEEDSANAB-CEEDSA(,13)  Get the current
NAB              02-CEEEN
000030 A502 0000                    320+         IILH  0,CEEY0009/65536        Load high half
of AUTO size@P1A  02-CEEEN
000034 A503 02C8                    321+         IILL  0,CEEY0009-(CEEY0009/65536*65536)
and low    @P1A  02-CEEEN
```

```
000038 1E01                          322+         ALR   0,1                      Compute new
value.               02-CEEEN
00003A 5500 C00C           0000C   323+         CL    0,CEECAAEOS-CEECAA(,12)  Compare with
EOS.              02-CEEEN
00003E A7D4 000D           00058   324+         JNH
CEEX0009                                          @P1C  02-CEEEN
000042 58F0 C2BC           002BC   325+         L     15,CEECAAGETS-CEECAA(,12) Get address
overflow routine   02-CEEEN
000046 05EF                        326+         BALR  14,15                    Get another
stack segment.       02-CEEEN
000048 181F                        327+         LR
1,15                                                02-CEEEN
00004A A7F4 0007           00058   328+         J     CEEX0009                 Branch around
statics      @P1C  02-CEEEN
00004E
0000

000050 000000A8                    329+CEEINPL0009    DC
A(CEEINPL)                                  @01A  02-CEEEN
000054 00000000                    330+CEEINT0009     DC
V(CEEINT)                                   @01A  02-CEEEN
                      00058        331+CEEX0009    EQU
*                                                    02-CEEEN
000058 50D0 1004           00004   332+         ST    13,CEEDSABKC-CEEDSA(,1)  Set back
chain.                  02-CEEEN
00005C 5000 104C           0004C   333+         ST    0,CEEDSANAB-CEEDSA(,1)   Set new NAB
value                02-CEEEN
000060 D701 1000 1000 00000 00000  334+         XC    CEEDSAFLAGS-CEEDSA(,1),CEEDSAFLAGS-
CEEDSA(1) . Clear     02-CEEEN
000066 5010 D008           00008   335+         ST    1,CEEDSAFWC-CEEDSA(,13)  Set forward
chain.              02-CEEEN
00006A 18D1                        336+         LR    13,1                     Set save area
address            02-CEEEN
                                   337+         POP   USING                    Clear any
temporary usings @P1M  02-CEEEN
                     R:D  00000    338+         USING CEEDSA,13                Addresability to
SF     V1R2M0   02-CEEEN
00006C D203 D048 C280 00048 00280  339+         MVC   CEEDSALWS,CEECAALWS-CEECAA(12) Get LWS
addr     V1R2M0   02-CEEEN
000072 1812                        340+         LR    1,2                      Move input r1
value to PARMREG   02-CEEEN
000074 C030 FFFF FFC6      00000   341+         LARL  R3,TESTLE                 Load EP into
1st base reg @P1C  02-CEEEN
                   R:3  00000      342+         USING
TESTLE,R3                                         @P1M  02-CEEEN
00007A A7F4 003F           000F8   343+         J     SKIPLE0005          Branch round LE
definitions     @PCC 01-DFHEI
```

```
                                   345+*/
*********************************************************************/

346+*/*                                                              */
                                   347+*/* Licensed Materials - Property of
IBM                           */

348+*/*                                                              */
                                   349+*/* 5650-ZOS
5688-198                                        */

350+*/*                                                              */
                                   351+*/* Copyright IBM Corp. 1991,
2017                               */

352+*/*                                                              */
                                   353+*/* Status =
HLE77B0                                         */

354+*/*                                                              */
                                   355+*/
*********************************************************************/

356+*


357+*****************************************************************************

                                   358+*        P R O G R A M  P R O L O G  A R E A 1   ( P
P A 1)          *

359+*****************************************************************************
```

```
360+*

000080                              361+PPA10010    DS
0F                                                    02-CEEPP
000080                              362+DFHPPA   DS
0F                                                      02-CEEPP
000080 20                           363+         DC    AL1(PPANL0010-*) Offset to the entry
name length         02-CEEPP
000081 CE                           364+         DC    X'CE'            LE/370
Indicator.                       02-CEEPP
000082 A0                           365+         DC
B'10100000'                             . PPA flags      02-CEEPP

366+*                                                                 @P4C
                                    367+*                     Bit 0   0 = Internal
Procedure
                                    368+*                             1 = External
Procedure
                                    369+*                     Bit 1   0 = Primary Entry
Point
                                    370+*                             1 = Secondary Entry
Point
                                    371+*                     Bit 2   0 = Block doesn't have a
DSA
                                    372+*                             1 = Block has a
DSA
                                    373+*                     Bit 3   0 = compiled
object
                                    374+*                             1 = library
object
                                    375+*                     Bit 4   0 = sampling interrupts
to library
                                    376+*                             1 = sampling interrupts
to code
                                    377+*                     Bit 5   0 = not an exit
DSA
                                    378+*                             1 = Exit
DSA
                                    379+*                     Bit 6   0 = own exception
model
                                    380+*                             1 = inherited (callers)
exception model
                                    381+*                     Bit 7
Reserved                         @P4C
000083 00                           382+         DC    X'00'            Member
flags                             02-CEEPP
000084 000000D0                     383+         DC    A(PPA20010)      Addr of Compile Unit
Block (PPA2) @D4A  02-CEEPP
000088 00000000                     384+         DC    A(0)             Blk Debug
Info                             02-CEEPP
00008C 00000000                     385+         DC    A(0)             Data Descriptors for
this entry point   02-CEEPP
000090 00000000                     386+         DC    A(0)             GPR save bit mask
X'10'      @D2A  02-CEEPP
000094 00000000                     387+         DC    A(0)             Member PPA1 word
X'14'      @D2A  02-CEEPP
000098 00000000                     388+         DC    A(0)                               Offset
X'18'      @D2A  02-CEEPP
                                    389+* Language Environment flags (16bits)          Offset
X'1C'      @D2A
00009C 00                           390+         DC
B'00000000'                               @D2A  02-CEEPP
                                    391+*                  Bit 0-1 00 = Old code entry performs
full save  @D2A
                                    392+*                          01 = Old code performs
partial save     @D2A
                                    393+*                          10 = Old code performs
partial save+R12 @D2A
                                    394+*                  Bit 2   0  = Allow asynchronous
exceptions      @D2A
                                    395+*                          1  = Defer asynchronous
exceptions      @D2A
                                    396+*                  Bit 3   0  = Word 0 of SA not
initialized       @D2A
                                    397+*                          1  = Word 0 of SA
initialized           @D2A
                                    398+*                  Bit 4   0  = Code is nonexternal
glue           @D2A
                                    399+*                          1  = Code is external
glue              @D2A
```

```
                                    400+*                 Bit 5   0  = Real return addr saved
in SA at    @D2A
                                    401+*                              offset
'0C'X                         @D2A
                                    402+*                         1  = Real return addr saved
in linkage  @D2A
                                    403+*
area                               @D2A
                                    404+*                 Bit 6   0  = Storage argument area
start        @D2A
                                    405+*
indeterminate                      @D2A
                                    406+*                         1  = Storage argument area
start valid  @D2A
                                    407+*                 Bit 7   0  = R12 must contain CAA
address upon  @D2A
                                    408+*                              old code
entry                        @D2A
                                    409+*                         1  = R12 not defined upon old
code entry@D2A
00009D 00                           410+          DC
B'00000000'                                         @P4C  02-CEEPP
                                    411+*                 Bit 8   0  = Not vararg
routine                   @D2A
                                    412+*                         1  = Vararg
routine                       @D2A
                                    413+*                 Bit 9   0  = Asynchronous interrupts
unsupported@D2A
                                    414+*                         1  = Asynchronous interrupts
supported  @D2A
                                    415+*                 Bit 10  0  = No module service
level            @D2A
                                    416+*                         1  = Module service level
applied       @D2A
                                    417+*                 Bit 11-13  =
Reserved                           @P4C
                                    418+*                 Bit 14  0  = extended flag area not
present     @P4A
                                    419+*                         1  = extended flag area
present         @P4A
                                    420+*                 Bit 15     =
Reserved                           @P4A
00009E 0000                         421+          DC    AL2(0)           Offset/2 to code desc
list        @P4A  02-CEEPP
0000A0                              422+          DS
0H                                                      02-CEEPP
0000A0 0006                         423+PPANL0010 DC    AL2(6)        .  Length of Entry Point
Name               02-CEEPP
0000A2 E3C5E2E3D3C5                 424+          DC    CL6'TESTLE'   .  Entry Point
Name                          02-CEEPP
0000A8                              425+CEEINPL   DS
0D                                                      02-CEEPP
0000A8 000000CC                     426+          DC    A(INPLEPA)  . A of A(first entry point
in CU)      @D4C  02-CEEPP
0000AC 00000008                     427+          DC    A(CEEINPLSTST-
CEEINPL)                                    02-CEEPP
0000B0                              428+CEEINPLSTST DS
0F                                                      02-CEEPP
0000B0 00                           429+          DC    X'00'            Control
Level                      @01A  02-CEEPP
0000B1 00                           430+          DC    X'00'
ENCLAVE=NO                        @01A  02-CEEPP
0000B2 00                           431+          DC
X'00'                                                 @01A  02-CEEPP
0000B3 07                           432+          DC    X'07'            Number of
items.                   @01C  02-CEEPP
0000B4 000000CC                     433+          DC    A(INPLEPA)  . A of A(first entry point
in CU)      @D4C  02-CEEPP
0000B8 00000000                     434+          DC    V(CEESTART)   . A(Address of
CEESTART)                     02-CEEPP
0000BC 00000000                     435+          DC
V(CEEBETBL)                                             02-CEEPP
0000C0 0000000F                     436+          DC    A(15)         . Memeber
id                                 02-CEEPP
0000C4 00000000                     437+          DC
A(0)                                                    02-CEEPP
0000C8 00070000                     438+          DC    XL4'00070000'          . EXECOPS(ON),
PLIST              02-CEEPP
0000CC 00000000                     439+INPLEPA   DC    A(TESTLE)      . A(first entry point in
CU)        @D4A  02-CEEPP
0000D0                              440+          DS
```

```
0H                                                        02-CEEPP

441+*


442+************************************************************************

                                    443+*        P R O G R A M  P R O L O G  A R E A 2   ( P P
A 2)          *

444+************************************************************************


445+*

                                    446+         EXTRN
CEESTART                                                  02-CEEPP
0000D0                              447+PPA20010    DS
0F                                                   02-CEEPP
0000D0 0F                           448+         DC    AL1(15)          Member
ID                                02-CEEPP
0000D1 00                           449+         DC    AL1(0)           Sub
ID                                  02-CEEPP
0000D2 00                           450+         DC    AL1(0)           Member
defined                           02-CEEPP
0000D3 01                           451+         DC    AL1(1)           Level of PPAx control
blocks      @D4C  02-CEEPP
0000D4 00000000                     452+PPA2S0010   DC    A(CEESTART)      A(CEESTART for this
load module)     02-CEEPP
0000D8 00000000                     453+         DC    A(0)             A(Compile Debug
Information (CDI) )     02-CEEPP
0000DC 00000014                     454+         DC    A(CEETIMES-PPA20010)   A(Offset to time
stamp)           02-CEEPP
```

```
0000E0 00000000                     455+PPA2M0010   DC    A(TESTLE)      . A(first entry point
in comp. unit)   02-CEEPP

456+*


457+************************************************************************

                                    458+*                T I M E    S T A M
P                                *

459+************************************************************************


460+*

                                    461+*        Time
Stamp
                                    462+*,Time Stamp = 2020/02/19
14:08:00                                      02-CEEPP
                                    463+*,Version 1 Release 1 Modification
0                                    02-CEEPP
0000E4                              464+CEETIMES DS
0F                                                        02-CEEPP
0000E4 F2F0F2F0                     465+         DC    CL4'2020'
Year                                  02-CEEPP
0000E8 F0F2                         466+         DC    CL2'02'
Month                                 02-CEEPP
0000EA F1F9                         467+         DC    CL2'19'
Day                                   02-CEEPP
0000EC F1F4                         468+         DC    CL2'14'
Hours                                 02-CEEPP
0000EE F0F8                         469+         DC    CL2'08'
Minutes                               02-CEEPP
0000F0 F0F0                         470+         DC    CL2'00'
Seconds                               02-CEEPP
0000F2 F140                         471+         DC    CL2'1'
Version                               02-CEEPP
0000F4 F140                         472+         DC    CL2'1'
Release                               02-CEEPP
0000F6 F040                         473+         DC    CL2'0'
Modification                          02-CEEPP

475+************************************************************************

                                    476+*  C O M M O N   A N C H O R   A R E A   ( C A
A )                    *
```

```
477+*********************************************************************

478+*

480+*********************************************************************

481+*********************************************************************

                     00004         482+LEPTRLEN EQU
4                                                  03-CEEDN

483+*

000000               00000 00400   484+CEECAA   DSECT    ,                CAA
mapping                         02-CEECA
000000                             485+CEECAAFLAG0 DS    X                CAA
Flags                           02-CEECA
                     00002         486+CEECAAXHDL  EQU   X'02'             Bypass exception
handling          02-CEECA
                                   487+*           EQU   X'FD'
Reserved
000001                             488+            DS    X
Reserved                           02-CEECA
                     00020         489+CEECAADBGINIT EQU X'20'             Debugger is
initialized            02-CEECA
000002                             490+CEECAALANGP DS    X                PL/I compatibility
flags            02-CEECA
                     00008         491+CEECAATHFN  EQU   X'08'             If set, NO PL/I
FINISH             02-CEECA
                                   492+*                                  on-unit
active
                                   493+*           EQU   X'F7'
Reserved
000003                             494+            DS    XL5
Reserved                           02-CEECA
000008                             495+CEECAABOS   DS    A                Start of current
storage segment    02-CEECA
00000C                             496+CEECAAEOS   DS    A                End of current
storage segment      02-CEECA
000010                             497+            DS    XL52
Reserved                           02-CEECA
000044                             498+CEECAATORC  DS    F                Thread level return
code            02-CEECA
000048                             499+            DS    XL44
Reserved                @CM0419C   02-CEECA
000074                             500+CEECAATOVF  DS    A                Stack overflow
routine   @CM0419A   02-CEECA
000078                             501+            DS    XL168
Reserved                @CM0419A   02-CEECA
000120                             502+CEECAAATTN  DS    A                Addr of LE/370
attention handler    02-CEECA

503+*

000124                             504+            DS    XL56
Reserved                           02-CEECA
00015C                             505+CEECAAHLLEXIT DS  A                Set by CEEBINT for
user hook exit   02-CEECA
000160                             506+            DS    XL56
Reserved                           02-CEECA
000198                             507+CEECAAHOOK  DS    XL12             Code to pass
control                02-CEECA
0001A4                             508+CEECAADIMA  DS    A                A(debugger
entry)                   02-CEECA

509+*

0001A8                             510+CEECAAHOOKS DS    0CL68            Hook control words
for debugger@G3C 02-CEECA
0001A8                             511+CEECAAALLOC DS    XL4              ALLOCATE descr.
built               02-CEECA
```

```
0001AC                             512+CEECAASTATE DS    XL4              New statement
begins               02-CEECA
0001B0                             513+CEECAAENTRY DS    XL4              Block
entry                       02-CEECA
0001B4                             514+CEECAAEXIT  DS    XL4              Block
```

```
exit                          02-CEECA
0001B8                              515+CEECAAMEXIT DS    XL4               Multiple block
exit                 02-CEECA
0001BC                              516+CEECAAPATHS DS    0CL32             PATH
hooks                         02-CEECA
0001BC                              517+CEECAALABEL DS    XL4               At a label
constant                   02-CEECA
0001C0                              518+CEECAABCALL DS    XL4               Before
CALL                          02-CEECA
0001C4                              519+CEECAAACALL DS    XL4               After
CALL                           02-CEECA
0001C8                              520+CEECAADO    DS    XL4               DO block
starting                     02-CEECA
0001CC                              521+CEECAAIFTRUE  DS   XL4              True part of
IF                      02-CEECA
0001D0                              522+CEECAAIFFALSE DS   XL4              False part of
IF                      02-CEECA
0001D4                              523+CEECAAWHEN    DS   XL4              WHEN group
starting                   02-CEECA
0001D8                              524+CEECAAOTHER   DS   XL4              OTHERWISE
group                       02-CEECA
0001DC                              525+CEECAACGOTO   DS   XL4              GOTO hook for
C                       02-CEECA
0001E0                              526+CEECAARSVDH1  DS   XL4              Reserved
hook                         02-CEECA
0001E4                              527+CEECAARSVDH2  DS   XL4              Reserved
hook                         02-CEECA
0001E8                              528+CEECAAMULTEVT DS   XL4              Multiple Event
Hook            @G3C 02-CEECA

529+*

0001EC                              530+CEECAAMEVMASK DS   XL4              Multiple Event Hook
Mask       @G3A 02-CEECA
0001F0                              531+CEECAACGENE DS    A
Reserved                            02-CEECA
0001F4                              532+CEECAACRENT DS    A                 C/370 writable
static                02-CEECA
0001F8                              533+CEECAACFLTINIT DS XL8               Used to covert
fixed to float        02-CEECA
000200                              534+CEECAACPRMS DS    A                 Parms passed to
IBMLIIA             02-CEECA
000204                              535+CEECAAC_RTL          DS   0F        Combination of 24
unique C/370 @DJC 02-CEECA
                                    536+*                                  trc types & 8
common trc types
000204                              537+CEECAAC_RTL_1        DS    X        C/370 RTL unique
trace options @DJA 02-CEECA

538+*                                                                 @DJA
000205                              539+CEECAAC_RTL_2        DS    X        C/370 RTL unique
trace options @DJA 02-CEECA

540+*                                                                 @DJA
000206                              541+CEECAAC_RTL_3        DS    X        C/370 RTL unique
trace options @DJA 02-CEECA
                                    542+*                    EQU   X'80'
Reserved                      @DJA
                     00040          543+CEECAA_SIGNALS_L     EQU   X'40'    Low-level
signals              @DJA 02-CEECA
                     00020          544+CEECAA_LOW_IO        EQU   X'20'    Low-level
I/O                  @DJA 02-CEECA
                     00010          545+CEECAA_INITTERM_L    EQU   X'10'    Low-level init/
term            @DJA 02-CEECA
                                    546+*                    EQU   X'08'
Reserved                      @DJA
                     00004          547+CEECAA_SIGNALS_H     EQU   X'04'    High-level
signals             @DJA 02-CEECA
                     00002          548+CEECAA_HIGH_IO       EQU   X'02'    High-level
I/O                  @DJA 02-CEECA
                     00001          549+CEECAA_INITTERM_H    EQU   X'01'    High-level init/
term           @DJA 02-CEECA

550+*                                                                 @DJA
000207                              551+CEECAAC_COMTRACE     DS    X        Common RTL trace
options       @DJA 02-CEECA
                                    552+*                    EQU   X'80'
Reserved                      @DJA
                                    553+*                    EQU   X'40'
Reserved                      @DJA
                     00020          554+CEECAA_RTLXPLI       EQU   X'20'    RTL XPLINK
trace                @DJA 02-CEECA
```

```
                     00010          555+CEECAA_RTLCICS       EQU   X'10'    RTL CICS
trace               @DJA 02-CEECA
                     00008          556+CEECAA_RTLALLOC      EQU   X'08'    RTL Alloc
trace               @DJA 02-CEECA
                     00004          557+CEECAA_RTLCOUNT      EQU   X'04'    RTL Function
counting          @DJA 02-CEECA
                     00002          558+CEECAA_RTLLOCKS      EQU   X'02'    RTL or user
locking            @DJA 02-CEECA
                     00001          559+CEECAA_RTLFUNC       EQU   X'01'    RTL function entry/
exit        @DJA 02-CEECA

560+*

000208                              561+CEECAACTHD  DS
A                                                     02-CEECA
00020C                              562+CEECAACURRFECB DS
A                                                     02-CEECA
000210                              563+CEECAAEDCV  DS    A                C/370 vector
table                  02-CEECA
000214                              564+CEECAACPCB  DS
A                                                     02-CEECA
000218                              565+CEECAACEDB  DS    A                C/370
CEDB                         02-CEECA
00021C                              566+            DS
XL3                                                   02-CEECA
```

```
00021F                              567+CEECAASPCFLAG3 DS
X                                                     02-CEECA
000220                              568+CEECAACIO   DS    A                Address oc
cio                      02-CEECA
000224                              569+CEECAAFDSETFD DS  F                Used by FD_*
macros                 02-CEECA
000228                              570+CEECAAFCBMUTEXOK DS
XL2                                                 02-CEECA
00022A                              571+            DS
XL2                                                   02-CEECA
00022C                              572+CEECAATC16  DS
F                                                     02-CEECA
000230                              573+CEECAATC17  DS
F                                                     02-CEECA
000234                              574+CEECAAEDCOV DS    A                C/370 Open
Libvec                   02-CEECA
000238                              575+CEECAACTOFSV DS
F                                                     02-CEECA
00023C                              576+CEECAATRTSPACE DS A                C/370 Open
Libvec                   02-CEECA
000240                              577+            DS
XL24                                                  02-CEECA
000258                              578+CEECAA_TCASRV DS  0CL36            TCA Service Rtn
Vector             02-CEECA
000258                              579+CEECAA_TCASRV_USERWORD  DS
A                                             02-CEECA
00025C                              580+CEECAA_TCASRV_WORKAREA  DS
A                                             02-CEECA
000260                              581+CEECAA_TCASRV_GETMAIN   DS
A                                             02-CEECA
000264                              582+CEECAA_TCASRV_FREEMAIN  DS
A                                             02-CEECA
000268                              583+CEECAA_TCASRV_LOAD      DS
A                                             02-CEECA
00026C                              584+CEECAA_TCASRV_DELETE    DS
A                                             02-CEECA
000270                              585+CEECAA_TCASRV_EXCEPTION DS
A                                             02-CEECA
000274                              586+CEECAA_TCASRV_ATTENTION DS
A                                             02-CEECA
000278                              587+CEECAA_TCASRV_MESSAGE   DS
A                                             02-CEECA
00027C                              588+            DS    XL4
Reserved                            02-CEECA
000280                              589+CEECAALWS   DS    A                Addr of PL/I
Language Working Space 02-CEECA
000284                              590+CEECAASAVR  DS    A                Register
save            @CM0419A   02-CEECA
000288                              591+            DS    XL36
Reserved                      @P6C 02-CEECA

592+*

0002AC                              593+CEECAASYSTM DS    X                Underlying
Operating System@MF0072A 02-CEECA
```

```
                     00000         594+CEECAASYUND EQU   X'00'
undefined               @MF0072A 02-CEECA
                     00001         595+CEECAASYUNS EQU   X'01'
unsupported             @MF0072A 02-CEECA
                     00002         596+CEECAASYVM  EQU   X'02'
VM                      @MF0072A 02-CEECA
                     00003         597+CEECAASYMVS EQU   X'03'
MVS                     @MF0072A 02-CEECA
0002AD                             598+CEECAAHRDWR DS    X                 Underlying
Hardware        @MF0072A 02-CEECA
                     00000         599+CEECAAHWUND EQU   X'00'
undefined               @MF0072A 02-CEECA
                     00001         600+CEECAAHWUNS EQU   X'01'
unsupported             @MF0072A 02-CEECA
                     00002         601+CEECAAHW370 EQU   X'02'               System / 370 non-
XA      @MF0072A 02-CEECA
                     00003         602+CEECAAHWXA  EQU   X'03'               System / 370
XA          @MF0072A 02-CEECA
                     00004         603+CEECAAHWESA EQU   X'04'               System / 370
ESA         @MF0072A 02-CEECA
0002AE                             604+CEECAASBSYS DS    X                 Underlying
Subsystem       @MF0072A 02-CEECA
                     00000         605+CEECAASSUND EQU   X'00'
undefined               @MF0072A 02-CEECA
                     00001         606+CEECAASSUNS EQU   X'01'
unsupported             @MF0072A 02-CEECA
                     00002         607+CEECAASSNON EQU   X'02'               no
subsystem               @MF0072A 02-CEECA
                     00003         608+CEECAASSTSO EQU   X'03'
TSO                     @MF0072A 02-CEECA
                     00005         609+CEECAASSCIC EQU   X'05'
CICS                    @MF0072A 02-CEECA
0002AF                             610+CEECAAFLAG2 DS
X                                         @MF0072A 02-CEECA
                     00080         611+CEECAABIMODAL EQU X'80'               Bimodal
addressing         @MF0072A 02-CEECA
                     00040         612+CEECAA_VECTOR EQU X'40'               vector hardware
available  @MF0072A 02-CEECA
                     00020         613+CEECAATIP     EQU X'20'               Thread termination
in progress      02-CEECA
                     00010         614+CEECAA_THREAD_INITIAL EQU X'10'       if on, indicates
this is the IPT    02-CEECA
                     00008         615+CEECAA_TRACE_ACTIVE   EQU X'08'       If on, library
trace is active      02-CEECA
                                   616+*                                      (TRACE runtime
option was set)
                     00004         617+CEECAA_ALTSTK_ACTIVE  EQU X'04'       If on, alt stack
active    @KCG0034 02-CEECA
                     00002         618+CEECAA_ENQ_WAIT_INTERRUPTABLE EQU X'02' PL/I doing
Exclusive   KN80230  02-CEECA
                                   619+*                                          file in
Multitasking   KN80230
                     00001         620+CEECAA_USRSTK_ACTIVE  EQU X'01'       If on, context
switching user stack 02-CEECA
                                   621+*                                          is
active              PQ04250
```

```
622+*

0002B0                             623+CEECAALEVEL DS    X                 LE/370 level
identifier             02-CEECA
                     0001C         624+CEL_LEVEL_IDENTIFIER EQU
X'1C'                                    @E8C 02-CEECA
0002B1                             625+CEECAA_PM   DS    X                 PROGRAM
MASK               @NX0166C 02-CEECA
0002B2                             626+CEECAA_INVAR DS   XL2               At same offset in
31 & 64 mode @G3C 02-CEECA
0002B4                             627+CEECAAGETLS DS    A                 ADDR OF LE/370
LIBRARY STACK MGR    02-CEECA
0002B8                             628+CEECAACELV  DS    A                 Addr of LE/370
LIBVEC               02-CEECA
0002BC                             629+CEECAAGETS  DS    A                 Addr of LE/370 get
stack stg rtn    02-CEECA
0002C0                             630+CEECAALBOS  DS    A                 Start of library
stack stg seg      02-CEECA
0002C4                             631+CEECAALEOS  DS    A                 End of library
stack stg seg        02-CEECA
0002C8                             632+CEECAALNAB  DS    A                 Next available byte
of lib stg      02-CEECA
0002CC                             633+CEECAADMC   DS    A                 Addr of ESPIE Devil-
```

```
May-Care rtn    02-CEECA
0002D0                          634+CEECAAABCODE DS   0F                Most recent ABEND
completion CDE    02-CEECA
0002D0                          635+CEECAACD    DS    XL4               Most recent CAASHAB
abend code      02-CEECA
0002D4                          636+CEECAARSNCODE DS  0F                Most recent ABEDN
reason Code       02-CEECA
0002D4                          637+CEECAARS    DS    XL4               Most recent CAASHAB
reason code     02-CEECA
0002D8                          638+CEECAAERR   DS    A                 Addr of the current
CIB             02-CEECA
0002DC                          639+CEECAAGETSX DS    A                 Addr of LE/370
stack stg extender   02-CEECA
0002E0                          640+CEECAADDSA  DS    A                 Addr of the dummy
DSA               02-CEECA
0002E4                          641+CEECAASECTSIZ DS  F                 Vector Section
Size                  02-CEECA
0002E8                          642+CEECAAPARTSUM DS  F                 Vector Partial Sum
Number           02-CEECA
0002EC                          643+CEECAASSEXPNT DS  F                 Log of Vector
Section Size          02-CEECA
0002F0                          644+CEECAAEDB   DS    A                 address of the
EDB                   02-CEECA
0002F4                          645+CEECAAPCB   DS    A                 address of the
PCB                   02-CEECA
                                646+*
-------------------------------------------------------------------
                                647+* - THE FOLLOWING TWO FIELDS ARE USED FOR VALIDATION OF
THE CAA.      -
                                648+*
-------------------------------------------------------------------
0002F8                          649+CEECAAEYEPTR DS   A                 addr of CAA
eyecatcher              02-CEECA
0002FC                          650+CEECAAPTR   DS    A                 addr of this
CAA                    02-CEECA
000300                          651+CEECAAGETS1 DS    A                 DSA alloc - R13 not
DSA addr        02-CEECA
000304                          652+CEECAASHAB  DS    A                 ABEND shunt routine
address         02-CEECA
000308                          653+CEECAAPRGCK DS    A                 pgm interrupt code
for CAADMC       02-CEECA
00030C                          654+CEECAAFLAG1 DS    X                 CAA Flag
1                        02-CEECA
                     00080      655+CEECAASORT  EQU   B'10000000'       Call to DFSORT is
active            02-CEECA
                     00040      656+CEECAA_USE_OLD_STK  EQU  B'01000000'  use old
stack               @P5A  02-CEECA
                     00020      657+CEECAA_CICS_EXT_REG EQU  B'00100000'  CICS extended
regs active   @DQA  02-CEECA
                     00010      658+CEECAASHAB_RECOVER_IN_ESTAE_MODE EQU
B'00010000'                @DYA   02-CEECA
                                659+*                                   LE ESTAE should set
up to    @DYA
                                660+*                                   retry at the
CEECAASHAB      @DYA
                                661+*                                   address in the same
mode and @DYA
                                662+*                                   key as when the LE
ESTAE was @DYA
                                663+*
established.                @DYA
                     00008      664+CEECAASHAB_IGNORED              EQU
B'00001000'                @DYA   02-CEECA
                                665+*                                   Set when CEECAASHAB
ignored  @DYA
                     00004      666+CEECAA_FETCH_RELES_IN_PROGRESS   EQU
B'00000100'                @E1A   02-CEECA
                     00002      667+CEECAA_CICS_VR_SPT EQU  B'00000010'  CICS vector regs
active     @E9A   02-CEECA
00030D                          668+CEECAASHAB_KEY    DS  X              IPK result when
abend shunt  @DYA   02-CEECA
                                669+*                                   routine is
established       @DYA
00030E                          670+            DS    CL2
reserved                    @DYC   02-CEECA
000310                          671+CEECAAURC   DS    F                 Thread level return
code.           02-CEECA
000314                          672+CEECAARSRV1 DS
4A                                                    02-CEECA
                                673+*
-------------------------------------------------------------------
                                674+* - THE FOLLOWING FIELD CONTAINS THE PRE-INIT
```

```
COMPATABILITY           -
                                675+* - CONTROL BLOCK
ADDRESS.                                        -
                                676+*
-------------------------------------------------------------------------

000324                          677+CEECAAPICICB DS   A                 Addr of pre-init
compatability cb   02-CEECA
000328                          678+CEECAARSRV2  DS
A                                                  02-CEECA
                                679+*
-------------------------------------------------------------------------
00032C                          680+CEECAAGOSMR  DS   H                 Go Some More. used
TRAV multiple    02-CEECA
00032E                          681+             DS   H
skip.                           02-CEECA
                                682+*
-------------------------------------------------------------------------
000330                          683+CEECAALEOV   DS   A                 Addr of LE/OpenMVS
LIBVEC           02-CEECA
000334                          684+CEECAA_SIGSCTR        DS  F         SIGSAFE
Counter            @C54544  02-CEECA

685+*

000338                          686+CEECAA_SIGSFLG        DS  XL4       SIGSAFE
Flags              @C54544  02-CEECA

687+*

                                688+*                     First
byte                               @P4A

689+*

                     00080      690+CEECAA_SIGPUTBACK     EQU X'80'     Signal
Putback                    02-CEECA
                     00040      691+CEECAA_SA_RESTART     EQU X'40'     SA_Restart
processing needed        02-CEECA
                                692+*                     EQU X'20'
<unused>
                     00010      693+CEECAA_SIGSAFE        EQU X'10'     Indicates that
synchronous          02-CEECA
                                694+*                                    signals are safe
to be delivered
                                695+*                                    regardless of
where the interrupt
                                696+*
occurred
                     00008      697+CEECAA_CANCELSAFE     EQU X'08'     Indicates that
synchronous @CM0565A 02-CEECA
                                698+*                                    cancels are safe
to be    @CM0565A
                                699+*                                    delivered
regardless of   @CM0565A
                                700+*                                    where the
interrupt       @CM0565A
                                701+*
occurred                @CM0565A
                     00004      702+CEECAA_SIGRESYNCH     EQU X'04'     One or more
synchronous signals     02-CEECA
                                703+*                                    may have been
recently put back
                                704+*                                    last time a signal
was resolicited
                                705+*                                    when returning to
non-XPLINK
                                706+*                                    user
code
                     00002      707+CEECAA_FRZ_UNSAFE     EQU X'02'     This thread is in
an unsafe state   02-CEECA
                                708+*                                    to be frozen (set
by members)
                     00001      709+CEECAA_NOAPPREGS      EQU X'01'     User Application
registers may      02-CEECA
                                710+*                                    be saved in a
nonstandard place

711+*                                                            @DFA
                                712+*                     Second
byte                              @P4A
```

```
713+*
                       00080         714+CEECAA_EINTR_RSOL     EQU   X'80'   Secondary signal
resolicit          02-CEECA
                                     715+*                                   in progress after
EINTR
                                     716+*                                   from inner
function            @P4A
                       00040         717+CEECAA_EINTR_PUTB     EQU   X'40'   Secondary
resolicited signal        02-CEECA
                                     718+*                                   has been put
back              @P4A
                           00020     719+CEECAA_EINTR_REST     EQU   X'20'   User catcher
returned after         02-CEECA
                                     720+*                                   catching
secondary
                                     721+*                                   resolicited signal
with
                                     722+*                                   SA_RESTART in
effect           @P4A
                           00010     723+CEECAA_EINTR_SIGG     EQU   X'10'   "Stray" signal
interrupted          02-CEECA
                                     724+*                                   CEEOSIGG while
secondary
                                     725+*                                   signal
resolicitation
                                     726+*                                   was in
progress                @P4A
                                     727+*                     EQU   X'08'
Reserved                       @P4A
                                     728+*                     EQU   X'04'
Reserved                       @P4A
                                     729+*                     EQU   X'02'
Reserved                       @P4A
                                     730+*                     EQU   X'01'
Reserved                       @P4A

731+*
```

```
                                     732+*                     (16 unused
bits)                              @P4C

733+*

00033C                               734+CEECAATHDID           DS  CL8       Posix thread
id                       02-CEECA
000344                               735+CEECAA_DCRENT         DS  A         CRENT anchor for
DCE                   02-CEECA
000348                               736+CEECAA_DANCHOR        DS  A         DCE anchor per
thread                  02-CEECA
00034C                               737+CEECAA_CTOC           DS  A         TOC anchor for
CRENT                   02-CEECA
000350                               738+CEECAARCB             DS  A
A(RCB)                               02-CEECA
000354                               739+CEECAACICSRSN         DS  A         CICS reason code
from member           02-CEECA
                                     740+*
language
000358                               741+CEECAAMEMBR           DS  A         Addr of thread-
level member list    02-CEECA
00035C                               742+CEECAA_SIGNAL_STATUS  DS  A         Signal stat for
terminating thd     02-CEECA
000360                               743+CEECAA_HCOM_REG7      DS 0A         Saved Reg7  from
HCOM                  02-CEECA
000360                               744+CEECAA_HCOM_REG14     DS  A         Saved Reg14 from
HCOM   @CH0092A    02-CEECA
000364                               745+CEECAA_STACKFLOOR     DS  A         Lowest usable addr
in HP stack      02-CEECA
000368                               746+CEECAAHPGETS          DS  A         HP stack extension
rtn              02-CEECA
00036C                               747+CEECAAEDCHPXV         DS  A         C/C++ XPLINK
libvec                   02-CEECA
000370                               748+CEECAAFOR1            DS  A         Reserved for
FORTRAN                  02-CEECA
000374                               749+CEECAAFOR2            DS  A         Reserved for
FORTRAN                  02-CEECA
000378                               750+CEECAATHREADHEAPID    DS  A         Thread
heapid         @NX0093A      02-CEECA
00037C                               751+CEECAA_SYS_RTNCODE    DS  F         Sys (kernel) return
code @CM1752    02-CEECA
```

```
000380                                752+CEECAA_SYS_RSNCODE    DS  F         Sys (kernel) reason
code @CM1752    02-CEECA
000384                                753+CEECAAGETFN           DS  A         Address of WSA swap
routine         02-CEECA
000388                                754+CEECAA_LER4           DS  CL8       Reserved for
expansion LE 1.4       02-CEECA

755+*****************************************************************

                                      756+*    LE V1R5M0 EXTERNAL CONTROL BLOCK
SECTION                      *
                                      757+*      - any external control block fields should be
added         *
                                      758+*        in this section.  Extra reserved space was
added          *
                                      759+*        with the intention of at end of project the
excess        *
                                      760+*        reserved space will be removed and all
external fields    *
                                      761+*        will be
shifted.                               *

762+*****************************************************************

000390                                763+CEECAASIGNGPTR        DS  A         Pointer to 'signam'
external        02-CEECA
                                      764+*                                  variable in a C
application
000394                                765+CEECAASIGNG           DS  F         value of sign of
lgamma()           02-CEECA
                                      766+*                                   -1 - negative
sign
                                      767+*                                    0 -
zero
                                      768+*                                   +1 - positive
sign
000398                                769+CEECAA_FORDBG         DS  A         Ptr to AFHDBHIM
-        @N80095A   02-CEECA
                                      770+*                                  FORTRAN hook
interface   @N80095A
00039C                                771+CEECAAAB_STATUS       DS  X         validity
flags           KN80120    02-CEECA
                   00080              772+CEECAAAB_GR0_VALID    EQU X'80'     CEECAAAB_GR0 is
valid    KN80120    02-CEECA
                   00040              773+CEECAAAB_ICD1_VALID   EQU X'40'     CEECAAAB_ICD1 is
valid   KN80120    02-CEECA
                   00020              774+CEECAAAB_ABCC_VALID   EQU X'20'     CEECAAAB_ABCC is
valid   KN80120    02-CEECA
                   00010              775+CEECAAAB_CRC_VALID    EQU X'10'     CEECAAAB_CRC is
valid    KN80120    02-CEECA
                   00008              776+CEECAAAB_GR15_VALID   EQU X'08'     CEECAAAB_GR15 is
valid   KN80120    02-CEECA
00039D                                777+CEECAA_STACKDIRECTION DS  X         Stack
Direction                     02-CEECA
                      00000           778+CEECAASTACK_UP        EQU X'00'
Up                                    02-CEECA
                      00001           779+CEECAASTACK_DOWN      EQU X'01'     Down
(XPLINK)                       02-CEECA
00039E                                780+                      DS  XL2
RESERVED                KN80120    02-CEECA
0003A0                                781+CEECAAAB_GR0          DS  A         Reg
0                    KN80120    02-CEECA
0003A4                                782+CEECAAAB_ICD1         DS  A
SDWAICD1                KN80120    02-CEECA
0003A8                                783+CEECAAAB_ABCC         DS  A
SDWAABCC                KN80120    02-CEECA
0003AC                                784+CEECAAAB_CRC          DS  A
SDWACRC                 KN80120    02-CEECA
                    002D4             785+CEECAAAB_GR15         EQU CEECAARS  reg
15                  KN80120    02-CEECA
0003B0                                786+CEECAAAGTS            DS  A         C compiler stk
inc       KN00102    02-CEECA
```

```
0003B4                                787+CEECAA_LER5N1         DS  CL4       reserved for
expansion   KCG0088    02-CEECA
0003B8                                788+CEECAAHERP            DS  A
A(CEEHERP)              KCG0061    02-CEECA
0003BC                                789+CEECAAUSTKBOS         DS  A         Start of user stack
seg  PQ04250    02-CEECA
0003C0                                790+CEECAAUSTKEOS         DS  A         End of user stack
seg    PQ04250    02-CEECA
```

```
0003C4                                  791+CEECAAUSERRTN@        DS  A         A(UserRtn) for
pthread     @01A     02-CEECA
0003C8                                  792+CEECAAUDHOOK          DS  XL8       hook swapping
xplink       @DBA     02-CEECA
                                        793+*                     DS  BL8
reserved                   @DBA
                                        794+*CEECAAUDHOOKSW        DS  BL4       hook
switch                 @DBA
                                        795+*                      DS  BL4
reserved                   @DBA
                                        796+*                      DS  XL6
reserved                   @DBA
0003D0                                  797+CEECAACEL_HPXV_B      DS  A         Address of XPLink
compat   @DCC     02-CEECA
                                        798+*                      vector for Base
library   @DCA
0003D4                                  799+CEECAACEL_HPXV_M      DS  A         Address of XPLink
compat   @DCA     02-CEECA
                                        800+*                      vector for Math
library   @DCA
0003D8                                  801+CEECAACEL_HPXV_L      DS  A         Address of
XPLink          @DCC     02-CEECA
                                        802+*                      compat vector
for         @DCA
                                        803+*                      Locale
library            @DCA
0003DC                                  804+CEECAACEL_HPXV_O      DS  A         Address of
XPLink          @DDA     02-CEECA
                                        805+*                      compat vector
for         @DDA
                                        806+*                      Open
library              @DDA
0003E0                                  807+CEECAACEL4VEC3        DS  A         Address of Vec3
LibVec     @DHA     02-CEECA
0003E4                                  808+CEECAA_CEEDLLF          DS A        Addr of newest
CEEDLLF.  Not   @P6M 02-CEECA
                                        809+*                                  same offset as in
64bit.       @P6M
0003E8                                  810+CEECAA_SAVSTACK       DS  A         Saved Stack pointer
used for   @DPA 02-CEECA
                                        811+*                                    OS_NOSTACK
linkage           @DPA
0003EC                                  812+                      DS  XL4
Reserved                        @DSC 02-CEECA
0003F0                                  813+CEECAA_USER_WORD      DS  F         CAA user
word                  @DVC 02-CEECA
0003F4                                  814+CEECAA_SAVSTACK_ASYNC DS  A         Zero or address of
4-byte field@DPA 02-CEECA
                                        815+*                                  where the stack
pointer can be @DPA
                                        816+*                                  saved. When the
stack pointer  @DPA
                                        817+*                                  is saved here,
asynchronous    @DPA
                                        818+*                                  signals will be
accepted.      @DPA
0003F8                                  819+                      DS  A         Reserved for
COBOL             @DZA 02-CEECA
0003FC                                  820+CEECAA_STACK_GUARD    DS  XL4       Stack Guard
token              @EAC 02-CEECA
                      00094             821+CEECELVGTUN EQU  148                Offset to Get user
nab service.     02-CEECA
0000F8                00000 0020C       822+TESTLE   CSECT                     Restore main
CSECT              @PAA 01-DFHEI
0000F8                                  823+SKIPLE0005    DS
0H                                                  @PAA 01-DFHEI
                      0000D             824+DFHEIPLR EQU   13               Working storage
register           @D3A 01-DFHEI
                  R:D  00000            825+         USING DFHEISTG,13      Addressability for
working storage @D3A 01-DFHEI
0000F8 D207 D0DC 1000 000DC 00000       826+         MVC
DFHEIBP(L'DFHEIBP+L'DFHEICAP),0(1)                     @D3AX01-DFHEI
                                           +                                      COPY EIB AND CA
PTRS       @D3A

827+*************************************************************************

                                        828+*              ESTABLISH EIB
ADDRESSIBILITY                               *

829+*************************************************************************
```

```
0000FE 58B0 D0DC            000DC   830+         L
DFHEIBR,DFHEIBP                                 @BBAC81A 01-DFHEI
                  R:B  00000        831+         USING
DFHEIBLK,DFHEIBR                                @BBAC81A 01-DFHEI

832+*************************************************************************

                                    833+*              END OF PROLOG CODE FOR EXEC
INTERFACE                 *

834+*************************************************************************
```

```
000102 D227 D200 3188 00200 00188   837          MVC
OUTAREA(40),MSG1
000108 D203 D200 B010 00200 00010   838          MVC
OUTAREA(4),EIBTRMID
                                    839 *        EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(43)
FREEKB ERASE
                                    840          DFHECALL =X'180660001200C20000082204000020',,
(______RF,OUTAREA*
                                                       ),
(FB_2,=Y(43))

842+*************************************************************************
00010E                              843+         DS
0H                                                     01-DFHEC
00010E 4110 D0E8            000E8
844+         LA    1,DFHEIPL                                          01-DFHEC
000112 41E0 31E4            001E4
845+         LA    14,=X'180660001200C20000082204000020'              01-DFHEC
000116 1BFF                         846+
SR    15,15                                          01-DFHEC
000118 4100 D200            00200
847+         LA    0,OUTAREA                                          01-DFHEC
00011C 90E0 1000            00000
848+         STM   14,0,0(1)                                          01-DFHEC
000120 41E0 31E0            001E0
849+         LA    14,=Y(43)                                          01-DFHEC
000124 50E0 100C            0000C
850+         ST    14,12(,1)                                          01-DFHEC
000128 9680 100C               0000C
851+         OI    12(1),X'80'         LAST ARGUMENT                  01-DFHEC
00012C 58F0 31D8            001D8
852+         L     15,=V(DFHEI1)                                      01-DFHEC
000130 0DEF                         853+
BASR  14,15               INVOKE EXEC INTERFACE           @P7C 01-DFHEC

854+*************************************************************************
                                    855 *
EXEC CICS RECEIVE
                                    856          DFHECALL
=X'040200001200000014000040000000'

858+*************************************************************************
000132                              859+         DS
0H                                                     01-DFHEC
000132 4110 D0E8            000E8
860+         LA    1,DFHEIPL                                          01-DFHEC
000136 41E0 31F3            001F3
861+         LA    14,=X'040200001200000014000040000000'              01-DFHEC
00013A 50E0 1000            00000
862+         ST    14,0(,1)                                           01-DFHEC
00013E 9680 1000               00000
863+         OI    0(1),X'80'          LAST ARGUMENT                  01-DFHEC
000142 58F0 31D8            001D8
864+         L     15,=V(DFHEI1)                                      01-DFHEC
000146 0DEF                         865+
BASR  14,15               INVOKE EXEC INTERFACE           @P7C 01-DFHEC

866+*************************************************************************
000148 D20C D200 31B0 00200 001B0
867          MVC  OUTAREA(13),MSG2
                                    868 *
EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(13) FREEKB ERASE
                                    869          DFHECALL
=X'180660001200C20000082204000020',,(______RF,OUTAREA*
                                                       ),
(FB_2,=Y(13))

871+*************************************************************************
00014E                              872+         DS
```

```
0H                                                      01-DFHEC
00014E 4110 D0E8           000E8
873+         LA    1,DFHEIPL                                      01-DFHEC
000152 41E0 31E4           001E4
874+         LA    14,=X'180660001200C20000082204000020'          01-DFHEC
000156 1BFF                      875+
SR    15,15                                    01-DFHEC
000158 4100 D200           00200
876+         LA    0,OUTAREA                                      01-DFHEC
00015C 90E0 1000           00000
877+         STM   14,0,0(1)                                      01-DFHEC
000160 41E0 31E2           001E2
878+         LA    14,=Y(13)                                      01-DFHEC
000164 50E0 100C           0000C
879+         ST    14,12(,1)                                      01-DFHEC
000168 9680 100C              0000C
880+         OI    12(1),X'80'         LAST ARGUMENT              01-DFHEC
00016C 58F0 31D8           001D8
881+         L     15,=V(DFHEI1)                                  01-DFHEC
000170 0DEF                      882+
BASR  14,15              INVOKE EXEC INTERFACE           @P7C 01-DFHEC

883+*****************************************************************************
                                 884 *
EXEC CICS RETURN
                                 885          DFHECALL
=X'0E0800001200001000'

887+*****************************************************************************
000172                           888+         DS
0H                                                      01-DFHEC
000172 4110 D0E8           000E8
889+         LA    1,DFHEIPL                                      01-DFHEC
000176 41E0 3202           00202
890+         LA    14,=X'0E0800001200001000'                      01-DFHEC
00017A 50E0 1000           00000
891+         ST    14,0(,1)                                       01-DFHEC
00017E 9680 1000              00000
892+         OI    0(1),X'80'          LAST ARGUMENT              01-DFHEC
000182 58F0 31D8           001D8
893+         L     15,=V(DFHEI1)                                  01-DFHEC
```

```
000186 0DEF                      894+         BASR  14,15               INVOKE EXEC
INTERFACE           @P7C 01-DFHEC

895+*****************************************************************************

                                 896
*
000188 A7A7A7A77A40C1E2          897 MSG1     DC   C'xxxx: ASM program invoked. ENTER TO
END.'
0001B0 D7D9D6C7D9C1D440          898 MSG2     DC   C'PROGRAM
ENDED'
                                 899
DFHREGS
                     00000       900+R0       EQU
0                                             @BBDA00X 01-DFHRE
                     00001       901+R1       EQU
1                                             @BBDA00X 01-DFHRE
                     00002       902+R2       EQU
2                                             @BBDA00X 01-DFHRE
                     00003       903+R3       EQU
3                                             @BBDA00X 01-DFHRE
                     00004       904+R4       EQU
4                                             @BBDA00X 01-DFHRE
                     00005       905+R5       EQU
5                                             @BBDA00X 01-DFHRE
                     00006       906+R6       EQU
6                                             @BBDA00X 01-DFHRE
                     00007       907+R7       EQU
7                                             @BBDA00X 01-DFHRE
                     00008       908+R8       EQU
8                                             @BBDA00X 01-DFHRE
                     00009       909+R9       EQU
9                                             @BBDA00X 01-DFHRE
                     0000A       910+R10      EQU
10                                            @BBDA00X 01-DFHRE
                     0000B       911+R11      EQU
11                                            @BBDA00X 01-DFHRE
                     0000C       912+R12      EQU
12                                            @BBDA00X 01-DFHRE
```

```
                      0000D         913+R13      EQU
13                                              @BBDA00X 01-DFHRE
                      0000E         914+R14      EQU
14                                              @BBDA00X 01-DFHRE
                      0000F         915+R15      EQU
15                                              @BBDA00X 01-DFHRE
                      0000A         916+RA       EQU
R10                                             @BBDA00X 01-DFHRE
                      0000B         917+RB       EQU
R11                                             @BBDA00X 01-DFHRE
                      0000C         918+RC       EQU
R12                                             @BBDA00X 01-DFHRE
                      0000D         919+RD       EQU
R13                                             @BBDA00X 01-DFHRE
                      0000E         920+RE       EQU
R14                                             @BBDA00X 01-DFHRE
                      0000F         921+RF       EQU
R15                                             @BBDA00X 01-DFHRE
                      00003         922 CODEREG  EQU
R3
                                    923          DFHEIRET                  INSERTED BY
TRANSLATOR

925+*****************************************************************************

                                    926+*              EPILOG CODE FOR EXEC
INTERFACE                          *

927+*****************************************************************************

0001BE                              928+         DS
0H                                              @BBAC81A 01-DFHEI
0001BE 4110 31C8           001C8    930+         LA   1,CEET0025               Get address of
termination list 02-CEETE
0001C2 58F0 31DC           001DC    931+         L    15,=V(CEETREC)           Get address of
termination rtn  02-CEETE
0001C6 05EF                         932+         BALR 14,15                    Call
termination routine.       02-CEETE
0001C8 000001D0                     933+CEET0025 DC   A(*+8)                Parm
1                               02-CEETE
0001CC 800001D4                     934+         DC   A(*+8+X'80000000')   Parm
2                               02-CEETE
0001D0 00000000                     935+         DC   A(0)
Enc_Modifier                     02-CEETE
0001D4 00000000                     936+         DC   A(0)                 Return
code.                         02-CEETE
000210                 00210 00008  937+CEEMAIN
CSECT                                                     02-CEETE
                                    938+CEEMAIN  RMODE
ANY                                                  02-CEETE
                                    939+CEEMAIN  AMODE
ANY                                                  02-CEETE
000210 00000000                     940+         DC
A(TESTLE)                                  @04A  02-CEETE
000214 00000000                     941+         DC
F'0'                                                 02-CEETE
0001D8                 00000 0020C  942+TESTLE
CSECT                                                    02-CEETE

943+*****************************************************************************

                                    944+*              END OF EPILOG CODE FOR EXEC
INTERFACE                  *

945+*****************************************************************************

0001D8                              946+
LTORG ,                                          @BBAC81A 01-DFHEI
0001D8 00000000                     947
=V(DFHEI1)
0001DC 00000000                     948
=V(CEETREC)
0001E0 002B                         949
=Y(43)
0001E2 000D                         950
=Y(13)
```

```
0001E4 180660001200C200             951
=X'180660001200C20000082204000020'
0001F3 0402000012000000             952
=X'040200001200000014000040000000'
```

```
000202 0E08000012000010             953
=X'0E0800001200001000'
00020C                              954+         DS
0H                                            @F8E1S @L1C 01-DFHEI

               Page   20
  Active Usings: TESTLE,R3  DFHEIBLK,R11
DFHEISTG,R13
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement
HLASM R6.0  2020/02/19 14.08
                                    957          DFHEISTG                  INSERTED BY
TRANSLATOR
                                    958          DFHEIEND                  INSERTED BY
TRANSLATOR
                                    959+*              TERMINATE DEFINITION OF DYNAMIC
STORAGE                *
0002C8                00000 002C8   960+DFHEISTG
DSECT                                                @BBAC81A 01-DFHEI
0002C8                002C8 002C8   961+
ORG                                                          01-DFHEI
0002C8                              962+DFHEIEND DS    0X                  END OF DYNAMIC
STORAGE      @BBAC81A 01-DFHEI
                                    963
END
```

Using the EXEC interface modules for AMODE(64) applications

For non- Language Environment AMODE(64) assembler language programs, the CALL statements that the language translator generates invoke EXEC interface modules that provide communication between your code and the CICS EXEC interface program, DFHEIG.

A language translator reads your source program and creates a new one. Normal language statements remain unchanged, but CICS commands are translated into CALL statements of the form required by the language in which you are coding. The calls invoke CICS-provided "EXEC" interface modules or *stubs* ; that is, function-dependent sections of code used by the CICS high-level programming interface. These stubs, which are provided in the SDFHLOAD library, must be link-edited with your application program. These stubs are invoked during execution of EXEC CICS commands.

The following stub programs are provided to transfer control from an AMODE(64) application program to CICS . These stubs use CICS internal control blocks to locate the required CICS code.

- DFHEAG0 (prolog and epilog stub).

  When the DFHEIENT macro calls this stub, it transfers control to the AMODE(64) PROLOG program.

  When the DFHEIRET macro calls this stub, it transfers control to the AMODE(64) EPILOG program.
- DFHEAG (command stub).

  When the DFHECALL macro calls this stub, it transfers control to the AMODE(64) initial command processor. DFHEG1 is an alias of DFHEAG and the DFHECALL macro actually calls the alias, DFHEG1.

You must link-edit the DFHEAG and DFHEAG0 stubs with the AMODE(64) application program. Use the following binder statements:

```
ORDER DFHEAG
INCLUDE SYSLIB(DFHEAG)
ENTRY
program_name
NAME
program_name
(R)
```

program_name is the name of the AMODE(64) application.

This is similar to the way that stubs DFHEAI and DFHEAI0 are link-edited with AMODE(24) and AMODE(31) assembler language application programs.

Uppercase translation

You can customize the translation of lower- and mixed-case characters and national characters that are input at a terminal. You can also translate operator messages produced by temporary storage data sharing.

Translating national characters to uppercase

In CICS, translation of terminal user-input to uppercase characters can be done either by using the UCTRAN option on the PROFILE and TYPETERM definitions, or by using the **EXEC CICS SET TERMINAL(termid) UCTRANST** command.

However, some languages have characters which are not part of the set of EBCDIC characters translated by UCTRAN, and so are never translated to uppercase, regardless of what you have specified on your resource definitions. To translate these national characters, you have two options:

- Use the XZCIN exit.

  XZCIN is described in Exit XZCIN . To use it for uppercase translation, you must supply your own translation routine, which is then invoked when terminal input occurs.
- Create a new terminal control table (TCT), based on your current TCT (or on the dummy TCT, DFHTCTDY, if you have TCT=NO specified in the SIT), and modify the translation table in it.

Whichever method you use, Character Data Representation Architecture is a useful reference for information on code pages.

Using DFHTCTxx

To translate national characters which are not handled by UCTRAN, you can modify the translation table in the terminal control table.

About this task

If you use RDO for all your terminals and have TCT=NO specified in your SIT or its overrides, CICS uses the dummy TCT, DFHTCTDY, to create control blocks for RDO-defined and autoinstalled terminals. It is not recommended that you modify DFHTCTDY directly. Instead, take a copy of the DFHTCTDY source file, save it under a new name, and modify the copy. The steps you need to follow are:

Note: If you are already using a customized TCT rather than DFHTCTDY (that is, something other than 'NO' or 'DY' is specified on the SIT TCT parameter), you must add your translation code to the TCT you are using.

Procedure

1. Take a copy of the DFHTCTDY assembler source file.

   CICS provides this in the CICSTS56.CICS.SDFHSAMP library.
2. Modify the translation table in the source file, as shown in Figure 140 on page 622.
3. Save the source file as DFHTCTxx, where 'xx' is any 2–character suffix other than 'DY'.
4. Use the CICS-supplied DFHAUPLE job to assemble, define to SMP/E, and linkedit the new table.
5. Specify the 2–character suffix of your new TCT on the SIT TCT parameter.
6. Restart CICS so that the new TCT takes effect.

Example

Figure 140 on page 622 shows a suggested way to code the assembler source statements to translate your national characters.

```
 MACRO
NATLANG
DFHUCTRT CSECT Resume UCTRAN table CSECT
.*
.* This example translates lowercase 'a' ( EBCDIC X'81') to
.* uppercase 'A' (EBCDIC X'C1') for a US code page.
.*
ORG TCZUCTAB+X'81' Reset the counter to the
character to be translated.
DC X'C1' Declare the replacement
character as a constant.
.*
.* Repeat the above two statements for each extra character you want
.* to be translated.
.*
ORG , Reset the location counter
&SYSLOC LOCTR Resume previous location counter
MEND End of macro definition
DFHTCT TYPE=INITIAL,SUFFIX=xx, *
MIGRATE=COMPLETE, *
ACCMETH=(VTAM), *
DUMMY=DUMMY
NATLANG Execute NATLANG
DFHTCT TYPE=FINAL
END DFHTCTBA
```

Figure 140. Suggested coding for national language character translation

Translating TS data sharing messages to uppercase

CICS temporary storage (TS) data sharing uses AXM services to write operator messages. These messages are in mixed-case English; table AXMMSTAB is used to remove unprintable characters. If necessary, you can modify AXMMSTAB to convert the messages to uppercase English.

The modules that use AXM message services are AXMSI, AXMSC, and DFHXQMN. AXMSI and AXMSC are both in a linklist library; DFHXQMN is in the CICS authorized library. To convert TS data sharing messages to uppercase, modify the copy of AXMMSTAB used by each of these modules by using SPZAP with the following input (for each module):

```
NAME modulename AXMMSTAB
VER 0081 818283848586878889
VER 0091 919293949596979899
VER 00A2 A2A3A4A5A6A7A8A9
REP 0081 C1C2C3C4C5C6C7C8C9
REP 0091 D1D2D3D4D5D6D7D8D9
REP 00A2 E2E3E4E5E6E7E8E9
```

Chapter 13. Setting up an application

Applications are one of the key capabilities of cloud-enabling CICS Transaction Server for z/OS. The large set of resources that makes up a business application within CICS® can be logically defined as a single entity and deployed on a platform as a single resource.

Before you begin

For an introduction to applications, see How it works: applications. For information about securing application resources, see Security for platforms and applications.

About this task

Most of the work in setting up an application is done in CICS Explorer. The CICS Cloud perspective provides the views to manage the lifecycle of applications.

The steps below outline the process of setting up an application. Each step contains a link to more detailed instructions.

Procedure

1. Design the application, considering what application entry points, policies, and resources that you want to deploy as part of the application.

   For more information, see Designing an application for cloud enablement.
2. In CICS Explorer, create CICS bundles to contain the application resources application entry points dependencies,and any CICS policies relating to the application.

   For more information, see Defining CICS bundles.
3. In CICS Explorer, create an application project to define the application bundle.

   The application bundle references the required CICS bundles. For more information, see Packaging CICS applications for deployment in a cloud environment.
4. In CICS Explorer, create an application binding project to describe the deployment rules for the application to a region type in a platform.
5. From CICS Explorer, export the application project, the application binding project, and the CICS bundles to zFS.

   The export process exports the files to the platform home directory in zFS in the applications, bindings, and bundles subdirectories. For more information, see Deploying an application to a platform.
6. In CICS Explorer, create an application definition.

   The application definition is aCICSPlex SM APPLDEF resource definition, which points to the application bundle and the application binding in the platform home directory for the platform where the application runs. For more information, see Deploying an application to a platform.
7. In CICS Explorer, install the application to a platform.

   CICSPlex SM uses the information in the application bundle and the application binding to install the CICS bundles that make up the application into all of the target CICS regions in the platform. In each region, the resources that are specified in the bundles are dynamically created and CICS checks that any resources that are specified as dependencies are present. For more information, see Deploying an application to a platform.
8. In CICS Explorer, make the application available for users to start through the available application entry points. For more information, see Managing applications.

Results

You have an application running on a platform.

© Copyright IBM Corp. 1974, 2023

What to do next

You can add further quality of service by deploying policies to control the environment. For more information, see CICS policies.

For more information about managing an application, see Administering platforms and applications.

Designing a CICS application for deployment to a platform

Before you create an application for deployment to a platform as part of a CICS cloud solution, identify how you plan to structure the application. For example, consider what resources to bundle with the application or declare as dependencies for it, what application entry points to provide for access to the application, where the application will be deployed, and whether you plan to run multiple versions of the application concurrently.

Identifying application entry points

There are some factors to consider when identifying entry points:

- The earlier in the transaction flow that you identify that a task is part of an application, the sooner you can apply policies and start monitoring for the cost of the application. This can be of particular importance if the application spans multiple CICS TS regions. To monitor the application across all of the CICS TS regions that it touches, the entry points should be defined on the CICS TS region that first accepts the inbound request.
- An application can have different functions that it performs. These are referred to as the operations of the application, for example, inquiring on a customer. There can be value in identifying these operations, and monitoring or applying policies to individual operations.

Which resources should be defined in CICS bundles?

Defining resources in a CICS bundle is an important step, because the management and lifecycle of those resources is delegated to the CICS bundles and management bundles that installed the resources. You no longer modify the resources or change their state individually, because they are automatically added, updated, or removed when you operate on the CICS bundles and the application. Application architects must therefore carefully consider which resources for an application should have their lifecycle tied to the lifecycle of a CICS bundle.

Review the information in Characteristics of bundled resources to help you choose which resources to define in CICS bundles and how to arrange them. If your application uses resources that cannot be defined in a CICS bundle, or that you do not want to define in a CICS bundle, you can continue to specify these as imports.

For supported resource types, a CICS resource is private if the resource is defined in a CICS bundle that is packaged and installed as part of an application. Because these resources are not available to any other application or version installed on the platform, or to other applications in the CICS region, the resource name does not have to be unique in your installation. Review the information in Private resources for application versions to understand the changes to behavior and management for private resources. If you do not want a resource of the supported resource types to be private, continue to specify it as an import.

Which bundles are installed where?

Which bundles are in the application and which bundles are added in the application binding, as an alternative to deploying them with the application or adding them to the platform? These might include policies or resource definitions that control or customize the behavior of the application for the target platform.

Packaging CICS applications for deployment in a cloud environment

For cloud enablement, you create applications. These applications package code and dependencies and resources for deployment to a platform. You create the application in CICS Explorer. Create CICS bundles for each application component and an application bundle that groups them together. Declare application entry points in these bundles and create an application binding..

Before you begin

For an introduction to applications, see How it works: applications. This task assumes that you already:

1. Decided how to structure your application. If not, see Designing an application for cloud enablement.
2. Set up a platform onto which the application can be deployed. If not, see Setting up a platform. Check that the Platform project for the target platform is present in your local workspace for CICS Explorer. Although you are not deploying the application at this stage, CICS Explorer requires the Platform project in order to validate the Application project and the Application Binding project.

About this task

You create an application in CICS Explorer. The following steps outline the procedure. For detailed steps, see Working with applications in the CICS Explorer product documentation.

Procedure

1. Create a CICS Bundle project for each application component.

   For each one, define appropriate resources in the CICS bundle and declaring any dependencies on other resources that are required by the component. A CICS bundle groups resources together and is version-controlled and managed as a single entity, so do not put resources that must be updated and managed separately in the same CICS bundle. For information about resources in CICS bundles, see Defining CICS bundles.
2. In your CICS Bundle projects, add application entry points to identify the resources that are access points to the application.

   For more information about application entry points, see Application entry points.
3. Create a CICS Application project that specifies the name and version of the CICS application.

   The CICS Application project defines the application bundle. Add references to the CICS bundles to include them in the application bundle.

   You can edit the application bundle after you created it to add or remove CICS bundles.
4. Create a CICS Application Binding project to define how each CICS bundle for the application is deployed to CICS region types in the target platform.

   You can edit the application binding after you created it to change how the CICS bundles are deployed.

   For information about application bindings, see Application binding.

What to do next

When you are ready to deploy the application on a platform, export the CICS Application project and the CICS Application Binding project to the platform home directory on zFS. This makes the application bundle, application binding, and associated CICS bundles available in the home directory of the target platform, ready to be installed in the CICS regions in the platform. You also create an application definition, which is a CICSPlex SM APPLDEF resource definition that points to the application bundle in the platform home directory. For information, see Deploying an application to a platform.

Invoking a multi-versioned application

Two or more versions of the same application can be installed, enabled and available on a platform at the same time. If multiple versions are available, callers can either access the highest available application version or use the **EXEC CICS INVOKE APPLICATION** command to invoke a specific or minimum version.

Before you begin

Before you can code your program, you should know:

- The name of the application.
- The name of the platform on which it is installed, or verify that it is installed on the current platform.
- The name of the operation that corresponds to one of the program entry points for the application to be invoked.
- The exact major version of the application you want to invoke.
- The exact or minimum minor version of the application you want to invoke.

About this task

Although two or more versions of the same application can be installed, enabled and available on a platform at the same time, only one of those versions is visible to the **EXEC CICS LINK** command. This version is the highest major and minor version of the application, and its entry points are public. So an **EXEC CICS LINK** to an entry point program always invokes the highest version of the application. The entry points for lower levels of an application are private, and so are not visible to **EXEC CICS LINK**.

Note: Micro version is always hidden as it reflects an internal change; for example, a bug fix. The caller always gets the latest micro version.

You use the **EXEC CICS INVOKE APPLICATION** to invoke an application through one of its program entry points, without having to know the name of the entry point program and regardless of whether its entry points are public. If no version is specified, then the highest major and minor version (the public level) is invoked. This is the same behavior as using an **EXEC CICS LINK** command to the application entry point. However, a lower version that is enabled and available can be invoked by using **EXEC CICS INVOKE APPLICATION** and specifying an appropriate major and minor version. You can specify that either an exact match on the application major version number and minor version number is required, or that a minor version number is the minimum that is required, but if a higher minor version is available it is to be used. If multiple higher minor versions are available the highest is used. The major version number cannot be exceeded and must match exactly.

For the full syntax of the **EXEC CICS INVOKE APPLICATION** command, see INVOKE APPLICATION.

Procedure

1. In your program, use the **EXEC CICS INVOKE APPLICATION** command and specify the name of the application you want to invoke.
2. Add the OPERATION and, if required, the PLATFORM option to the command.
   - The OPERATION option specifies the name of the application operation which the application entry point program implements.
   - The PLATFORM option specifies the name of the platform on which the application is installed. If no platform name is specified, the current platform name is used.
3. Optional: Add the MAJORVERSION and MINORVERSION options, and either the EXACTMATCH or MINIMUM keyword to the command.
   - The MAJORVERSION option specifies the major version number of the application as a fullword binary value. If MAJORVERSION is specified, then MINORVERSION must also be specified. If no version is specified, then the highest major and minor version of the application is invoked.

- The MINORVERSION option specifies the minor version number of the application as a fullword binary value.
- The EXACTMATCH keyword specifies that an exact match on the application major version number and minor version number is required.
- The MINIMUM keyword specifies that the specified minor version number is the minimum that is required, but to use a higher version if it is available. If multiple higher minor versions are available the highest is used. This applies to minor version numbers only. The major version number cannot be exceeded and must match exactly.

When you use either the EXACTMATCH or MINIMUM keyword, there is no match criteria for micro version. The highest micro version is always used.

4. Add the COMMAREA and LENGTH options to the command.
   - The COMMAREA option specifies a communication area that is to be made available to the called program. In this option, the data area is passed, and you must give it the name DFHCOMMAREA in the receiving program.
   - The LENGTH option specifies a halfword binary value that is the length in bytes of the COMMAREA. This value must not exceed 24 KB if the COMMAREA is to be passed between any two CICS servers.
5. Add the CHANNEL option to the command.

   The CHANNEL option specifies the 1 to 16 character name of a channel that is to be made available to the called entry point program. If the channel does not exist, it is created.

Results

When your program issues this command, CICS invokes the specified application at the appropriate entry point.

Examples of EXEC CICS INVOKE APPLICATION

Example 1

This example shows how to invoke an application that runs on the current platform, passing an exact match for the application major version number and minor version number:

```
EXEC CICS INVOKE APPLICATION(PAYROLL)
        OPERATION(APPLY_TAX_CHANGES)
        MAJORVERSION(2)
        MINORVERSION(3)
        EXACTMATCH
```

Where:

PAYROLL
Is the name of the application that is being invoked.

APPLY_TAX_CHANGES
Is the name of the application operation that is to be invoked.

2
Is the major version of application **PAYROLL** to be invoked.

3
Is the minor version of application **PAYROLL** to be invoked. This must match exactly.

When this code is run, the **APPLY_TAX_CHANGES** operation of version 2.3.x of application **PAYROLL** is invoked, where x is the highest microlevel.

Example 2

This example shows how to invoke an application that runs on a specified platform, and specifying a minimum minor version of the application:

```
EXEC CICS INVOKE APPLICATION(PENSIONS)
        OPERATION(UPDATE_PENSIONS)
        PLATFORM(TEST_PENSIONS_PLATFORM)
```

```
        MAJORVERSION(4)
        MINORVERSION(2)
        MINIMUM
```

Where:

PENSIONS
Is the name of the application that is being invoked.

UPDATE_PENSIONS
Is the name of the application operation that is to be invoked.

TEST_PENSIONS_PLATFORM
Is the name of the platform on which the application is installed.

4
Is the major version of application **PENSIONS** to be invoked.

2
Is the minimum minor version of application **PENSIONS** to be invoked. If higher minor versions exist, the highest minor version is invoked.

When this code is run, the **UPDATE_PENSIONS** operation of version 4.2 or higher of the **PENSIONS** application is invoked. If, for example, versions 4.2.4, 4.3.3, and 4.4.1 of **PENSIONS** were enabled and available, then the command would result in version 4.4.1 being invoked.

Chapter 14. Debugging applications

You can debug CICS applications using utilities that are supplied with CICS, or other tools available from IBM. The debug tools supplied with CICS include a workstation-based debugging tool, and the execution diagnostic facility (EDF).

You use these tools to step through the code and check the syntax of application programming and system programming commands. Additional IBM tools for debug include IBM Developer for z/OS and the IBM Debug Tool.

You can also collect transaction dumps and perform tracing for debug purposes. For more information, see CICS transaction dump.

Execution diagnostic facility (EDF)

You can use the execution diagnostic facility (EDF) to test an application program online, without modifying the program or the program-preparation procedure. The CICS execution diagnostic facility is supported by the CICS-supplied transaction, CEDF, which invokes the DFHEDFP program.

Note: You can also invoke CEDF indirectly through another CICS-supplied transaction, CEDX, which enables you to specify the name of the transaction you want to debug. When this section refers to the CEDF transaction (for example, when it explains about CICS starting a new CEDF task below) remember that it may have been invoked by the CEDX command.

The names of your programs should not begin with the letters "DFH" because this prefix is used for CICS system modules and samples. Attempting to use EDF on a CICS-supplied transaction has no effect. However, you can use EDF with CICS sample programs and some user-replaceable modules. (For example, you can use EDF to debug DFHPEP.)

EDF intercepts the execution of CICS commands in the application program at various points, allowing you to see what is happening. Each command is displayed before execution, and most are displayed after execution is complete. Screens sent by the application program are preserved, so you can converse with the application program during testing, just as a user would on a production system.

When a transaction runs under EDF control, EDF intercepts it at the following points, allowing you to interact with it:

- At **program initiation** , after the EXEC interface block (EIB) has been updated, but before the program is given control.
- At the **start of the execution of each CICS command** . This interrupt happens after the initial trace entry has been made, but before the command has been performed. Both standard CICS commands and the Front End Programming Interface (FEPI) commands are intercepted. EXEC DLI and EXEC SQL commands and any requests processed through the resource manager interface are also intercepted at this point.
- At the **end of the execution of every command** except for ABEND, XCTL, and RETURN commands (although these commands could raise an error condition that EDF displays). EDF intercepts the transaction when it finishes processing the command, but before the HANDLE CONDITION mechanism is invoked, and before the response trace entry is made.
- At **program termination**.
- At **normal task termination**.
- When an **ABEND** occurs and after **abnormal task termination**.

If you want to work through an example of EDF, see *Designing and Programming CICS Applications (ISBN 1565926765)*, which guides you through a sample EDF session.

Note: For a program translated with the option NOEDF, these intercept points still apply, apart from before and after the execution of each command. For a program with CEDF defined as NO on its

© Copyright IBM Corp. 1974, 2023

resource definition or by the program autoinstall exit, the program initiation and termination screens are suppressed as well.

Each time EDF interrupts the execution of the application program a new CEDF task is started. Each CEDF task is short lived, lasting only long enough for the appropriate display to be processed.

The terminal that you are using for the EDF interaction must be in transceive (ATI/TTI) status and be able to send and receive data. This is the most common status for display terminals, but you can find out by asking your system programmer to check its status, or you can use CEMT.

For a transaction initiated at a terminal, you can use EDF on the same terminal as the transaction you are testing, or on a different one. On the same terminal, you **must** start by clearing the screen and entering the transaction code CEDF, otherwise you may get unpredictable results. The message **THIS TERMINAL: EDF MODE ON** is displayed at the top of an empty screen. You clear the screen again and run your transaction in the normal way.

When you are using EDF, the user task is not directly purgable. If you need to terminate the task, first forcepurge the CEDF task, then attempt to press the Enter key while the EDF screen is displayed. If pressing the Enter key brings no response, forcepurge the CEDF task a second time. CEDF will terminate, and the user transaction will receive an AED3 abend.

This chapter describes:

- "Restrictions when using EDF" on page 630
- "What does EDF display?" on page 632
- "Testing programs using EDF" on page 638
- "Using EDF to change information" on page 643
- "Using EDF menu functions" on page 644

Restrictions when using EDF

When you are using EDF to debug your application programs, you must be aware of a number of restrictions.

Open TCBs and EDF
Even if your program typically runs using an OPEN TCB (L8, L9, X8, or X9), CEDF forces the program to run on the QR TCB, because CEDF itself is not threadsafe.

Parameter list stacking
CEDF only has one level of stacking for its copies of the EXEC CICS parameter list. If an application calls an EXEC-capable global user exit or user-replaceable module (URM), the parameter list for the EXEC CICS commands issued by the global user exit or URM might overlay the parameter list for EXEC CICS commands issued by the main program.

Security considerations

EDF is such a powerful tool that your installation might restrict its use with attach-time security. The external security manager used by your installation defines the security attributes for the EDF transaction. If you are not authorized to use CEDF, you cannot initiate the transaction.

Application prerequisites

User application programs that are to be debugged using EDF must be assembled (or compiled) with the translator option EDF, which is the default. If you specify NOEDF, the program cannot be debugged using EDF. There is no performance advantage in specifying NOEDF, but the option can be useful to prevent commands in already debugged subprograms appearing on EDF displays.

Application programs that are to be debugged using EDF must also have the attribute CEDF(YES) in their resource definition, which is the default. If a program is defined with CEDF(YES) and compiled with the translator option EDF, EDF diagnostic screens are displayed for the program. If the program is defined with CEDF(YES) but compiled with the translator option NOEDF, only the program initiation and termination screens are displayed. If CEDF(NO) is specified, no EDF screens are displayed.

If a program with the attribute CEDF(NO) links to a program with the attribute CEDF(YES), it might not be possible to use EDF for the transaction. For example, if the CICSPlex SM dynamic transaction routing program EYU9XLOP is defined with the attribute CEDF(NO), and the user-replaceable program EYU9WRAM (for workload management processing) is defined with the attribute CEDF(YES), you cannot use EDF to debug EYU9WRAM. For successful debugging of multiple programs within a transaction, ensure that all the programs are defined with CEDF(YES).

Restrictions for single-screen mode

There are some restrictions on the use of EDF that make it preferable or even necessary to use one particular screen mode:

- EDF can be used only in single-screen mode when running a remote transaction.
- VM PASSTHRU is not supported by EDF when testing in single-screen mode.
- In single-screen mode, the user transaction and CEDF must not specify message journaling, because the messages interfere with the EDF displays. Message journaling is controlled by the profile definition for each transaction.
- In single screen mode, do not specify PROTECT=YES in the profile definition of the CEDF transaction. If this option is specified, message protection for the CEDF transaction is ignored. The user transaction can still specify the PROTECT=YES option even when running under CEDF. This restriction does not apply to dual-screen mode.
- If a SEND LAST command is issued, EDF is ended before the command is processed if you are using single-screen mode.
- To test an application program that uses screen partitions, or that does its own request unit (RU) chaining, run in dual-screen mode.
- In single-screen mode, if the profile for the user transaction specifies INBFMH=ALL or INBFMH=DIP, the profile for CEDF must have the same INBFMH value. Otherwise the user transaction ends with the ADIR abend. Dual-screen mode does not require the profiles to match in this respect.
- If the inbound reply mode is set to `character` to enable the attribute setting keys, EDF disables the keys in single-screen mode.
- When using CECI under EDF in dual-screen mode, certain commands (for example, ASSIGN and ADDRESS) are issued against the EDF terminal and not the transaction terminal. See INVOKE CECI for information about how to invoke CECI from CEDF.
- When using EDF in dual-screen mode, avoid starting a second task at the EDF terminal, for example by issuing a START command. Because EDF is a pseudoconversational transaction, it does not prevent a second task from starting at the terminal it is using. This might lead to a deadlock in certain circumstances.
- When using EDF screen suppression in dual screen mode, commands that cause a long wait, such as DELAY, WAIT, or a second RECEIVE, might cause EDF to appear as if it has finished. If the task ends abnormally, EDF is reactivated at the monitoring terminal.

Restrictions for both screen modes

Other restrictions apply to both screen modes:

- If a transaction issues the FREE command, EDF is switched off without warning.
- EDF does not intercept calls to the CPI Communications interface (CPI-C) or the SAA Resource Recovery interface (CPI-RR). You can test transactions that use CPI calls under EDF, but you cannot see EDF displays at the call points.
- When processing a SIGNON command, CEDF suppresses display of the password or password phrase value to reduce the risk of accidental disclosure.
- When using EDF against a connection, file control commands functions that are shipped over the connection will not be displayed due to the implementation of CEDF.

What does EDF display?

All EDF displays have the same general format, but the contents depend on the point at which the task was interrupted. The display indicates which of these interception points has been reached and shows information relevant to that point.

Figure 141 on page 632 shows a typical display; occurring after execution of a **SEND MAP** command.

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00032 APPLID: 1234567
DISPLAY:00
STATUS: COMMAND EXECUTION COMPLETE
 1
EXEC CICS SEND MAP
MAP ('T1 ')
FROM ('.....................................................'...)
LENGTH (154)
MAPSET ('DFH0T1 ')
CURSOR
 2
TERMINAL
ERASE
NOFLUSH
NOHANDLE



OFFSET:X'002522' LINE:00673 EIBFN=X'1804'
RESPONSE: NORMAL EIBRESP=0
 3
ENTER: CONTINUE
 4
F1 : UNDEFINED F2 : SWITCH HEX/CHAR F3 : END EDF SESSION
F4 : SUPPRESS DISPLAYS F5 : WORKING STORAGE F6 : USER DISPLAY
F7 : SCROLL BACK F8 : SCROLL FORWARD F9 : STOP CONDITIONS
F10: PREVIOUS DISPLAY F11: EIB DISPLAY F12: ABEND USER TASK
```

Figure 141. Typical EDF display

Note:

1. Header
2. Body
3. Message line
4. Menu of functions

The display consists of a header, a body (the primary display area), a message line, and a menu of functions you can select at this point. If the body does not fit on one screen, EDF creates multiple screens, which you can scroll through using function keys F7 and F8. The header, menu, and message areas are repeated on each screen.

The header
The header of the display contains the following information; the identifier of the transaction being executed, the name of the program being executed, the internal task number assigned by CICS to the transaction, the applid of the CICS region where the transaction is being executed, a display number, and the STATUS, that is, the reason for the interception by EDF.

The body
The body or main part of the display contains the information that varies with the point of intercept. The following screens show you the body contents at program initiation, at the start and the end of execution of a CICS command, at program and task termination, and at abnormal termination.

EDF Example: At program initiation

This example shows you the display at program initiation. EDF displays the COMMAREA (if any) and the contents of the principal fields in the EIB.

For programming information about these EIB fields, see Attention identifier constants, DFHAID. If there isn't a COMMAREA, line 4 on the screen remains blank and EIBCALEN has a value of zero.

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00032 APPLID:
1234567 DISPLAY:00
STATUS: PROGRAM INITIATION

COMMAREA = '3476559873'
EIBTIME = 92920
EIBDATE = 91163
EIBTRNID = 'AC20'
EIBTASKN = 32
EIBTRMID = 'S246'

EIBCPOSN = 4
EIBCALEN = 10
EIBAID = X'7D' AT X'032F059A'
EIBFN = X'0000' AT X'032F059B'
EIBRCODE = X'000000000000' AT X'032F059D'
EIBDS = '........'
+ EIBREQID = '........'


ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: UNDEFINED
```

Figure 142. Typical EDF display at program initiation

EDF Example: At the start of execution of a CICS command

This example shows you the display at the start of execution of a CICS command. EDF displays the command, including keywords, options, and argument values.

A typical EDF display at start of execution of a CICS command is shown in Figure 143 on page 633. You can display the information in hexadecimal or character form (and switch from one to the other) by pressing PF2. If character format is requested, numeric arguments are shown in signed numeric character format.

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00032 APPLID:
1234567 DISPLAY:00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS SEND MAP
MAP ('T1 ')
FROM ('..........................................................'..)
LENGTH (154)
MAPSET ('DFH0T1 ')
CURSOR
TERMINAL
ERASE
NOFLUSH
NOHANDLE




OFFSET:X'002522' LINE:00673 EIBFN=X'1804'

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: ABEND USER TASK
```

Figure 143. Typical EDF display at start of execution of a CICS command

Figure 144 on page 634 shows a similar screen for the **start of execution** of an EXEC SQL command running with Db2.

```
TRANSACTION: LOKO PROGRAM: TLOKO TASK: 00082 APPLID: 1234567
DISPLAY:00
STATUS: ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER DSNCSQL
EXEC SQL UPDATE
DBRM=TLOK0, STMT=00242, SECT=00001
IVAR 001: TYPE=CHAR, LEN=00010 AT X'001E5A99'
DATA=X'F0F0F0F0F0F1F0F0F0F0'









OFFSET:X'000298' LINE: UNKNOWN EIBFN= X'0A02'
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : UNDEFINED PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: ABEND USER TASK
```

Figure 144. Typical SQL display at start of execution of a SQL command

In addition to options and values, the command is identified by its hexadecimal offset within the program. If the program was translated with the DEBUG translator option, the line number also appears, as shown in Figure 143 on page 633. (See “Defining translator options” on page 592 for information about this option.)

At the start of an EXEC SQL or EXEC DLI command, the body of the EDF display shows you the parameter list of the CALL to which your command translates. If a DLI command generates multiple CALL statements, you see only the last CALL statement.

EDF Example: At the end of execution of a command

This example shows you the display at end of execution of a command. EDF provides a display in the same format as at the start of the command. At this point, you can see the effects of executing the command, in the values of the variables returned or changed and in the response code.

EDF does not provide this display for the ABEND, XCTL, and RETURN commands (although these commands could raise an error condition that EDF displays). The completion screen corresponding to the **about to execute** screen in Figure 143 on page 633 is shown in Figure 145 on page 635.

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00054 APPLID:
1234567 DISPLAY:00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS SEND MAP
MAP ('T1 ')
FROM ('.....................................................'...)
LENGTH (154)
MAPSET ('DFH0T1 ')
CURSOR
TERMINAL
ERASE
NOFLUSH
NOHANDLE



OFFSET:X'002522' LINE:00673 EIBFN=X'1804'
RESPONSE: NORMAL EIBRESP=0

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: ABEND USER TASK
```

Figure 145. Typical EDF display at completion of a CICS command

For CICS commands, response codes are described both by name (for example, NORMAL or NOTFND) and by the corresponding EIBRESP value in decimal form. For DL/I, the response code is a 2-character DL/I status code, and there is no EIBRESP value. Programming information, including a list of EIBRESP codes, is in EIB fields, and DL/I codes are documented in DL/I status codes in IMS product documentation.

Figure 146 on page 635 and Figure 147 on page 636 show typical screens for an EXEC DLI command.

```
TRANSACTION: XDLI PROGRAM: UPDATE TASK: 00111 APPLID:
1234567 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC DLI GET NEXT
USING PCB (+00003)
FIRST
SEGMENT ('A ')
INTO (' ')
SEGLENGTH (+00012)
FIRST
VARIABLE
+SEGMENT ('B ')




OFFSET:X'000246' LINE: 00000510 EIBFN:X'000C'
RESPONSE: 'AD'

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: ABEND USER TASK
```

Figure 146. Typical EDF display at completion of a DLI command (screen one)

```
TRANSACTION: XDLI PROGRAM: UPDATE TASK: 00111 APPLID:
1234567 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC DLI GET NEXT
+
FIRST
SEGMENT ('C ')
SEGLENGTH (+00010)
LOCKED
INTO ('SMITH ')
WHERE (ACCOUNT = '12345')
FIELDLENGTH (+00005)




OFFSET:X'000246' LINE: 00000510 EIBFN:X'000C'
RESPONSE: 'AD'

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: ABEND USER TASK
```

Figure 147. Typical EDF display at completion of a DLI command (screen two)

```
TRANSACTION: LOKO PROGRAM: TLOKO TASK: 00111 APPLID: 1234567
DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER DSNCSQL
EXEC SQL UPDATE
PLAN=TLOK0, DBRM=TLOK0, STMT=00242, SECT=00001
SQL COMMUNICATION AREA:
SQLCABC = 136 AT X'001E5A18'
SQLCODE = 000 AT X'001E5A1C'
SQLERRML = 000 AT X'001E5A20'
SQLERRMC = '' AT X'001E5A22'
SQLERRP = 'DSN' AT X'001E5A68'
SQLERRD(1-6) = 000, 000, 00001, -1, 00000, 000 AT X'001E5A70'
SQLWARN(0-A) = '_ _ _ _ _ _ _ _ _ _ _' AT X'001E5A88'
SQLSTATE = 00000 AT X'001E5A93'

OFFSET:X'000298' LINE: UNKNOWN EIBFN= X'0A02'
RESPONSE:

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : UNDEFINED PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: ABEND USER TASK
```

Figure 148. Typical SQL display at completion of an SQL command

EDF Example: At program and task termination

This example shows you the display at program termination and normal task termination. There is no body information; all the pertinent information is in the header.

Figure 149 on page 637 and Figure 150 on page 637 show summarized screens for program and task termination.

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00054 APPLID:
1234567 DISPLAY:00
STATUS: PROGRAM TERMINATION









ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: ABEND USER TASK
```

Figure 149. Typical EDF display at program termination

```
TRANSACTION: AC20 TASK: 00054 APPLID: 1234567 DISPLAY: 00
STATUS: TASK TERMINATION









CONTINUE EDF? (ENTER YES OR NO) REPLY: YES
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: UNDEFINED
```

Figure 150. Typical EDF display at task termination

EDF Example: At abnormal termination

This example shows you the display when an abend or abnormal task termination occurs.

When an **abend** or **abnormal task termination** occurs, EDF displays the screens shown in Figure 151 on page 637 and Figure 152 on page 638.

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK:00054 APPLID:
1234567 DISPLAY: 00
STATUS: AN ABEND HAS OCCURRED
COMMAREA = '1287656678'
EIBTIME = 135510
EIBDATE = 91163
EIBTRNID = 'AC20'
EIBTASKN = 76
EIBTRMID = 'S232'
EIBCPOSN = 4
EIBCALEN = 10
EIBAID = X'7D' AT X'032F059A'
EIBFN = X'1804' SEND AT X'032F059B'
EIBRCODE = X'000000000000' AT X'032F059D'
EIBDS = '........'
+ EIBREQID = '........'


ABEND : ABCD

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: UNDEFINED
```

Figure 151. Typical EDF display when an abend occurs

```
TRANSACTION: AC20 TASK: 00054 APPLID: 1234567 DISPLAY: 00
STATUS: ABNORMAL TASK TERMINATION
COMMAREA = '2934564671'
EIBTIME = 135510
EIBDATE = 91163
EIBTRNID = 'AC20'
EIBTASKN = 76
EIBTRMID = 'S232'
EIBCPOSN = 4
EIBCALEN = 10
EIBAID = X'7D' AT X'032F059A'
EIBFN = X'1804' SEND AT X'032F059B'
EIBRCODE = X'000000000000' AT X'032F059D'
EIBDS = '........'
+ EIBREQID = '........'


ABEND : ABCD
CONTINUE EDF? (ENTER YES OR NO) REPLY: YES
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: UNDEFINED
```

Figure 152. Typical EDF display at abnormal task termination

The body displays the COMMAREA and the values of the fields in the EIB as well as the following items:

- The abend code
- If the abend code is ASRA (that is, a program interrupt has occurred), the program status word (PSW) at the time of interrupt, and the source of the interrupt as indicated by the PSW
- If the PSW indicates that the instruction giving rise to the interrupt is within the application program, the offset of that instruction relative to the main entry point

Testing programs using EDF

You can run EDF by using either the CEDF or CEDX transaction. If you are testing a nonterminal transaction, use the CEDX transaction, which enables you to specify the name of the transaction. If you are testing a transaction that is associated with a terminal, use the CEDF transaction.

You can typically select one of several methods for running EDF, but there are a few situations in which a specific method is required. For example, you must use single-screen mode for remote transactions. See “Restrictions when using EDF” on page 630 for other conditions that affect your choice.

Interrupting program execution

Using EDF, you can perform different operations at intercept points in your program to help debug it.

The power of EDF lies in what you can do at each of the intercept points. For example, you can:

- Change the argument values before a command is executed. For CICS commands, you cannot change the command itself, or add or delete options, but you can change the value associated with any option. You can also suppress execution of the command entirely using NOOP. See “Using EDF to change information” on page 643 for further details.
- Change the results of a command, either by changing the argument values returned by execution or by modifying the response code. This allows you to test branches of the program that are hard to reach using ordinary test data (for example, what happens on an input/output error). It also allows you to bypass the effects of an error to check whether this eliminates a problem.
- Display the working storage of the program, the EXEC interface block (EIB), and for DL/I programs, the DL/I interface block (DIB).
- Invoke the command interpreter (CECI). Under CECI you can execute commands that are not present in the program to gain additional information or change the execution environment.
- Display any other location in the CICS region.

- Change the working storage of the program and most fields in the EIB and the DIB. EDF stops your task from interfering with other tasks by preventing you from changing other areas of storage.
- Display the contents of temporary storage and transient data queues.
- Suppress EDF displays until one or more of a set of specific conditions is fulfilled. This speeds up testing.
- Retrieve up to 10 previous EDF displays or saved screens.
- Switch off EDF mode and run the application normally.
- Stop the task with an abend.

The first two types of changes are made by overtyping values in the body of the command displays. "Using EDF to change information" on page 643 tells you how to do this. You use the function keys in the menu for the others; "Using EDF menu functions" on page 644 tells you exactly what you can do and how to go about it.

```
TRANSACTION: DLID PROGRAM: DLID TASK: 00049 APPLID: IYAHZCIB
DISPLAY:00
ADDRESS: 00000000









WORKING STORAGE IS NOT AVAILABLE
ENTER: CURRENT DISPLAY
PF1 : UNDEFINED PF2 : BROWSE TEMP STORAGE PF3 : UNDEFINED
PF4 : EIB DISPLAY PF5 : INVOKE CECI PF6 : USER DISPLAY
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL PF12: REMEMBER DISPLAY
```

Figure 153. Typical EDF display from which CECI can be invoked

Using EDF in single-screen mode

When you use EDF with just one terminal, the EDF inputs and outputs are interleaved with those from the transaction.

This sounds complicated, but works quite easily in practice. The only noticeable peculiarity is that when a SEND command is followed by a RECEIVE command, the display sent by the SEND command appears twice: once when the SEND is executed, and again when the RECEIVE is executed. It is not necessary to respond to the first display, but if you do, EDF preserves anything that was entered from the first display to the second.

You can start EDF in two ways:

- Entering transaction code CEDF from a cleared screen
- Pressing the appropriate function key (if one has been defined for EDF)

Next, you start the transaction to be tested by completing the following steps:

1. Press the CLEAR key to clear the screen.
2. Enter the transaction code of the transaction you want to test.

When both EDF and the user transaction are sharing the same terminal, EDF restores the user transaction display at the following times:

- When the transaction requires input from the operator
- When you change the transaction display
- At the end of the transaction
- When you suppress the EDF displays
- When you request USER DISPLAY

To enable restoration, user displays are remembered at the following times:

1. At the start of task, before the first EDF screen for the task is displayed
2. Before the next EDF screen is displayed, if the user display has been changed
3. On leaving SCREEN SUPPRESS mode

If you use CEDF with an application program that has been translated with option NOEDF, or one that has NO specified for CEDF in its resource definition, EDF cannot ascertain when the display is changed by that application program. Therefore EDF cannot save a copy of that display for later use. The next EDF display overwrites any display sent by the application program and cannot be restored.

Similarly, CEDF cannot restore the current display when it is about to be changed by the application, or when the transaction requires input from the operator. Therefore an output command to the principal facility from the application program might result in random background information from a previous EDF display appearing on the screen.

An input command can be executed against the previous EDF display, rather than a display from the application program, or, if it is the first receive in the transaction, it might require explicit input from the CEDF panel instead of being satisfied by the contents of the initial TIOA.

These considerations apply to any screen I/O operation performed by the application program.

When EDF restores the transaction display, it does not sound the alarm or affect the keyboard in the same way as the user transaction. The effect of the user transaction options is seen when the SEND command is processed, but not when the screen is restored. When you have NOEDF specified in single-screen mode, take care that your program does not send and receive data because you will not see it.

When EDF restores the transaction display on a device that uses color, programmed symbols, or extended highlighting, these attributes are no longer present and the display is monochrome without the programmed symbols or extended highlighting. Also, if the inbound reply mode in the application program is set to `character` to enable the attribute-setting keys, EDF resets this mode, causing these keys to be disabled. If these changes prevent your transaction from executing properly, test in a dual-screen mode.

If you end your EDF session part way through the transaction, EDF restores the screen with the keyboard locked if the most recent RECEIVE command has not been followed by a SEND command; otherwise, the keyboard is unlocked.

Pseudoconversational programs

EDF makes a special provision for testing pseudoconversational transactions from a single terminal. If the terminal came out of EDF mode between the several tasks that make up a pseudoconversational transaction, it would be very hard to do any debugging after the first task. So, when a task terminates, EDF asks the operator whether EDF mode is to continue to the next task. If you are debugging a pseudoconversational task, press Enter to accept the default, which is `Yes`. If you have finished, reply `No`.

Using EDF in dual-screen mode

In dual-screen mode, you use one terminal for EDF interaction and another for sending input to, and receiving output from, the transaction that is being tested.

You start by entering, at the EDF terminal, the transaction `CEDF` *`tttt`* , where *tttt* is the name of the terminal on which the transaction is to be tested.

The message that CEDF gives in response to this transaction depends on whether there is already a transaction running on the second terminal. If the second terminal is not busy, the message displayed at the first terminal is:

```
TERMINAL tttt: EDF MODE ON
```

Nothing further happens until a transaction is started on the second terminal, when the PROGRAM INITIATION display appears.

You can also use EDF in dual-screen mode to monitor a transaction that is already running on the second terminal. If, for example, you believe a transaction at a specific terminal might be looping, you can go to another terminal and enter a CEDF transaction that names the terminal at which this transaction is running. The message displayed at the first terminal is:

```
TERMINAL tttt: TRANSACTION RUNNING: EDF MODE ON
```

EDF picks up control at the next **EXEC CICS** command executed, and you can then observe the sequence of commands that are causing the loop, assuming that at least one **EXEC CICS** command is executed.

EDF and remote transactions

You cannot use EDF in dual-screen mode if the transaction that is being tested, or the terminal that invokes it, is owned by another CICS region.

If a remote transaction ends abnormally while under EDF using a CRTE routing session, EDF displays the abnormal task termination screen, followed by message DFHAC2206 for the user transaction. The CRTE session is not affected by the user task abend. Also, if you opted to continue with EDF after the abend, your terminal remains in EDF mode within the CRTE routing session.

There is a difference in execution as well. For remote transactions, EDF purges its memory of your session at the termination of each transaction, whether EDF is to be continued or not. This means that any options you have set, and any saved screens, are lost between the individual tasks in a pseudoconversational sequence.

EDF and non-terminal transactions

Use EDF to test transactions that execute without a terminal: for example, transactions started by an **EXEC CICS START** command, or transactions initiated by a transient data trigger-level. To test nonterminal transactions, use the CEDX *trnx* command, where *trnx* is the transaction identifier.

To test a transaction using CEDX, the following conditions must be met:

- The terminal you use for the EDF displays, at which you enter the CEDX command, must be logged on to the CICS region in which the specified transaction is to execute.
- The CEDX command must be issued before the specified transaction is started by CICS. Other instances of the same transaction that are already executing when you issue the CEDX command are ignored.
- The transaction you specify on the CEDX command must run in the local CICS region. You cannot use the CRTE routing transaction followed by the CEDX transaction.

When you use CEDX to debug a transaction, CICS controls the EDF operation by modifying the definition of the transaction specified on the CEDX command, to reference a special transaction class, DFHEDFTC. When you switch off EDF (using CEDX *tranid* ,OFF) CICS modifies the transaction definition back to its normal transaction class.

If you use the read-only form of CEDX, that is CEDY, the same conditions apply. In this case, transaction class DFHEDFTO is used.

EDF and DTP programs

You can test a transaction that is using distributed transaction processing (DTP) across a remote link by telling execution diagnostic facility (EDF) to monitor the session on the link.

You can do this on either (or both) of the participating systems that are running under CICS and have EDF installed. You cannot do this if the transaction has been routed from another CICS region because you must use single-screen mode for remote transactions.

For APPC and MRO links, you can name the system identifier of the remote system:

```
CEDF sysid
```

This causes EDF to associate itself with any transaction attached across any session belonging to the specified system.

For APPC, MRO, and LU6.1 links, you can use the session identifier that the transaction is using:

```
CEDF sessionid
```

You can determine the session identifier with the **INQUIRE TERMINAL** command, but this means that the transaction must be running and must have reached the point of establishing a session before you start EDF.

If a transaction that uses distributed transaction processing also has a terminal associated with it, or if you can invoke it from a terminal (even though it does not use one), you can use EDF to test it in the usual way from that terminal.

When you have finished testing the transaction on the remote system, turn off EDF on that system identifier or session identifier before logging off from CICS with CESF. For example:

```
CEDF sysid
,OFF
```

Failure to turn off EDF could cause another transaction that is using a link to that system to be suspended.

EDF and distributed program link commands

You can use EDF, in single- or dual-terminal mode, to test a transaction that includes a distributed program link (DPL) command. However, EDF displays only the DPL command invocation and response screens. CICS commands issued by the remote program are not displayed, but a remote abend, and the message a `remote abend has occurred` , is returned to the EDF terminal, along with the system identifier of the system from which the abend was received. After control is returned to your local program, EDF continues to test as normal, but the program status word (PSW) is not displayed if the abend is in a remote program.

Stopping EDF

If you want to end EDF control of a terminal, the method you use depends on where you are in the testing.

If the transaction that is being tested is still executing and you want it to continue, but without EDF, press the END EDF SESSION function key. If you have reached the task termination intercept, EDF asks if you want to continue. If you do not, overtype the reply as `NO` (`YES` is the default). If no transaction is executing at the terminal, clear the screen and enter:

```
CEDF ,OFF
```

(The space and comma are required.)

If you are logging off from dual-screen mode, clear the screen and enter `CEDF` *`tttt`* `,OFF` .

In all these cases, the message `THIS TERMINAL: EDF MODE OFF` is displayed at the top of an empty screen.

Using EDF to change information

Most of the changes you make with EDF involve changing information in memory. To make these changes, type over the information that is displayed on the screen with the information that you require.

You can change any area that you can move the cursor to by using the tab keys, except for the menu area at the bottom of the screen.

When you change the screen, you must observe the following rules:

- On CICS command screens, you can type over any argument value, but not the keyword of the argument. You cannot remove an optional argument, and you cannot add or delete an option.
- When you change an argument in the command display (rather than the working storage screen), you can change only the part shown on the display. If you attempt to type beyond the value displayed, the changes are not made and no diagnostic message is generated. If the argument is so long that only part of it appears on the screen, change the area in working storage to which the argument points. To determine the address, display the argument in hexadecimal format so that the address of the argument location is also displayed.
- When you type over an argument value that is a fullword, the maximum value that you can enter is 2147483639.
- In character format, numeric values always have a sign field. You can type over a sign field with only a minus character (-) or a blank.
- When an argument is to be displayed in character format, some characters might not be displayable (including lowercase characters). EDF replaces each nondisplayable character with a period. If you type over a period, remember that the storage might contain a nondisplayable character.

  You cannot type over a character with a period to change it; the change is ignored and no diagnostic message is issued. To change a character to a period, switch the display to hexadecimal format, use the F2 key, and type over with the value X ' 4B ' .
- When storage is displayed in both character and hexadecimal format, if you change both formats and the changes conflict, the value of the hexadecimal field takes precedence. No diagnostic message is issued.
- The arguments for some commands, for example, HANDLE CONDITION, are program labels rather than numeric or character data. The form in which EDF displays (and accepts modifications to) these arguments depends on the programming language in use:
  - For COBOL, a null argument is displayed and you cannot modify it, for example, ERROR ( ).
  - For C and C++, labels are not valid.
  - For PL/I, the address of the label constant is used, for example, ERROR (X'001D0016').
  - For assembler language, the address of the program label is used, for example, ERROR (X'00030C').
  - For AMODE(64) assembler language, labels are not supported.

  If no label value is specified on a HANDLE CONDITION command, EDF displays the condition name alone without the parentheses.
- You can type over the response field with the name of any exception condition that can occur for the current function, including ERROR, or the word NORMAL. The effect when EDF continues is that the program takes the action that is prescribed for the specified response. You can get the same effect by changing the EIBRESP field in the EIB display to the corresponding values. If you change the EIBRESP value or the response field on the " **command execution complete** " screen, EIBRCODE is updated. EIBRESP appears on second EIB screen and is the only one you can change (EIBRCODE protected). You can get the same effect by changing the EIBRESP value on the EIB display; EDF changes related values in the EIB and command screens accordingly.
- If uppercase translation is not specified for the terminal you are using, ensure that you always enter uppercase characters.
- You can type over any command with NOOP or NOP before processing to suppress processing of the command. You can type over with blanks or use the ERASE EOF key for the same effect. When the

screen is redisplayed with NOOP, you can restore the original verb line by erasing the whole verb line with the ERASE EOF key and pressing the ENTER key.

- You can enter 64-bit addresses if the argument is already a 64-bit address, including addresses with an underscore in the middle, for example AAAAAAAABBBBBBBB or AAAAAAAA_BBBBBBBB.

When you type over a field that represents a data area in your program, the change is made directly in application program storage and is permanent. However, if you change a field that represents a constant (a program literal), program storage is not changed, because this change might affect other parts of the program that use the same constant, or other tasks using the program. The command is executed with the changed data, but when the command is displayed after processing, the original argument values reappear. For example, you might test a program that contains the following code:

```
EXEC CICS SEND MAP('MENU') END-EXEC.
```

If you use EDF to change the name from MENU to MENU2 before you run the command, the map used is MENU2, but the map displayed on the response is MENU. You can use the "previous display" key to verify that the map name you used. If you process the same command more than once, you must enter this type of change each time.

EDF responses

You can use this list of rules to learn the EDF responses to any keyboard entry.

The rules are as follows, in the order shown:

1. If the CLEAR key is used, EDF redisplays the screen with any changes ignored.
2. If invalid changes are made, EDF accepts any valid changes and redisplays the screen with a diagnostic message.
3. If the display number is changed, EDF accepts any other changes and shows the requested display.
4. If a function key is used, EDF accepts any changes and performs the action requested by the function key. Pressing ENTER with the cursor under a function key definition in the menu at the bottom of the screen is the same as pressing a function key.
5. If the ENTER key is pressed and the screen has been modified (other than the REPLY field), EDF redisplays the screen with changes included.
6. If the ENTER key is pressed and the screen has not been modified (other than the REPLY field), the effect differs according to the meaning of the ENTER key. If the ENTER key means CONTINUE, the user transaction continues to execute. If it means CURRENT DISPLAY, EDF redisplays the status display.

Using EDF menu functions

The function keys that you can use at each point are displayed in a menu at the bottom of every EDF display.

Functions that apply to all displays are always assigned to the same key, but definitions of some keys depend on the display and the intercept point. To select an option, press the indicated function key. Where a terminal has 24 function keys, EDF treats PF13 through PF24 as duplicates of PF1 through PF12. If your terminal has no PF keys, place the cursor under the option you want and press the ENTER key.

ABEND USER TASK

Terminates the task being monitored. The message `ENTER ABEND CODE AND REQUEST ABEND AGAIN` is displayed to confirm this action. Enter the abend code at the cursor position, then request this function again to abend the task with a transaction dump identified by the specified code. If you enter `NO` , the task is abended without a dump and with the 4-character default abend code of four question marks (`????`).

Abend codes that start with the character A are reserved for use by CICS. Using a CICS abend code might cause unpredictable results.

You cannot use this function if an abend is already in progress, or the task is terminating.

BROWSE TEMP STORAGE
Produces a display of the temporary storage queue CEBR *xxxx* , where *xxxx* is the terminal identifier of the terminal running EDF. This function is only available from the working storage (PF5) screen. You can use CEBR commands to display or modify temporary storage queues and to read or write transient data queues.

CONTINUE
Redisplays the current screen to incorporate any changes. If you made no changes, CONTINUE causes the transaction under test to resume execution up to the next intercept point. To continue, press ENTER.

CURRENT DISPLAY
Redisplays the current screen to incorporate any changes. If you made no changes, EDF displays the command screen for the last intercept point. To execute this function, press ENTER from the appropriate screen.

DIB DISPLAY
Shows the contents of the DL/I interface block (DIB). This function is only available from the working-storage screen (PF5). See IMS messages and codes in IMS product documentation for information about DIB fields.

EIB DISPLAY
Displays the contents of the EXEC interface block (EIB). For an example of an EIB display, see Figure 142 on page 633. For programming information about the EIB, see EIB fields. If COMMAREA exists, EDF also displays its address and one line of data in the dump format.

INVOKE CECI
Accesses the command-level interpreter (CECI). This function is only available from the working storage (PF5) screen. See Figure 153 on page 639 for an example of the screen from which CECI is invoked. You can then use CECI commands, discussed in "Command-level interpreter (CECI)" on page 656. These CECI commands include **INQUIRE** and **SET** commands against the resources referenced by the original command before and after command execution. See inbound reply mode for restrictions when running CECI in dual-screen mode. The use of CECI from this panel is like the use of CEBR within CEDF.

END EDF SESSION
Ends the EDF control of the transaction. The transaction continues running from that point but no longer runs in EDF mode.

NEXT DISPLAY
If you returned to a previous display, displays the next one forward and increases the display number by one. This option is the reverse of PREVIOUS DISPLAY.

PREVIOUS DISPLAY
Sends the previous display to the screen, unless you saved other displays. The number of the display from the current intercept point is always 00. As you request previous displays, the display number decreases by 1 to -01 for the first previous display, -02 for the one before that, and so on, down to the oldest display, -10. When no more previous screens are available, the PREVIOUS option is no longer available on the menu, and the corresponding function key is inoperative.

REGISTERS AT ABEND
Displays storage that contains the values of the registers if a local ASRA abend occurs. The layout of the storage is 64-bit registers, followed by 16-byte program status word (PSW) at abend. Figure 154 on page 646 shows a typical screen.

In some cases, when a second program check occurs in the region before EDF has captured the values of the registers, this function does not appear on the menu of the abend display. If this situation occurs, a second test run might provide more information.

```
TRANSACTION: UT PROGRAM: ASRA31 TASK: 0000487 APPLID: IYK2ZKE1
DISPLAY: 00
ADDRESS: 0006F010
0006F010 000000 00000000 00000090 00000000 0010003C ................
 1
0006F020 000010 00000000 00041800 00000000 AE410028................
 1
0006F030 000020 00000000 2C2E35A8 00000000 00000000.......y........
 1
0006F040 000030 00000000 2C2D7800 00000000 7F2C9918............".r.
 1
0006F050 000040 00000000 2E410000 00000000 2C2D7818................
 1
0006F060 000050 00000000 00100008 00000000 00100100................
 1
0006F070 000060 00000000 009AF000 00000000 00100690......0.........
 1
0006F080 000070 00000000 AE410158 00000000 00100690................
 1
0006F090 000080 079D0000 80000000 00000000 2E410178................
 2
0006F0A0 000090 00040004 00000000 00000000 00000000................
0006F0B0 0000A0 00000000 00000000 00000000 00000000 ................
0006F0C0 0000B0 00000000 00000000 00000000 00000000................
0006F0D0 0000C0 00000000 00000000 00000000 00000000 ................
0006F0E0 0000D0 00000000 00000000 D4F0F0F0 F0F4F8F9........M0000489
0006F0F0 0000E0 D4F0F0F0 F0F4F8F9 00000000 00000000 M0000489........
0006F100 0000F0 00000000 000000E4 E3404000 00F00000 .......UT..0..

ENTER: CURRENT DISPLAY
PF1 : UNDEFINED PF2 : BROWSE TEMP STORAGE PF3 : UNDEFINED
PF4 : EIB DISPLAY PF5 : INVOKE CECI PF6 : USER DISPLAY
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL PF12: REMEMBER
DISPLAY
```

Figure 154. Typical EDF display for REGISTERS AT ABEND

Notes:

1. Register values
2. PSW

REMEMBER DISPLAY
Places a display that would not typically be kept in memory, such as an EIB display, in the EDF memory. EDF automatically saves the displays at the start and completion of each command. The memory can hold up to 10 displays. The displays are numbered in reverse chronological order (that is, -10 is the oldest display, and -01 is the newest). All pages associated with the display are kept in memory and can be scrolled when recalled. Note, however, that if you save a working-storage display, only the screen on view is saved.

SCROLL BACK
View the previous screen in the display. This function applies to an EIB, DIB, or command display that does not all fit on one screen. When the screen on view is not the first one of the display, and there is a plus sign (+) before the first option or field, select this function to view previous screens in the display. See Figure 142 on page 633 for an example.

SCROLL FORWARD
View the next screen in the display. This function applies to an EIB, DIB, or command display that does not all fit on one screen. When the display does not fit, a plus sign (+) appears after the last option or field in the display, to show that there are more screens. Select this function to display the next screen.

SCROLL BACK FULL
View the previous screen in a working-storage display. This function applies to displays of working storage and is similar to the SCROLL BACK option for EIB and DIB displays. SCROLL BACK FULL gives a working-storage display one full screen backward, showing addresses lower in storage than those addresses on the current screen.

SCROLL FORWARD FULL
View the next screen in a working-storage display. This function applies to displays of working storage and is similar to the SCROLL FORWARD option for EIB and DIB displays. SCROLL FORWARD FULL gives a working-storage display one full screen forward, showing addresses higher in storage than those addresses on the current screen.

SCROLL BACK HALF
Reverse the display of working storage by half a screen. This function is similar to SCROLL BACK FULL, except that the display of working storage is reversed by only half a screen.

SCROLL FORWARD HALF
Advance the display of working storage by half a screen. This function is similar to SCROLL FORWARD FULL, except that the display of working storage is advanced by only half a screen.

STOP CONDITIONS
Specify which conditions cause EDF to resume displays after using the SUPPRESS DISPLAYS function. The menu screen shown in Figure 155 on page 647 is displayed. You can use the STOP CONDITIONS and SUPPRESS DISPLAYS functions together to reduce the interaction when you check a program that you know is partly working.

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 0086 APPLID:
1234567 DISPLAY: 00
DISPLAY ON CONDITION:-

COMMAND: EXEC CICS
OFFSET: X'......'
LINE NUMBER:
CICS EXCEPTION CONDITION: ERROR
ANY CICS CONDITION NO
TRANSACTION ABEND YES
NORMAL TASK TERMINATION YES
ABNORMAL TASK TERMINATION YES

DLI ERROR STATUS:
ANY DLI ERROR STATUS




ENTER: CURRENT DISPLAY
PF1 : UNDEFINED PF2 : UNDEFINED PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : UNDEFINED PF8 : UNDEFINED PF9 : UNDEFINED
PF10: UNDEFINED PF11: UNDEFINED PF12: REMEMBER DISPLAY
```

Figure 155. Typical EDF display for STOP CONDITIONS

By default, EDF resumes displays if any of the following conditions occur, as shown in Figure 155 on page 647 :

- Any CICS exception condition
- Transaction abend
- Normal task termination
- Abnormal task termination

You can use the STOP CONDITIONS menu to turn off any default that does not apply, and add conditions specific to your program.

You can specify any or all these events as STOP CONDITIONS:

- A specific type of function and option, such as READNEXT file or ENQ resource, is encountered; for example, FEPI ADD or GDS ASSIGN.
- The command at a specific offset or on a specific line number (assuming that the program was translated with the DEBUG option) is encountered.
- Any DL/I error status occurs, or a specific DL/I error status occurs.

- A specific exception condition occurs. When ERROR (the default) is specified for CICS EXCEPTION CONDITION, EDF redisplays a screen in response to any error condition (for example, NOTOPEN, EOF, or INVREQ). If you specify a specific condition, such as EOF, for CICS EXCEPTION CONDITION, and NO (the default) is specified for ANY CICS CONDITION, EDF redisplays the screen only when that condition (EOF) occurs.

  If you specify YES for ANY CICS CONDITION, EDF overrides the CICS exception conditions and redisplays a screen whenever any command results in a non-zero EIBRESP value such as NOTOPEN, EOF, or QBUSY.
- Any exception condition occurs for which the CICS action is to raise ERROR; for example, INVREQ or NOTFND.
- An abend occurs.
- The task ends normally.
- The task ends abnormally.

When you use an offset for STOP CONDITIONS, you must specify the offset of the BALR instruction corresponding to a command. The offset can be determined from the code listing produced by the compiler or assembler. In COBOL, C, C++, or PL/I, you must use the compiler option that produces the assembler listing to determine the relevant BALR instruction.

When you use a line number, you must specify it exactly as it appears on the listing, including leading zeros, and it must be the line on which a command starts. If you have used the NUM or the SEQUENCE translator options, the translator uses your line numbers as they appear in the source. Otherwise, the translator assigns line numbers.

Line numbers can be found in the translator listing (SYSPRINT in the translator step) if you have used either the SOURCE or VBREF translator options. If you have used the DEBUG translator option, as you must to use line numbers for STOP CONDITIONS, the line number also appears in your compilation (assembly) listing, embedded in the translated form of the command, as a parameter in the CALL statement.

You can specify that EDF resumes displays at DL/I commands as well as at CICS commands. Type over the CICS qualifier on the command line with `DLI` and enter the type of DL/I command at which you want display suppression to stop. You must be running a DL/I program or have executed one earlier in the same task. You can suppress DL/I commands as early as the program initiation panel.

You can also stop display suppression when a particular DL/I status code occurs. For information about the status codes that you can use, see the list of codes in the DL/I interface block (DIB) in IMS messages and codes in IMS product documentation.

SUPPRESS DISPLAYS
Suppresses all EDF displays until one of the specified STOP CONDITIONS occurs. When the condition occurs, however, you still have access to the ten previous command displays, even though they were not sent to the screen when they were originally created.

SWITCH HEX/CHAR
Switches displays between character and hexadecimal form. The switch applies only to the command display and does not affect previously remembered displays, STOP CONDITIONS displays, or working storage displays.

In DL/I command displays that contain the WHERE option, only the key values (the expressions that follow each comparison operator) can be converted to hexadecimal.

UNDEFINED
The indicated function key is not defined for the current display at the current intercept point.

USER DISPLAY
Displays what would be on the screen if the transaction was not running in EDF mode. (You can use it only for single terminal checkout.) To return to EDF after using this key, press the ENTER key.

WORKING STORAGE
Shows the contents of the 24-bit or 31-bit working storage area in your program or of any other address in the CICS region. Figure 156 on page 649 shows a typical working storage screen.

This function does not support 64-bit storage.

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00030 APPLID:
1234567 DISPLAY:00
ADDRESS: 035493F0 WORKING STORAGE
035493F0 000000 E3F14040 00000000 00010000 00000000 T1 ..........
03549400 000010 00000000 00000000 F1000000 00000000........1.......
03549410 000020 F0000000 00000000 F0000000 00000000 0.......0.......
03549420 000030 F0000000 00000000 F0000000 00000000
0.......0.......
03549430 000040 00000000 00000000 00000000 00000000 ................
03549440 000050 D7C1D5D3 00000000 D9C5C3C4 00000000
PANL....RECD....
03549450 000060 D3C9E2E3 00000000 C8C5D3D7 00000000 LIST....HELP....
03549460 000070 84000000 00000000 A4000000 00000000
d.......u.......
03549470 000080 82000000 00000000 C4000000 00000000 b.......D.......
03549480 000090 E4000000 00000000 C2000000 00000000
U.......B.......
03549490 0000A0 D5000000 00000000 E2000000 00000000 N.......S.......
035494A0 0000B0 7B000000 00000000 6C000000 00000000
#.......%.......
035494B0 0000C0 4A000000 00000000 F1000000 00000000 ¢.......1.......
035494C0 0000D0 F2000000 00000000 F3000000 00000000
2.......3.......

ENTER: CURRENT DISPLAY
PF1 : UNDEFINED PF2 : BROWSE TEMP STORAGE PF3 : UNDEFINED
PF4 : EIB DISPLAY PF5 : INVOKE CECI PF6 : USER DISPLAY
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL PF12: REMEMBER
DISPLAY
```

Figure 156. Typical EDF display for working storage

The working storage contents are displayed in a form like that of a dump listing; that is, in both hexadecimal and character representation. The address of working storage is displayed at the top of the screen. You can browse through the entire area using the scroll commands, or you can enter a new address at the top of the screen. This address can be anywhere in the CICS region. The working storage display provides two additional scrolling keys, and a key to display the EIB (the DIB if the command is a DL/I command).

The meaning of *working storage* depends on the programming language of the application program, as follows:

COBOL
All data storage defined in the WORKING-STORAGE section of the program

C, C++ and PL/I
The dynamic storage area (DSA) of the current procedure

Assembler language
The storage defined in the current DFHEISTG DSECT

Assembler language programs do not always acquire working storage; for example, it might not be required if the program does not issue CICS commands. When you LINK to such a program, the following message might be issued: `Register 13 does not address DFHEISTG`. This message does not necessarily mean an error, but there is no working storage to look at.

Except for COBOL programs, working storage starts with a 72 byte standard format save area; that is, registers 14 to 12 begin at offset 12, and register 13 is stored at offset 4. For AMODE(64) programs, working storage starts with a format 4 save area (F4SA); registers 14 to 12 begin at offset 8, and register 13 is stored at offset 128.

Working storage can be changed at the screen; ou can use either the hexadecimal section or the character section. You can type over the ADDRESS field at the head of the display with a hexadecimal address; storage starting at that address is then displayed when you press ENTER. You can examine any location in the address space. For more information, see “Using EDF to change information” on page 643.

If you are examining program storage that is not part of the working storage of the program currently running, which is unique to the particular transaction under test, the corresponding field on the screen is protected to prevent you from overwriting storage that might belong to or affect another task.

If the initial part of a working storage display line is blank, the blank portion is not part of working storage. This situation can occur because the display is doubleword aligned.

At the beginning and end of a task, working storage is not available. In these circumstances, EDF generates a blank storage display so that you can still examine any storage area in the region by typing over the address field.

If you terminate a PL/I or Language Environment program with an ordinary non-CICS return, EDF does not intercept the return, and you cannot see working storage. If you use a RETURN command instead, you get an EDF display before execution and at program termination.

If you are using a Language Environment -enabled program, working storage is freed at program termination if the program is terminated using a non-CICS return. In this case, working storage is not available for display.

Temporary storage browse (CEBR)

You can use the browse transaction (CEBR) to browse temporary storage queues and delete them. You can also use the CEBR transaction to transfer the contents of a transient data queue to temporary storage in order to look at them, and to reestablish the transient data queue when you have finished.

The CEBR commands that perform these transfers allow you to add records to a transient data queue and remove all records from a transient data queue.

Some installations restrict the use of the CEBR transaction, particularly in production systems, to prevent modifications that were not intended or not authorized. Installations also can protect individual resources, including temporary storage and transient data queues. If you are using the CEBR transaction and experience an abend described as a security failure, you probably have attempted to access a queue to which your user ID is not authorized.

This chapter describes:

- “Using the CEBR transaction” on page 650
- “What does the CEBR transaction display?” on page 652
- “Using the CEBR function keys” on page 653
- “Using the CEBR commands” on page 654
- “Using the CEBR transaction with transient data” on page 656

Using the CEBR transaction

You start the CEBR transaction by entering the transaction identifier CEBR, followed by the name of the queue you want to browse.

You can enter a name using as many as 16 characters. For example, to display the temporary storage queue named AXBYQUEUENAME111 you type CEBR AXBYQUEUENAME111 and press ENTER. If the queue name includes lowercase characters, ensure that uppercase translation is suppressed for the terminal you are using, and then enter the correct combination of upper and lowercase characters. CICS responds with a display of the queue, for example, as shown in Figure 157 on page 651.

Alternatively, you can start the CEBR transaction from the CEDF transaction. You do this by pressing PF5 from the initial CEDF screen (see Figure 141 on page 632) which takes you to the working-storage screen, and then pressing PF2 from that screen to browse temporary storage (that is, invoke the CEBR transaction). CEBR can also be started from CEMT I TSQ by entering 'b' at the queue to be browsed. The CEBR transaction responds by displaying the temporary storage queue whose name consists of the four letters CEBR followed by the four letters of your terminal identifier. (CICS uses this same default queue name if you invoke the CEBR transaction directly and do not supply a queue name.) The result of invoking the CEBR transaction without a queue name or from an EDF session at terminal S21A is shown in Figure

158 on page 652. If you enter the CEBR transaction from the CEDF transaction, you return to the EDF panel when you press PF3 from the CEBR screen.

If you want to use CEBR to display data from a temporary storage queue that has one or more embedded blanks in the queue name, you must enter the queue name in the TS QUEUE field from the CEBR screen, as shown in Figure 157 on page 651. If you type such a queue name immediately after entering CEBR, unpredictable results will occur

```
 CEBR TSQ AXBYQUEUENAME111 SYSID CIJP REC 1 OF 3 COL 1 OF 5
ENTER COMMAND ===>
************************** TOP OF QUEUE ******************************
00001 HELLO
00002 HELLO
00003 HELLO
************************* BOTTOM OF QUEUE ****************************









PF1 : HELP PF2 : SWITCH HEX/CHAR PF3 : TERMINATE BROWSE
PF4 : VIEW TOP PF5 : VIEW BOTTOM PF6 : REPEAT LAST FIND
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL PF12: UNDEFINED
```

Figure 157. Typical CEBR display of temporary storage queue contents

```
CEBR TSQ AXBYQUEUEAME1AA SYSID CIJP REC 1 OF 0 COL 1 OF
 1
ENTER COMMAND ===>
 2
************************** TOP OF QUEUE
*****************************
************************ BOTTOM OF QUEUE ****************************



 3 









TS QUEUE AXBYQUEUEAME1AA DOES NOT EXIST
 4
PF1 : HELP PF2 : SWITCH HEX/CHAR PF3 : TERMINATE BROWSE
 5
PF4 : VIEW TOP PF5 : VIEW BOTTOM PF6 : REPEAT LAST FIND
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL PF12: UNDEFINED
```

1
Header

2
Command area

3
Body

4
Message line

5
Menu of options

Figure 158. Typical CEBR display of default temporary storage queue

What does the CEBR transaction display?

A CEBR transaction display consists of a header, a command area, a body (the primary display area), a message line, and a menu of functions you can select at this point.

The header

The header shows:

- The transaction being run, that is, CEBR.
- The identifier of the temporary storage queue (AXBYQUEUEAME111 in Figure 157 on page 651 and (AXBYQUEUEAME1AA in Figure 158 on page 652). You can overtype this field in the header if you want to switch the screen to another queue. If the queue name includes lowercase characters, ensure that uppercase translation is suppressed for the terminal you are using, and then enter the correct combination of upper and lowercase characters.
- The system name that corresponds to a temporary storage pool name or to a remote system. If you have not specified one, the name of the local system is displayed. You can overtype this field in the header if you want to browse a shared or remote queue.
- The number of the highlighted record.
- The number of records in the queue (three in AXBYQUEUEAME111 and none in AXBYQUEUEAME1AA)

- The position in each record at which the screen starts (position 1 in both cases) and the length of the longest record (22 for queue AXBYQUEUEAME111 and zero for queue AXBYQUEUEAME1AA).

The command area

The command area is where you enter commands that control what is to be displayed and what function is to be performed.

These commands are described in “Using the CEBR commands” on page 654 . You can also modify the screen with function keys shown in the menu of options at the bottom of the screen. The function keys are explained in “Using the CEBR function keys” on page 653.

The body

The body is where the queue records are shown. Each line of the screen corresponds to one queue record.

If a record is too long for the line, it is truncated. You can change the portion of the record that is displayed, however, so that you can see an entire record on successive screens. If the queue contains more records than will fit on the screen, you can page forward and backward through them, or specify at what record to start the display, so that you can see all the records you want.

The message line

CEBR uses the message line between the body and menu to display messages to the user.

For example, the "Does not exist" message shown in Figure 158 on page 652.

Using the CEBR function keys

The function keys that you can use at any time are displayed at the bottom of every CEBR transaction screen, and have the same meaning on all screens.

If your terminal does not have PF keys, you can simulate their use by placing the cursor under the description and pressing ENTER. Where a terminal has 24 function keys, the CEBR transaction treats PF13 through PF24 as duplicates of PF1 through PF12 respectively.

PF1 HELP
Displays a help screen that lists all the commands you can use when the CEBR transaction is running. You can return to the main screen by pressing ENTER.

PF2 SWITCH HEX/CHAR
Switches the screen from character to hexadecimal format, and back again.

PF3 TERMINATE BROWSE
Terminates the CEBR transaction. If you entered the CEBR transaction directly, it frees up your terminal for the next transaction. If you entered from an EDF session, it returns you to the working-storage screen from which you entered. If you entered from CEMT I TSQ, it returns you to the CEMT screen.

PF4 VIEW TOP
Displays the first records in the queue and has the same effect as the TOP command.

PF5 VIEW BOTTOM
Displays the last records in the queue and has the same effect as the BOTTOM command.

PF6 REPEAT LAST FIND
Repeats the previous FIND command.

PF7 SCROLL BACK HALF
Moves the display backward by one-half the number of records that fit on the screen, so that the records on the top half of the screen move to the bottom half.

PF8 SCROLL FORWARD HALF
Advances the display by one-half the number of records that fit on the screen, so that the records on the bottom half of the screen move to the top half.

PF9 VIEW RIGHT (or VIEW LEFT)
Changes the screen to show the columns immediately after (to the right of) or before (to the left of) the columns currently on display. The key is not defined if the entire record fits on one line of the screen. It moves you to the right until the end of the record is reached, and then reverses to move left back to the beginning of the record. You can also use the COLUMN command to change the column at which the display begins.

PF10 SCROLL BACK FULL
Moves the screen backward by the number of records that fit on the screen, to show the records immediately before those currently on display.

PF11 SCROLL FORWARD FULL
Advances the screen by the number of records that will fit on the screen, to show the records immediately after those currently on display.

Using the CEBR commands

CEBR provides a number of commands that you can use to view and manipulate the records in the temporary storage queue.

BOTTOM
(Abbreviation: B)

Shows the last records in the temporary storage queue (as many as fill up the body of the screen, with the last record on the last line).

COLUMN nnnn
(Abbreviation: C nnnn)

Displays the records starting at character position (column) nnnn of each record. The default starting position, assumed when you initiate the CEBR transaction, is the first character in the record.

FIND /string
(Abbreviation: F /string)

Finds the next occurrence of the specified string. The search starts in the record *after* the **current record**. The current record is the one that is highlighted. In the initial display of a queue, the current record is set to one, and therefore the search begins at record two.

If the string is found, the record containing the string becomes the highlighted line, and the display is changed to show this record on the second line. If you cannot see the search string after a successful FIND, it is in columns of the record beyond those on display; use the scroll key or the COLUMN command to shift the display right or left to show the string.

For example:

```
FIND /05-02-93
```

locates the next occurrence of the string "05-02-93" The **/** character is a delimiter. It does not have to be **/** , but it must not be a character that appears in the search argument. For example, if the string you were looking for was "05/02/93" instead of "05-02-93" , you could not use the following:

```
FIND /05/02/93
```

There is a slash in the search string. The following examples would work:

```
FIND X05/02/93 or FIND S05/07/93
```

Any delimiter except a **/** or one of the digits in the string works. If there are any spaces in the search string, you must repeat the delimiter at the end of the string. For example:

```
FIND /CLARE JACKSON/
```

The search string is not case-sensitive. When you have entered a FIND command, you can repeat it (that is, find the next occurrence of the string) by pressing PF6.

GET xxxx
(Abbreviation: G xxxx)

Transfers the named transient data queue to the end of the temporary storage queue currently on display. This enables you to browse the contents of the queue. xxxx must be either the name of an intrapartition transient data queue, or the name of an extrapartition transient data queue that has been opened for input. See "Using the CEBR transaction with transient data" on page 656 for more information about browsing transient data queues.

LINE nnnn
(Abbreviation: L nnnn)

Starts the body of the screen at the queue record one prior to nnnn, and sets the current line to nnnn. (This arrangement causes a subsequent FIND command to start the search after record nnnn.)

PURGE
Deletes the queue being browsed.

Do not use PURGE to delete the contents of an internally generated queue, such as a BMS logical message.

Note: If you purge a recoverable temporary storage queue, no other task can update that queue (add a record, change a record, or purge) until your task ends.

PUT xxxx
(Abbreviation: P xxxx)

Copies the temporary storage queue that is being browsed to the named transient data queue. xxxx must be either the name of an intrapartition transient data queue, or the name of an extrapartition transient data queue that has been opened for output. See "Using the CEBR transaction with transient data" on page 656 for more information about creating or restoring a transient data queue.

QUEUE xxxxxxxxxxxxxxxx
(Abbreviation: Q xxxxxxxx)

Changes the name of the queue you are browsing. The value that you specify can be in character format using up to 16 characters (for example, QUEUE ABCDEFGHIJKLMNOP) or in hexadecimal format (for example, QUEUE X'C1C2C3C4'). If the queue name includes lowercase characters, ensure that uppercase translation is suppressed for the terminal you are using, and then enter the correct combination of upper and lowercase characters. The CEBR transaction responds by displaying the data that is in the named queue.

You can also change the queue name by overtyping the current value in the header.

SYSID xxxx
(Abbreviation: S xxxx)

Changes the name of the temporary storage pool or remote system where the queue is to be found.

You can also change this name by overtyping the current SYSID value in the header.

Note: If ISC is not active in the CICS system on which the CEBR transaction is running then the SYSID will default to the local SYSID.

TERMINAL xxxx
(Abbreviation: TERM xxxx)

Changes the name of the queue you are browsing, but is tailored to applications that use the convention of naming temporary storage queues that are associated with a terminal by a constant in the first four characters and the terminal name in the last four. The new queue name is formed from the first four characters of the current queue name, followed by xxxx.

TOP
(Abbreviation: T)

Causes the CEBR transaction to start the display at the first record in the queue.

Using the CEBR transaction with transient data

The GET command reads each record in the transient data queue that you specify and writes it at the end of the temporary storage queue you are browsing, until the transient data queue is empty. You can then view the records that were in the transient data queue.

When you have finished your inspection, you can copy the temporary storage queue back to the transient data queue (using the PUT command). This typically leaves the transient data queue as you found it, but not always. Here are some points you need to be aware of when using the GET and PUT commands:

- If you want to restore the transient data queue unchanged after you have browsed it, make sure that the temporary storage queue on display at the time of the GET command is empty. Otherwise, the existing temporary storage records are copied to the transient data queue when the subsequent PUT command is issued.
- After you get a transient data queue and before you put it back, other tasks can write to that transient data queue. When you issue your PUT command, the records in the temporary storage queue are copied **after** the new records, so that the records in the queue are no longer in the order in which they were originally created. Some applications depend on sequential processing of the records in a queue.
- After you get a **recoverable** transient data queue, no other task can access that queue until your transaction ends. If you entered the CEBR transaction from the CEDF transaction, the CEDF transaction must end, although you can respond "yes" to the "continue" question if you are debugging a pseudoconversational sequence of transactions. If you invoked the CEBR transaction directly, you must end it.
- Likewise, after you issue a PUT command to a recoverable transient data queue, no other task can access that queue until your transaction ends.

The GET and PUT commands do not need to be used as a pair. You can add to a transient data queue from a temporary storage queue with a PUT command at any time. If you are debugging code that reads a transient data queue, you can create a queue in temporary storage (with the CECI transaction, or the CEBR GET command, or by program) and then refresh the transient data queue as many times as you like from temporary storage. Similarly, you can empty a transient data queue by using a GET command without a corresponding PUT command.

Command-level interpreter (CECI)

You can use the command-level interpreter (CECI) transaction to check the syntax of CICS commands and process these commands interactively on a 3270 screen. CECI allows you to follow through most of the commands to execution and display the results.

CECI also provides you with a reference to the syntax of the whole of the CICS command-level application programming and system programming interface.

CECI interacts with your test system to allow you to create or delete test data, temporary storage queues, or to deliberately introduce wrong data to test out error logic. You can also use CECI to repair corrupted database records on your production system.

The interpreter is such a powerful tool that your installation can restrict its use with attach-time security. (The external security manager used by your installation defines the security attributes for the CECI and CECS transactions.) If this has been done, and you are not authorized to use the interpreter transaction you select, you will not be able to initiate the transaction.

This chapter describes:

- “What does CECI display?” on page 657
- “Using CECI” on page 661
- “Saving commands” on page 663
- “How CECI runs” on page 664

What does CECI display?

All CECI screens have the same basic layout. CECI displays consist of the command line, the status line, the screen body, the message line, and the CECI option on function keys.

- "The command line" on page 657
- "The status line" on page 658
- "The screen body" on page 660
- "The message line" on page 660
- "CECI options on function keys" on page 661

The command line

The command line is the first line of the screen. You enter the command you want to process or whose syntax you want to check here. This can be the full or abbreviated syntax.

The rules for entering and abbreviating the command are:

- The keywords EXEC CICS are optional.
- The options of a command can be abbreviated to the number of characters sufficient to make them unique. Valid abbreviations are shown in uppercase characters in syntax displays in the body of the screen.
- The quotes around character strings are optional, and all strings of characters are treated as character-string constants unless they are preceded by an ampersand (&), in which case they are treated as variables.
- Options of a command that receive a value from CICS when the command is processed are called **receivers** , and need not be specified. The value received from CICS is included in the syntax display, and stored in the variable if one has been specified, after the command has been processed.
- If you issue a CECI command with two of the keywords in conflict, CECI ignores the first keyword and issues an error message, such as this one, from a READ command:

```
E INTO option conflicts with SET option and is ignored
```

- If you put a question mark in front of your command, the interpreter stops after the syntax check, even if you have used the transaction code CECI. If you want to proceed with execution, remove the question mark.

The following example shows the abbreviated form of a command. The file control command:

```
EXEC CICS READ FILE('FILEA') RIDFLD('009000')
INTO(&REC)
```

can be entered on the command input line, as:

```
READ FIL(FILEA) RID(009000)
```

or at a minimum, as:

```
READ F(FILEA) RI(009000)
```

In the first form, the INTO specification creates a variable, &REC, into which the data is to be read. However, INTO is a receiver (as defined above) and you can omit it. When you do, CICS creates a variable for you automatically.

The status line

As you go through the process of interpreting a command, CECI presents a sequence of displays. The format of the body of the screen is essentially the same for all; it shows the syntax of the command and the option values selected.

The status line on these screens tells you where you are in the processing of the command, and is one of:

- COMMAND SYNTAX CHECK
- ABOUT TO EXECUTE COMMAND
- COMMAND EXECUTION COMPLETE
- COMMAND NOT EXECUTED

From any of these screens, you can select additional displays. When you do, the body of the screen shows the information requested, and the status line identifies the display, which may be any of:

- EXPANDED AREA
- VARIABLES
- EXEC INTERFACE BLOCK
- SYNTAX MESSAGES

You can request them at any time during processing and then return to the command interpretation sequence.

There is also one input field in the status line called NAME=. This field is used to create and name variables.

Command syntax check

When the status line shows **command syntax check** , it indicates that the command entered on the command input line has been syntax checked but is not about to be processed.

This is always the status if you enter CECS or if you precede your command with a question mark. It is also the status when the syntax check of the command gives severe error messages.

In addition, you get this status if you attempt to execute one of the commands that the interpreter cannot execute. Although any command can be syntax-checked, using either CECS or CECI, the interpreter cannot process the following commands any further:

- EXEC CICS commands that depend upon an environment that the interpreter does not provide:
  - FREE
  - FREEMAIN
  - GETMAIN
  - HANDLE ABEND
  - HANDLE AID
  - HANDLE CONDITION
  - IGNORE CONDITION
  - POP HANDLE
  - PUSH HANDLE
  - SEND LAST
  - SEND PARTNSET
  - WAITCICS
  - WAIT EVENT
  - WAIT EXTERNAL
- BMS commands that refer to partitions (because the display cannot be restored after the screen is partitioned)

- EXEC DLI
- CPI Communication (CPI-C) commands
- SAA Resource Recovery interface (CPI-RR) commands

About to execute command

This example shows you the typical CECI display for about to execute command.

This display (as shown in Figure 159 on page 659) appears when none of the reasons for stopping at **command syntax check** applies.

```
 READ FILE('FILEA') RIDFLD('009000')
STATUS: ABOUT TO EXECUTE COMMAND NAME=
EXEC CICS READ
File( 'FILEA ' )
< SYsid() >
SEt() | Into()
< Length() >
RIdfld( '009000' )
< Keylength() < GEneric > >
< RBa | RRn | DEBRec | DEBKey >
< GTeq | Equal >
< Update < Token() > >









PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11
SF
```

Figure 159. Typical CECI display for about to execute command

If you press the ENTER key without changing the screen, CECI executes the command. You can still modify it, however. If you do, CECI ignores the previous command and processes the new one from scratch. This means that the next screen displayed is **command syntax check** if the command cannot be executed or else **about to execute command** if the command is correct.

CICS command completes

This example shows you the typical CECI display for command execution complete.

This display (as shown in Figure 160 on page 660) appears after the interpreter has run a command, in response to the ENTER key from an unmodified **about to execute command** screen.

```
 INQUIRE FILE NEXT
STATUS: COMMAND EXECUTION COMPLETE NAME=
EXEC CICS INquire File( 'DFHCSD ' )
< STArt | END | Next >
< ACcessmethod( +0000000003 ) >
< ADd( +0000000041 ) >
< BAsedsname( ' ' ) >
< BLOCKFormat( +0000000016 ) >
< BLOCKKeylen( -0000000001 ) >
< BLOCKSize( -0000000001 ) >
< BRowse( +0000000039 ) >
< Cfdtpool( ' ' )
< DElete( +0000000043 ) >
< DIsposition( +0000000027 ) >
< DSname( 'CFV01.CICS03.PSK.CSD ' ) >
< EMptystatus( +0000000032 ) >
< ENAblestatus( +0000000033 ) >
< EXclusive( +0000000001 ) >
< Fwdrecstatus( +0000000361 ) >
+ < Journalnum( +00000 ) >


RESPONSE: NORMAL EIBRESP=+0000000000 EIBRESP2=+0000000000
PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11
SF
```

Figure 160. Typical CECI display for command execution complete

The command has been processed and the results are displayed on the screen.

Any **receivers** , whether specified or not, together with their CICS-supplied values, are displayed intensified.

The screen body

The body of the CECI screens contains information that is common to all three displays.

The full syntax of the command is displayed. Options specified in the command line or assumed by default are intensified, to show that they are used in executing the command, as are any receivers. The < > brackets indicate that you can select an option from within these brackets. If you make an error in your syntax, CECI diagnoses it in the message area that follows the body, described in “The message line” on page 660. If there are too many diagnostic messages, the rest of the messages can be displayed using PF9.

Arguments can be displayed in either character or hexadecimal format. You can use PF2 to switch between formats. In character format, some characters are not displayable (including lowercase characters on some terminals); CECI shows them as periods. You need to switch to hexadecimal to show the real values, and you need to use caution when modifying them.

If the value of an option is too long for the line, only the first part is displayed followed by "..." to indicate there is more. You can display the full value by positioning the cursor at the start of the option value and pressing Enter. This action produces an expanded display.

If the command has more options than can fit on one screen, a plus sign (+) appears at the left side of the last option of the current display to indicate that there are more. An example of this is shown in “CICS command completes” on page 659. You can display additional pages by scrolling with the PF keys.

The message line

CECI uses the message line to display error messages. After execution of a command, the message line shows the response code.

The **S** that precedes the message indicates that it is severe (bad enough to prevent execution). There are also warning messages (flagged by **W**) and error messages (flagged by **E**), which provide information without preventing execution. **E** messages indicate option combinations unusual enough that they might not be intended and warrant a review of the command before you proceed with execution.

Where there are multiple error messages, CECI creates a separate display containing all of them, and uses the message line to tell you how many there are, and of what severity. You can get the message display with PF9.

Figure 160 on page 660 shows the second use of the message line, to show the result of executing a command. CECI provides the information in both text (NORMAL in the example in Figure 160 on page 660) and in decimal form (the EIBRESP and EIBRESP2 value).

CECI options on function keys

The single line at the foot of the screen provides a menu indicating the effect of the function keys for the display.

The function keys are described below. If the terminal has no function keys, the same effect can be obtained by positioning the cursor under the required item in the menu and pressing ENTER.

F1 HELP
displays a HELP panel giving more information on how to use the command interpreter and on the meanings of the function keys.

F2 HEX
(SWITCH HEX/CHAR) switches the display between hexadecimal and character format. This is a mode switch; all subsequent screens stay in the chosen mode until the next time this key is pressed.

F3 END
(END SESSION) ends the current session of the interpreter.

F4 EIB
(EIB DISPLAY) shows the contents of the EXEC interface block (EIB).

F5 VAR
(VARIABLES) shows all the variables associated with the current command interpreter session, giving the name, length, and value of each.

F6 USER
(USER DISPLAY) shows the current contents of the user display panel (that is, what would appear on the terminal if the commands processed thus far had been executed by an ordinary program rather than the interpreter). This key is not meaningful until a terminal command is executed, such as SEND MAP.

F7 SBH
(SCROLL BACK HALF) scrolls the body half a screen backward.

F8 SFH
(SCROLL FORWARD HALF) scrolls the body half a screen forward.

F9 MSG
(DISPLAY MESSAGES) shows all the messages generated during the syntax check of a command.

F10 SB
(SCROLL BACK) scrolls the body one full screen backward.

F11 SF
(SCROLL FORWARD) scrolls the body one full screen forward.

Using CECI

You start the command-level interpreter by entering either of two transaction identifiers, CECS or CECI, followed by the name of the command you want to test.

You can list command options too, although you can also do this later. For example:

```
CECS READ FILE('FILEA')
```

or

```
CECI READ FILE('FILEA')
```

CICS responds with a display of the command and its associated functions, options, and arguments. If you leave out the command, CECI provides a list of possible commands to get you started. You can use any of the commands described for programming purposes in the CICS command summary and System commands .

If you use the transaction code CECS, the interpreter checks your command for correct syntax. If you use CECI, you have the option of executing your command when the syntax is correct. CICS uses two transaction identifiers to allow different security to be assigned to syntax checking and execution.

Making changes

Until CICS executes a command, you can change it by changing the contents of the command line, by changing the option values shown in the syntax display in the body, or by changing the values of variables on the Variables screen. (You can still make changes after a command is executed, but, unless they are in preparation for another command, they have no effect.)

When you make your changes in the command line or on the Variables screen, they last for the duration of the CECI transaction. If you make them in the body of the syntax screen, however, they are temporary. They last only until the command is executed and are not reflected in the command line.

Not all characters are displayable on all terminals. When the display is in character rather than hexadecimal format, CECI shows these characters as periods (X'4B'). When you overtype a period, the current value might not be a period, but might be an undisplayable character.

Furthermore, you cannot change a character to a period when the display is in character mode. If you attempt this, CECI ignores your change, and does not issue a diagnostic message. To make such a change, you have to switch the display to hexadecimal and enter the value (X'4B') that represents a period.

There is a restriction on changes in hexadecimal format as well. If you need to change a character to a blank, you cannot enter the code (X'40') from a hexadecimal display. Again, your change is ignored and CECI does not issue a message. Instead, you must switch to character mode and blank out the character.

After every modification, CECI rechecks your syntax to ensure that no errors have appeared. It restarts processing at the `command syntax check` if there are any execution-stoppers, and at `about to execute command` if not. Only after you press Enter on an unmodified `about to execute command` screen does CECI execute your command.

Defining variables

When the command might exceed the line length of the command input area because of the values of the options, or you want to connect two commands through option values, you can define variables.

To display the list of variables, press PF5. For each variable associated with the current interpreter session, the name, length, and value are displayed. CECI creates the first three variables and they are always displayed, unless you explicitly delete them. They are intended to provide examples and help you create command lists.

After these first three variables, any variables that you have created are displayed. For example, you might enter the following command:

```
READ FILE('FILEA') RID('009000') INTO(&REC)
```

The &REC is displayed as a variable.

Typically, the value supplied for an option in the command line is taken as a character string constant. However, you can specify a variable to represent this value, for example when you want to connect two commands through option values. For example, to change a record with CECI, you might first enter the following command:

```
EXEC CICS READ UPDATE INTO(&REC)
FILE('FILEA') RID('009000')
```

You can then modify the record by changing the variable &REC, and then enter the following:

```
EXEC CICS REWRITE FROM(&REC) FILE('FILEA')
```

The ampersand in the first position tells CECI that you are specifying a variable.

You can create variables with the required values and specify the variable names in the command to overcome the line length limitation. Variables can have a data type of character, doubleword, fullword, halfword, or packed decimal. You can create variables in any of the following ways:

- Name the variable in a receiver. The variable is created when the command is processed. The data type and length are implied by the option.
- Add new entries to the list of variables already defined. To create a new variable, type its name and length in the appropriate columns on the first unused line of the variables display, and then press ENTER.
  - For character variables, use the length with which the variable has been defined.
  - For doublewords, type **FD**.
  - For fullwords, type **F**.
  - For halfwords, type **H**.
  - For packed variables with a length of 4 bytes, type **P**.
  - For packed variables with a length of 8 bytes, type **D**.

  Character variables are initialized to blanks. The others are initialized to zero in the appropriate form. After a variable is created, you can change the value by modifying the data field on the **variables** display.
- Use the NAME field on the status line when you have produced an expanded area display of a specific option. To do this, position the cursor under the option on a syntax display and press ENTER. To assign the variable name you want associated with the displayed option value, type it into the NAME field and press ENTER again.
- Copy an existing variable. To do this, obtain an expanded area display of the variable to copy, type over the name displayed with the name of the new variable, and press ENTER.
- Use the NAME field directly on a syntax display. This creates a character variable whose contents are the character string on the command line, for use in command lists, as explained in "Saving commands" on page 663.

You can also delete a variable, although CECI discards all variables at session end. To delete a variable before session end, position the cursor under the ampersand that starts the name, press ERASE EOF, and then press ENTER.

Saving commands

Sometimes you might want to execute a command, or a series of commands, under CECI, repeatedly. One technique for doing this is to create a temporary storage queue containing the commands. You then alternate reading the command from the queue and executing it.

CECI provides shortcuts both for creating the queue and for executing commands from it. To create the queue:

1. Start a CECI session.
2. Enter the first (or next) command you want to save on the command line, put &DFHC in the NAME field in the status line, and press ENTER. This action causes the typical syntax check, and it also stores your command as the value of &DFHC, which is the first of those three variables that CECI always defines for you. If you select the variables display, you see that &DFHC is the value of your command.
3. After the syntax is correct but before execution (on the **about to execute command** screen), change the command line to &DFHW and press ENTER. This causes CECI to use the value of &DFHW for the command to be executed. &DFHW is the second of the variables CECI supplies, and it contains a command to write the contents of variable &DFHC (that is, your command) to the temporary storage

queue named " CItttt" , where "tttt" is the name of your terminal and two blanks precede the letters "CI".

4. Execute this WRITEQ command (through the **command execution complete** screen). This stores your command on the queue.
5. If you want to save more than one command, repeat steps “2” on page 663 to “4” on page 664 for each.

When you want to execute the saved commands from the list, do the following:

1. Enter &DFHR on the command line and press ENTER. &DFHR is the last of the CECI-supplied variables, and it contains a command to read the queue that was written earlier. Execute this command; it brings the first (next) of the commands you saved into the variable &DFHC.
2. Then enter &DFHC on the command line and press ENTER. CECI replaces the command line with the value of &DFHC, which is your command. Press ENTER to execute your command.
3. Repeat these two steps, alternating &DFHR and &DFHC on the command line, until you have executed all the commands you saved.

You can vary this procedure to suit your needs. For example, you can skip commands in the sequence by skipping step (2). You can change the options of the saved command before executing it in the same way as a command entered normally.

If you want to repeat execution of the saved sequence, you need to specify the option ITEM(1) on the first execution of the READQ command, in order to reposition your read to the beginning of the queue.

How CECI runs

The interpreter runs as a conversational transaction, using programs supplied by CICS . Everything you do between the start of a session and the end is a single logical unit of work in a single task.

Locks and enqueues produced by commands that you execute remain for the duration of your session. If you read a record for update from a recoverable file, for example, that record is not available to any other task until you end CECI.

Abends

CECI runs all commands with the NOHANDLE option, so that execution errors do not ordinarily cause abends.

CECI also issues a **HANDLE ABEND** command at the beginning of the session, so that it does not lose control even if an abend occurs. Consequently, when you get one, CECI handles it and there is no resource backout. If you are doing a series of related updates to protected resources, make sure that you can complete all the updates. If you cannot complete all the updates, use a **SYNCPOINT ROLLBACK** command or an **ABEND** command with the CANCEL option to remove the effects of your earlier commands on recoverable resources.

Exception conditions

For some commands, CECI might return exception conditions even when all specified options are correct. These conditions return because, on some commands, CECI uses options that you do not specify explicitly. For example, the **ASSIGN** command always returns the exception condition INVREQ under CECI. Even though CECI might return the information you requested correctly, it attempts to get information from other options, some of which are invalid.

Program control commands

Because the interpreter is itself an application program, the interpretation of some program control commands might produce different results from an application program executing those commands. For example, an **ABEND** command is intercepted unless you use the CANCEL option.

If you use a **LINK** command, the target program runs in the environment of the interpreter. In particular, if you modify a user display during a linked-to program, the interpreter is not aware of the changes.

Similarly, if you interpret an **XCTL** command, CECI passes control to the named program and never gets control back, so the CECI session is ended.

Terminal sharing

When the command that is being interpreted is one that uses the same screen as the interpreter, the command interpreter manages the sharing of the screen between the interpreter display and the user display.

The user display is restored in the following situations:

- When the command that is being processed requires data from the operator
- When the command that is being processed is about to modify the user display
- When USER DISPLAY is requested

When a SEND command is followed by a RECEIVE command, the display sent by the SEND command appears twice; once when the SEND command is processed and again when the RECEIVE command is processed. You do not have to respond to the SEND command, but if you do, the interpreter stores and displays it when the screen is restored for the RECEIVE command.

When the interpreter restores the user display, it does not sound the alarm or affect the keyboard in the same way as when a SEND command is processed.

Shared storage: ENQ commands without LENGTH option

Normally, when you use the EXEC CICS ENQ command without the LENGTH option, the effect is to specify as the resource a data area with a specific location (address) in storage. Multiple tasks can enqueue on this resource and must refer to the same location in storage. CECI is not able to emulate this behavior, because it uses its own working storage, rather than shared storage.

If you execute an ENQ command in CECI without the LENGTH option, CICS enqueues on an address within storage owned by the CECI task. Other tasks, whether CECI or not, cannot enqueue on this same storage. CECI does not provide support for using shared storage for its variables.

It is not possible to emulate the intended behavior by specifying the storage address as the RESOURCE option, and adding the LENGTH option, when the ENQ command is executed in a CECI task, then specifying the same storage address without the LENGTH option in another CECI or non-CECI task. When the LENGTH option is specified, CICS enqueues on the value of the resource rather than on its location. CICS therefore regards the enqueues with and without the LENGTH option as different enqueues, and the tasks are not serialized as intended.

When the LENGTH option is specified for the same ENQ command issued from multiple tasks, the enqueue works as expected, because the location of the data area (whether in storage owned by CECI or in other storage) does not matter when the LENGTH option is specified.

Preparing to use debuggers with CICS applications

CICS supports the use of workstation-based and host-based debuggers for isolating and fixing bugs, and for testing applications. Before you can use a debugger with CICS applications, you must perform the following tasks.

Procedure

1. Choose between a workstation-based and host-based debugger.

   When you debug an application program, you interact with the program through the debugging tools. For example, you may want to examine storage, set breakpoints, or step through your code. This interaction is a *debugging session*. In CICS , you can choose the environment in which you conduct your debugging session:

Workstation-based
A *debugger client* on the workstation provides a graphical user interface which you use to perform the debugging tasks. The debugger client communicates with a *debugger server* which runs on your CICS system, and interacts with the program that is being debugged.

For more information, see "Debugging CICS applications from a workstation" on page 670.

Host-based
A debugging tool running in your CICS system provides a terminal interface which you use to perform the debugging tasks. The debugging tool interacts directly with the application as it executes.

CICS supports Debug Tool for host-based debugging. For more information, see "Using Debug Tool with CICS applications" on page 671.

Different application programs may have different debugging requirements (for example, Java programs cannot be debugged in a host-based debugging session). CICS lets different users use workstation-based and host-based debugging concurrently in the same region.

2. Ensure that your application programs will be intercepted by the debugging tool, and that others will not.

   Even in a test or development system, most of your application programs will function correctly most of the time. And when you are debugging, you will probably want to focus on one application at a time. At the same time, your colleagues might want to debug different applications. So you will need a way to specify those programs in your CICS system that are to interact with your debugging session, and those that are to interact with other users' debugging sessions, while letting most programs in the system run normally.

   Debugging profiles let you do all this. A debugging profile specifies a set of application programs which are to be debugged together. When you make a profile active, the programs it defines run under the control of the debugger, using a debugging session that you have specified. When you make the profile inactive, the programs run normally again, as do programs that are not referred to in debugging profiles. Debugging profiles also let you define the characteristics of the debugging session that you will use to debug a particular program.

   For more information, see "Debugging profiles" on page 667.

3. Prepare your programs for interacting with a debugger.

   CICS supports application programs written in a variety of languages. The compiled language programs (COBOL, PL/I, C, C++, and Language Environment-enabled Assembler subroutines) run under the control of Language Environment ; Java programs run in a Java virtual machine (JVM). Because there are, essentially, two different runtime environments for programs, there are two different ways to make your programs interact with the debugger.

   - For compiled language programs, you must decide when you compile your programs that you want them to interact with the debugger, and specify the appropriate compiler options. See the compiler documentation for more information.
   - For Java programs, you can decide at run time that you want them to interact with the debugger, and specify the appropriate JVM options. See JVM profile validation and properties CICS for more information.

4. Ensure that your CICS system is set up to support the debugging environment.

   When you have debugging profiles in your CICS system, there is an overhead in starting a program, even when all the profiles are inactive. This overhead, although small, is unlikely to be acceptable in a high-performance production system. In any case, you would not normally debug your applications in such a system. Therefore, the use of debugging profiles is optional, and if you want to use them, your system programmer will need to configure CICS accordingly.

Debugging profiles

A debugging profile specifies a set of one or more application programs which are to be debugged together.

For example:

- All instances of program PYRL01 running in system CICS1
- All Java classes with names beginning " `setBankAccount` "
- All programs with names beginning "'PYRL'" run by user APPDEV02

CICS uses the following information in the debugging profile to decide if an instance of a program should run under the debugger's control. The parameters specify:

- The transaction under which the program is running
- The terminal associated with the transaction. You can specify the terminal identifier or the z/OS Communications Server netname.
- The name of the program
- For COBOL programs, the name of the unit for compilation (the program or class name)
- For Java objects, the class name
- The userid of the signed-on user
- The applid of the CICS region in which the transaction is running

Many of the parameters can be generic, allowing you to specify a set of values which begin with the same characters (for example, TRN0, TRN1, TRN2, TRNA, TRNB, ...)

Debugging profiles contain the following additional information:

Status
The status of the profile: *active* or *inactive* :

- When a profile is active, it is examined each time a program is started in a region for which debugging is required.

  Note: If you change a profile while it is active, the changes take effect immediately: the next time a program is started, the changed parameters are used to decide if the program should run under the debugger's control.
- When a profile is inactive, it is ignored when a program is started.

Debugging display device settings
The debugging display device settings specify how you will interact with the debugger:

- For a Java program, you can use a debugging tool on a workstation
- For a compiled language program, you can use:

  A3270 terminal
  Adebugging tool on a workstation

The JVM profile name
For Java programs only, you can specify the JVM profile that will be used when a program is debugged

Debug Tool and Language Environment options
For compiled language programs only, you can specify options to be passed to Debug Tool and Language Environment when a program is debugged

You can create debugging profiles for the following sorts of program:

Compiled language programs
Java application programs

The information stored in the profile is different for each type of program.

Profiles are stored in a CICS file which can be shared by more than one CICS region. A profile that is shared by several CICS regions is either active or inactive in all the regions: it cannot be active in some regions and inactive in others.

CICS provides a set of sample profiles which are optionally generated when your system is set up to use debugging profiles. You can use these profiles as a starting point for creating your own profiles.

Using debugging profiles to select programs for debugging

To select a program for debugging, you must create one or more debugging profiles. Each profile specifies a number of parameters that CICS uses to decide if an instance of a program should run under the debugger's control.

Profiles can be active or inactive: if one of the active profiles matches the program instance, the program runs under the debugger's control. Inactive profiles are not examined when CICS starts a program. Profiles are inactive when they are created.

Table 60 on page 668 is an example which shows how parameters in the debugging profiles are used to select program instances for compiled language programs; Table 61 on page 668 shows how parameters in a debugging profile are used to select the program instance for a Java program.

Table 60. Examples of debugging profile parameters for compiled language programs

Debugging profile	Transaction	Terminal	Program	User	Applid
Profile 1	PRLA	T001	PYRL01	TESTER5	CICSTST2
Profile 2	PRLA	*	PYRL02	*	*
Profile 3	PRL*	*	*	*	CICSTST3

Table 61. Example of a debugging profile for a Java program

Debugging profile	Transaction	Bean	Method	User	Applid
Profile 4	PRLA	NewEmployee	setBasicSalary	TESTER5	CICSTST2

This is how each profile controls which programs run under the debugger's control:

Profile 1
In this example, all the parameters in the table are specified explicitly: Program PYRL01 will run under the debugger's control only if all these conditions are satisfied:

- The transaction is PRLA
- The transaction was started by terminal input from terminal T001
- The transaction is being run by user TESTER5
- The transaction is running in region CICSTST2

Profile 2
In this example, some of the parameters in the table are *generic parameters* , specified as * ; generic parameters of this type match all values. This profile specifies that every instance of program PYRL02 that runs under transaction PRLA will be under the debugger's control.

Profile 3
This example contains another sort of generic parameter: PRL* matches all values that start with the characters 'PRL'. This profile specifies that every program that runs under a transaction whose ID starts with the characters 'PRL' in region CICSTST3 will be under the debugger's control.

Profile 4
Method `setBasicSalary` will run under the debugger's control only if all these conditions are satisfied:

- The transaction is PRLA
- The method is a method of bean `NewEmployee`
- The transaction is being run by user TESTER5
- The transaction is running in region CICSTST2

You should choose the parameters that you specify in your debugging profiles with care, to ensure that programs do not start under the debugger's control unexpectedly:

- If you can do so, specify values for all, or most, of the parameters, to restrict debugging to particular programs in particular circumstances. Use specific values rather than generic values where possible.
- Whenever possible, specify the userid and applid explicitly in each debugging profile.
- Although it is inadvisable to debug programs in a production region, there may be times when you need to do so. On these occasions, use a debugging profile in which all the parameters are specified explicitly.
- Activate debugging profiles only when you need to use them, and inactivate them immediately after use.

Using generic parameters in debugging profiles

You can supply generic values for many of the parameters in your debugging profiles. To specify generic parameters, use an asterisk (*) as a *wildcard character*. You can use the wildcard character on its own, or at the end of a parameter. Leaving a parameter blank is equivalent to specifying an asterisk.

About this task

For example:

* matches all possible values
TR* matches TR, TRA, TRAA and TRAQ
TRA* matches TRA, TRAA and TRAQ, but not TR

When wildcards are used, a starting program may match more than one active profile. In this case, CICS selects the profile that is the best match, using the following principles:

- All parameters must match, either exactly, or when wildcards are considered.
- The best match is a profile that contains no wildcards.
- The next best matches are profiles that contain *. Within this grouping, the best matches are those that contain the smallest number of * characters, and the greatest number of explicitly specified characters.

For example, considering transaction TRAA:

- TRAA is the best possible match (all characters match)
- TRA* is a better match than TR*

It is advisable to avoid complex use of wildcards in your debugging profiles, as it is not always obvious which of many profiles will be the best match for a given program instance. However, should you need to do so, you can use the information in Figure 161 on page 670 to work out exactly which of several profiles will be the best match.

For each field in turn:

1. Count the number of characters (excluding * but including trailing blanks) for each field (*C*)
2. Count the number of * characters (*A*)
3. Determine the length of the field (*L*)
4. Calculate *M* as *C -(L * A)*. Note that *M* may be negative.

For each profile in turn, sum the values of *M* for all the fields (*R*).

The profile with the greatest value of *R* is the best match. If two or more matching profiles have the same greatest value of *R* , CICS chooses one of them, basing its selection on the sequence in which the profiles were created.

Figure 161. The debugging profile matching algorithm

Debugging CICS applications from a workstation

You can debug a CICS application using debugging tools that run on a workstation.

Components of the debugging tool

There are two components to the debugging tools in this environment:

Debugger client
The debugger client runs on the workstation and provides the graphical user interface (GUI) for you to interact with the application program. For example, you can use the debugger client to set breakpoints, to step through your program, and to examine the variables used by your program.

Debugger server
The debugger server runs on the same system as the application program, and communicates with the debugger client.

What CICS applications can be debugged

You can debug the following sorts of CICS applications using a debugger client on a workstation:

- Applications written in a compiled language (COBOL, PL/I, C, C++)
- Language Environment-enabled Assembler subroutines
- Java applications running in a JVM
- Applications that use a combination of compiled language programs and Java programs

You cannot debug PLT programs using a debugger client on a workstation.

Debugging tools that you can use

You can use IBM Developer for z Systems® as your debugger client.

For compiled languages and Language Environment-enabled Assembler subroutines, you can use Debug Tool for z/OS as your debugger server.

For Java programs, the debugger server is the Java Virtual Machine (JVM) executing in debug mode.

Preparing to debug applications from a workstation

Before you can debug CICS applications using a workstation, your system programmer must prepare your CICS region for debugging.

Procedure

1. Install a suitable debugger client such as IBM Developer for z Systems on your workstation.

   The documentation available with the debug product contains the information you need to install and use it.
2. Create one or more *debugging profiles*.

   A debugging profile specifies which programs will run under the debugger's control.

   Note: A debugging profile is not the same thing as a JVM profile. To debug a Java application, you need both profiles.
3. If you want to debug programs that are written in COBOL, PL/I, C or C++; or Language Environment-enabled Assembler subroutines, consider how you want to conduct your debug session, and compile your programs with the appropriate options. For more information, see Debug Tool for z/OS.
4. If you want to debug a Java program, you must ensure that it runs in a Java virtual machine (JVM) that is enabled for debugging.

   To do this:

   a) Create a JVM profile with parameters which enable the JVM for debugging. See Debugging a Java application for more information.

   b) Specify the JVM profile when you create a debugging profile for the Java program. If you do not specify the JVM profile, the JVM uses the profile specified in the PROGRAM definition.
5. Start the debugger client on your workstation.
6. If you are using WebSphere Studio as your debugger, set at least one breakpoint in your program.
7. Activate the debugging profiles that define the program instances that you want to debug. When you activate profiles for compiled language programs, you must define debugging options that specify the attributes of the debugging session that is started when the program runs.

Results

When you have completed all these steps, the programs that you have selected in the final step will run under the control of a debugger.

Using Debug Tool with CICS applications

Debug Tool helps you test programs and examine, monitor, and control the execution of application programs.

You can debug the following sorts of CICS applications using Debug Tool:

- Applications written in a compiled language (COBOL, PL/I, C, C++)
- Language Environment-enabled Assembler subroutines
- Applications that use a combination of compiled language programs and Java programs. Debug Tool does not debug the Java portions of these applications

You cannot debug PLT programs using Debug Tool.

You can use Debug Tool in four ways:

Single terminal mode
Debug Tool displays its screens on the same terminal as the application

Dual terminal mode
Debug Tool displays its screens on a different terminal than the one used by the application

Batch mode
Debug Tool does not have a terminal, but uses a commands file for input and writes output to a log

Remote debug mode
Debug Tool works with a debugger client to display results on a workstation

For more information about Debug Tool, see Debug Tool for z/OS.

Note: If you use Debug Tool in Single terminal mode, or Dual terminal mode, the terminal which is used by Debug Tool must be a local terminal in the region where the application is running: you cannot use a terminal on a terminal-owning region to interact with Debug Tool in an application-owning region.

Preparing to debug applications with Debug Tool

Before you can debug CICS applications using Debug Tool, your system programmer must prepare your CICS region for debugging.

Procedure

1. Consider how you want to conduct your debug session, and compile your programs with the appropriate options.

   For more information, see Debug Tool for z/OS.
2. Create one or more debugging profiles. A debugging profile specifies which programs will run under the debugger's control.
3. Activate the debugging profiles that define the program instances that you want to debug. When you activate the profiles, you must specify the display device with which you will interact with the debugger.

Results

When you have completed all these steps, the programs that you have selected in the final step will run under the control of Debug Tool.

Chapter 15. Deploying applications

You can deploy applications into CICS from CICS Explorer and application development products such as IBM Developer for z/OS. Deployment involves ensuring the code is in the right place, for example a data set or directory in zFS, and creating the resources in the target CICS regions to enable the application. You can also deploy services to support applications, such as business events, web services, and user exit programs. To deploy any application in CICS, you must have the correct access level to update directories and data sets in the z/OS system and install resources in CICS.

Deploying an application to a platform

To deploy an application to a platform, you export the CICS Application project and CICS Application Binding project from CICS Explorer to the platform home directory in zFS. Then you create and install an application definition (APPLDEF) in CICSPlex SM. Finally, you enable the application and make it available for use..

Before you begin

For an introduction to applications, see How it works: applications. This task assumes that you already:

- Decided how to structure your application. If not, see Designing an application for cloud enablement.
- Set up a platform onto which the application can be deployed. If not, see Setting up a platform.
- Created a CICS Application project and CICS Application Binding project in CICS Explorer. If not, see Packaging CICS applications for deployment in a cloud environment.

About this task

You export application artefacts from CICS Explorer to the platform home directory in zFS, and create an application definition. The application definition, which is an APPLDEF resource definition in the data repository for the CICSplex, identifies the location of the application bundle and the target platform where the application runs. You can create the application definition immediately following the export process, or you can create it at a later time.

The following steps outline the procedure. For detailed steps, see Working with applications in the CICS Explorer product documentation.

For information about automating application deployment without using CICS Explorer, see Automate the deployment and undeployment of CICS applications with the DFHDPLOY utility.

Use this procedure for the first installation of a particular application on a platform, when there are no other versions of the application installed on the platform. When you replace an existing version of an application with a new version, follow the procedure in Managing applications.

Procedure

1. Using the CICS Cloud perspective in CICS Explorer, export the CICS Application project from CICS Explorer to the platform home directory in zFS for the platform where you want to run the application.

   When you export the CICS Application project, CICS Explorer® also exports the Application Binding project and the CICS bundles associated with the application bundle and application binding to the home directory for the platform on zFS. Any CICS bundles that are already deployed to the platform home directory and installed in the CICSplex at the correct version are not included in the export. For more information about the platform directory structure, see Platform directory structure in z/OS UNIX.
2. Use the CICS Explorer to create an application definition (APPLDEF).

   This definition points to the location of the application bundle in the platform home directory on zFS and identifies the target platform for the application. You can choose to create an application

© Copyright IBM Corp. 1974, 2023

definition immediately after exporting your platform project, by checking the box in the application export wizard. To create your application definition at another time, use the New Application Definition wizard in the CICS Explorer.

CICSPlex SM creates an APPLCTN resource to represent the application in the CICSplex, and also creates a record for the application in the data repository, which is used in recovery processing for the bundles for applications. CICSPlex SM uses the information in the application bundle and the application binding to install the CICS bundles in the CICS regions in the platform. The application is initially installed in a disabled state.

When you install an application in an active platform, CICSPlex SM installs the CICS bundles immediately in all the CICS regions that are defined as part of the platform, and running at the time when you install the application definition. CICSPlex SM also installs the CICS bundles in CICS regions in the platform if you start or restart the regions after the time when you install the application definition. Resources in these CICS bundles will be installed at region start up but might not become fully enabled until after control is returned to CICS . If you add further CICS regions to the platform after the time when you install the application definition, CICSPlex SM installs the CICS bundles in those regions as well.

The relationship between the application and each installed CICS bundle is saved in a management part. The management part is a MGMTPART record that is created automatically for each CICS bundle during the application install process. It records the CICS regions where the bundle is installed, and tracks the status of the bundle in the CICS regions.

3. Using the Cloud Explorer view or the online application editor in the CICS Explorer, enable the installed application and check its status.

   An application is initially installed in a disabled state. When you enable an application, CICSPlex SM enables the CICS bundles that were installed for that application in all the CICS regions , but the application is not yet available to callers through its application entry points. If you start or restart a CICS region in the platform after the time when you enable the application, CICSPlex SM installs the bundles in that region in an enabled state.

4. When you are ready to make the application available to users of the platform, use the Cloud Explorer view or the application descriptor editor in CICS Explorer to make the application available.

   When you make an application available, CICS allows callers to access the application through the CICS resources that are declared as its application entry points. Callers can either access the highest available application version, or use the EXEC CICS INVOKE APPLICATION command to access a specific application version that is available. The availability status of an application is restored if you start or restart a CICS region in the platform after the time when you make the application available.

What to do next

You also add further quality of service by deploying policies to control the application. For more information, see CICS policies.

For more information about managing an application, see Administering platforms and applications.

Installing application programs

An application program generally means any user program that uses the CICS command-level application programming interface (API). To install an application program to run under CICS you must translate and compile your source statements, link-edit the resulting object modules into CICS libraries, and define the program to CICS as a resource.

Procedure

1. If your compiler does not translate CICS commands, you must translate the program source code to turn CICS commands into calls that are understood by the compiler.

   a. If your program does not use CICS commands and is invoked only by a running transaction (and never directly by CICS task initiation), you do not need a translator step.

b. You must also translate CICS command-level programs that access DL/I services through either the DL/I CALL or EXEC DLI interfaces. Applications that access Db2 services using the EXEC SQL interface need an additional precompilation step.

2. Compile your program source to produce object code.
3. Link-edit the object module to produce a load module, which you store in an application load library in the DFHRPL or dynamic LIBRARY concatenation.

   You need additional INCLUDE statements for applications that access Db2 services using the EXEC SQL interface.
4. Create resource definitions for any transaction that calls the program, and install them.
5. Define the program to CICS as a resource, using one of the following methods:

   - Create a resource definition in the CSD for a single CICS region. See An overview of resource definition.
   - Use program autoinstall, so that CICS dynamically generates a resource definition in a single CICS region when the program is loaded. See Autoinstalling programs, map sets, and partition sets.
   - Create a resource definition using CICSPlex SM Business Application Services (BAS). The resource definition can be installed in multiple CICS regions. See Administering BAS.
   - Create a resource definition in a standalone CICS bundle using the IBM CICS Explorer or IBM Developer for z/OS. The CICS bundle can be installed in multiple CICS regions. See Defining CICS bundles.
   - Create a resource definition in a CICS bundle, and package and install the CICS bundle as part of an application deployed on a platform. The application can be installed in multiple CICS regions, and you can install multiple versions of the application at the same time. See Deploying applications to a platform.

Program installation steps

There are a number of steps that need to be performed to install an application program to run under CICS.

About this task

The steps are as follows:

Procedure

1. If your compiler does not translate CICS commands, you must translate the program source code to turn CICS commands into calls that are understood by the compiler.

   a. If your program does not use CICS commands and is invoked only by a running transaction (and never directly by CICS task initiation), you do not need a translator step.

   b. You must also translate CICS command-level programs that access DL/I services through either the DL/I CALL or EXEC DLI interfaces. Applications that access Db2 services using the EXEC SQL interface need an additional precompilation step.
2. Compile your program source to produce object code.
3. Link-edit the object module to produce a load module, which you store in an application load library in the DFHRPL or dynamic LIBRARY concatenation.

   You need additional INCLUDE statements for applications that access Db2 services using the EXEC SQL interface.
4. Create resource definition entries, in the CSD for any transaction that calls the program, and install them.
5. Create a resource definition entry in the CSD for the program using one of the following methods:

   - Using program autoinstall.
   - Using RDO.

Using dynamic program LIBRARY resources

For an application to run, the load module has to reside in a data set in a CICS load LIBRARY concatenation.

CICS has two types of load LIBRARY concatenations:

- The static load LIBRARY concatenation: DFHRPL.
- One or more dynamically defined load LIBRARY concatenations.

Static LIBRARY concatenation DFHRPL
The startup JCL defines the static load LIBRARY concatenation, DFHRPL, to CICS. DFHRPL contains critical data sets, that must be available for CICS to start and run, and application program entities. When CICS is running, changes to the DFHRPL data set names are not possible without stopping and restarting CICS. Such changes are not usually an option in today's continuous availability environment.

DFHRPL data set names must conform to the MVS data set naming convention.

Data sets in a dynamic program LIBRARY concatenation can reside in the extended addressing space (EAS) of an extended address volume (EAV) DASD volume.

Dynamic program LIBRARY concatenation
You can define program LIBRARY concatenations to CICS dynamically. The use of dynamic program LIBRARY concatenations provides a number of advantages for the system programmer and the organization:

- They contain one or more data sets from which program artifacts can be loaded.
- You can bring new application programs for deployment into service at any time without affecting continuous availability.
- You can withdraw existing application programs in dynamic LIBRARY concatenations from service without affecting continuous availability.
- You can install patches to existing application programs by installing them in a LIBRARY concatenation with a higher ranking than the existing LIBRARY, without affecting continuous availability.
- You can take offline data sets in dynamic LIBRARY concatenations for compression without affecting continuous availability.

LIBRARY data set names must conform to the MVS data set naming convention and you can use alias data sets. Data sets in a dynamic program LIBRARY concatenation can reside in the extended addressing space (EAS) of an extended address volume (EAV) DASD volume.

You do not have to use dynamic program LIBRARY concatenations. You may use DFHRPL. In fact, you must define the following data sets in DFHRPL:

- SDFHLOAD
- Phase 1 PLT programs
- Non-SMS managed data sets
- Data sets with DISP other than SHR

You can install or create dynamic LIBRARY concatenations as either enabled or disabled.

Enabled
When you install or create a LIBRARY with an enabled status of enabled, CICS attempts to allocate and then concatenate the data sets, before finally opening the LIBRARY concatenation. If any of these steps fails, those that had already succeeded are undone and the LIBRARY is installed as disabled. A message indicates the step that has failed.

Disabled
When you install or create a LIBRARY with an enabled status of disabled, CICS does not attempt to allocate or concatenate the data sets. When the data sets are available and the LIBRARY is ready for

use, perform a SET LIBRARY ENABLED command to allocate and concatenate the data sets and open the LIBRARY.

If any of the enable steps in the SET LIBRARY ENABLED operation fails, those that had already succeeded are undone and the LIBRARY remains disabled. A message indicates the step that has failed.

You can use CICS Explorer or IBM Developer for z/OS to define dynamic LIBRARY concatenations in CICS bundles, for resource management and deployment in multiple CICS regions.

The following examples help you to use dynamic program LIBRARY concatenations.

Examples of using dynamic LIBRARY resources

You choose whether to adopt dynamic LIBRARY resources. You might use them in a test environment, or a production environment, or both. You may decide to move some data sets out of DFHRPL and define them dynamically and to use a combination of DFHRPL and dynamic program LIBRARY resources.

Good candidates for defining in a dynamic LIBRARY are vendor packages or in-house applications that are supplied in one or more data sets.

The following examples show how you can use dynamic program LIBRARY concatenations to manage your programs.

Applying an emergency fix to a CICS system

Install a temporary LIBRARY containing program fixes, to a CICS region.

About this task

- A version of an application being used by the CICS system has a problem that needs correcting.
- An updated version of the application has been created.
- You cannot restart CICS at this time to apply the fix.
- The person who performs this action has the appropriate access authority.

Procedure

1. Add programs and other artifacts which provide the fix to a PDS or PDSE data set, or set of data sets.
2. Define a LIBRARY resource that includes the data set, or data sets, containing the fixes.

   The LIBRARY resource must have a ranking that places it above the LIBRARY containing the failing version of the program in the search order. This might mean placing the LIBRARY before DFHRPL in the search order.

   Use one of the following methods to define the LIBRARY resource:

   - The Business Application Services (BAS) component of CICSPlex SM.
   - The CICS resource management transaction CEDA
   - The CICS CSD update batch utility program DFHCSDUP
   - The **EXEC CICS CREATE LIBRARY** command
   - CICS Explorer

   You can use the CICS Explorer to define a LIBRARY resource in a single CICS region, or in the data repository for the CICSplex, or in a CICS bundle. For information about the management of resources defined in CICS bundles, see Characteristics of resources in CICS bundles.
3. If you did not use EXEC CICS CREATE to define the LIBRARY, install the new LIBRARY resource using CEDA, the CICSPlex SM WUI, or the CICS Explorer.
4. Issue a EXEC CICS SET PROGRAM NEWCOPY or EXEC CICS SET PROGRAM PHASEIN command, or the equivalent instruction in CICSPlex SM or CEMT, for the affected program or programs.

Results

The CICS system continues to run and the corrected version of the application is now in the search order before the problem version so that the fixed version is used instead.

Installing a new application to a CICS system

Introduce a new application, which has been provided in one or more data sets, into a running CICS system without affecting continuous availability.

About this task

- The application has been provided in one or more PDS or PDSE data set. The application could be a third party (vendor) product that is provided as a set of application artifacts in one or more PDS or PDSE data set or a new in-house application.
- You cannot restart CICS at this time, to apply the fix.
- The person who performs the action has the appropriate access authorization.

Procedure

1. Define a LIBRARY resource that includes the data set or data sets containing the new application.

   Typically, the application will have no intersects with any existing LIBRARY resources and can use the default ranking value.

   Use one of the following methods to define the LIBRARY resource:

   - The Business Application Services (BAS) component of CICSPlex SM.
   - The CICS resource management transaction CEDA
   - The CICS CSD update batch utility program DFHCSDUP
   - The **EXEC CICS CREATE LIBRARY** command
   - CICS Explorer

   You can use the CICS Explorer to define a LIBRARY resource in a single CICS region, or in the data repository for the CICSplex, or in a CICS bundle. For information about the management of resources defined in CICS bundles, see Characteristics of resources in CICS bundles.
2. If you did not use **EXEC CICS CREATE** to define the LIBRARY, install the new LIBRARY resource using CEDA, the CICSPlex SM WUI, or CICS Explorer.
3. Define to CICS the programs, and map sets that make up the application and transaction definition or definitions that reference it.
4. Install the programs and other definitions.

Results

The new application is installed in the CICS production system and continuous availability is maintained.

Installing a new application to a set of CICS systems

Introduce a new application which has been provided in one or more data sets, into a set of CICS systems in a CICSplex. Such systems are more likely to be in production, although they could also be in a test or a development CICSplex.

About this task

- The application has been provided in one or more PDS or PDSE data sets.
- The application will be introduced to multiple CICS systems at the same time.
- You cannot restart the CICS regions at this time to add the new application or the application is not critical to the running of CICS.
- The person who performs this action has the appropriate access authorization.

Procedure

1. Define a CICSPlex SM LIBRARY definition (LIBDEF), that includes the application data set, using CICSPlex SM BAS.
2. Specify a ranking for the LIBRARY that reflects its ordering relative to other LIBRARY resources in use in the CICS regions. Typically, the application will have no intersects with any existing LIBRARY resources and can use the default ranking value.
3. Install the new LIBDEF, specifying a target scope that covers the set of CICS systems.
4. Define to CICS the programs, map sets and any other artifacts that make up the application and transaction definition or definitions, that reference it.
5. Install the programs and other definitions into the set of CICS systems and start to use them.

Results

The CICS regions are running with the new application.

Reorganizing CICS applications in the LIBRARY

You can reorganize applications in the LIBRARY to make them easier to track and manage. Applications stored in the LIBRARY are defined as DD card data sets in the DFHRPL concatenation.

Before you begin

Ensure that you have the required access authorization.

About this task

The goal of this task is to restructure the organization of applications into LIBRARY resources, so that the data set names are related to the applications they contain, not to operational suitability.

Procedure

1. Decide how to allocation the applications to LIBRARY data sets, and whether to use a single LIBRARY for each application, or multiple applications for each LIBRARY. If you use a single application for each LIBRARY the system configuration is simpler and easier to maintain. You must also determine which applications require multiple data sets to be concatenated in the LIBRARY, and which applications require single data sets.
2. Determine which applications remain in DFHRPL and which applications become dynamic resources.
3. Decide which of the applications to be defined in dynamic LIBRARYs are critical to CICS startup, and which applications are not critical to startup.
4. Define LIBRARY resources for each application, or for each set of applications if they are to be grouped, to be a dynamic resource, using the Business Application Services (BAS) component of CICSPlex SM, CEDA, DFHCSDUP, or EXEC CICS CREATE.
   a) Specify a ranking for each LIBRARY that represents its ordering relative to other libraries in the CICS region. Typically, the application does not intersect with other libraries, and can use the default ranking value.
   b) Specify a status for each LIBRARY that is critical to the running of CICS . Keep the default status, NONCRITICAL, for libraries that are not critical to the running of CICS.
   c) Specify the names of the data sets in the LIBRARY.
5. Install the new LIBRARY resources using either the CEDA INSTALL LIBRARY command, or the CICSPlex SM WUI.
6. Remove those data sets that contain applications in dynamic LIBRARY concatenations, from the DFHRPL concatenation on the next CICS restart.
7. Optionally, reset the ranking for each LIBRARY to its intended permanent value, if you set the ranking to a value that placed the LIBRARY before DFHRPL for testing purposes.

8. Install the new LIBRARY resources during CICS restart using a GRPLIST, during BAS installation, or after a CICS restart. When you do this, the system loads the programs from the new LIBRARY resources because they are now no longer in the DFHRPL concatenation.

Results

CICS runs normally. The LIBRARY applications are now better organized, and are easier to track. It is also easier to determine which CICS systems applications are installed on.

Taking a LIBRARY offline or removing an application from a CICS system

Take a LIBRARY offline; for example to compress a PDS or remove an application from a running CICS system.

About this task

- The application is in a known data set or set of data sets in a dynamic LIBRARY resource.
- The person who performs the action has the appropriate access authorization.

Procedure

1. Disable the LIBRARY using the EXEC CICS SET LIBRARY command, CEMT, the CICSPlex SM WUI, or the CICS Explorer.
2. When all uses of the application have completed, perform an operation to cause the loaded copies of the programs to be removed, for example a SET PROGRAM NEWCOPY, or let them fail at the next reload.
3. One reason for disabling the LIBRARY might be to compress the data set and then re-enable the LIBRARY or reinstall the LIBRARY definition to start using the application again. After re-enabling the LIBRARY, issue PROGRAM NEWCOPY or PHASEIN to use the programs again.

Results

While the LIBRARY is disabled, no new users can use the application, unless a copy exists in another LIBRARY that comes after the disabled LIBRARY in the search order, in which case it is loaded from there.

Switching between two LIBRARY concatenations

Introduce one LIBRARY to CICS and take another LIBRARY offline, so that a program in the new LIBRARY is loaded to replace that program in the old LIBRARY.

About this task

- A LIBRARY containing the program or various program artifacts making up an application is currently installed in CICS.
- A new version of the program or application is available in one or more PDS or PDSE data sets.
- The person who performs this action has the appropriate access authorization.

Procedure

1. Define a CICSPlex SM LIBRARY definition (LIBDEF) that includes the new application data set or data sets, using CICSPlex SM BAS.
2. Install the new LIBRARY.
3. Issue a PROGRAM NEWCOPY or PHASEIN command to start using the new copy of the program or programs.
4. Disable the old LIBRARY resource and discard it, if it is not likely to be used again.
5. Optionally, change the ranking of the new LIBRARY using the SET LIBRARY command, the WUI, or the CICSPlex SM API, back to that of the old LIBRARY.

Results

CICS runs with the new LIBRARY and new version of the application.

Discovering information about LIBRARY resources in a CICS system

Discover information about LIBRARY resources.

About this task

Discover information about LIBRARY resources, such as the following:

- Which LIBRARY resources are installed in CICS.
- The current search order through the active LIBRARY resources in CICS ; for example, installed and enabled LIBRARY resources.
- The relative positions of two LIBRARY concatenations in the search order.
- Which LIBRARY resources are critical.
- The data sets that are defined to a LIBRARY concatenation.

A dynamic program LIBRARY concatenation that is defined and installed as part of an application deployed on a platform is private to that version of that application. You can inquire on or browse private resources using the **EXEC CICS INQUIRE LIBRARY** system programming command. By default, CICS searches for the resources that are available to the program where the **EXEC CICS INQUIRE LIBRARY** command is issued. You can also choose to browse private resources for a specified application. For more information about browsing private resources, including examples of browsing in a different application context, see Browsing resource definitions.

Procedure

- In a running CICS system, issue the **EXEC CICS INQUIRE LIBRARY** command, or for public LIBRARY resources only, use the **INQUIRE LIBRARY** command in the CEMT transaction.

  If you do not specify a LIBRARY or properties, your inquiry shows all installed LIBRARY resources that are available to the program where the EXEC CICS INQUIRE command is issued. If you specify some properties of the LIBRARY resources, a subset of installed LIBRARY resources are shown. If you name a specific LIBRARY, details for that LIBRARY are shown. The detailed view of a LIBRARY shows the data sets in its concatenation.
- In the CICSPlex SM WUI, from the main menu, click **CICS operations** > **Program operations views** > **LIBRARYs, including DFHRPL**.

  Click the **Number of DSNAMEs** field name to display the LIBRARY data set names records.
- In the CICS Explorer , click **Operations** on the menu bar and select **LIBRARYs** , or select **LIBRARY DS Name** to display the LIBRARY data set names records.

  If you are connected to a CICS system, the resources displayed in the view are the resources in that system. If you are connected to a CICSplex, the resources displayed in the view depend on the browse scope you have set.
- For an application that is deployed on a platform, in the Cloud Explorer view in the CICS Explorer, double-click the installed application to open the application descriptor editor. Select the **Resources** tab, then select the **Libraries** tab to view the private LIBRARY resources for the application.

  You can filter the resources by the CICS system where they are installed, or by the CICS bundle where they are defined.

  Select the **Library DS Names** tab to view the DD names that z/OS has generated for the LIBRARY concatenation of data sets for the private LIBRARY resources.

Results

The inquiry shows the critical status and enablement status of the LIBRARY resources, their rankings, and also their absolute position in the overall search order. Disabled LIBRARY resources appear in the list

but do not participate in the search order. You can compare the search position numbers of two LIBRARY concatenations to determine which is before the other in the overall search order.

Discovering LIBRARY information for programs in a CICS system

Inquire on programs in the CICS system to discover the LIBRARY and data set in that LIBRARY from which the program was loaded; for example, to validate that it is loaded from the intended location.

Before you begin

- CICS is running.
- Programs are in use in the CICS system.

Procedure

Use the WUI or CEMT INQUIRE PROGRAM command to inquire on a program.

Results

The program information returned includes the LIBRARY and data set from which it was loaded:

- If the program was loaded from an installed LIBRARY, the LIBRARY and LIBRARYDSN names are returned.
- If the program was loaded from a LIBRARY that has been disabled, the LIBRARY name is returned but the LIBRARYDSN is blank.
- If the program was loaded from a LIBRARY that has been discarded, both LIBRARY and LIBRARYDSN are blank.
- If the program has not been loaded, both LIBRARY and LIBRARYDSN are blank.
- If the program was loaded from the LPA, both LIBRARY and LIBRARYDSN are blank.

Amending the CRITICAL property of LIBRARY resource

Specify that a LIBRARY in one or more CICS regions is critical to CICS startup.

Before you begin

- CICS is running.
- At least one dynamic LIBRARY is active in the CICS system.

Procedure

1. Use the CICSPlex SM WUI, CEMT, or the SPI to change the critical status of an installed LIBRARY. This change takes effect at the next warm or emergency start, when the critical status determines whether or not CICS startup continues uninterrupted if any of the data sets in the LIBRARY concatenation are unavailable or if any other problem prevents the LIBRARY from being recovered as enabled.
2. For a permanent change to the CRITICAL status, use CICSPlex SM BAS or CEDA to define LIBRARY definitions with the required critical status, and install the definitions at each CICS cold or initial start using CICSPlex SM BAS installation or GRPLIST installation.

Results

The CICS behavior on restart changes depending on the critical setting for the LIBRARY and the availability of the data sets in the LIBRARY.

Keeping track of changes to the LIBRARY configuration

Use the audit log to determine changes to the LIBRARY configuration in a CICS system

About this task

Messages DFHLD0555I and DFHLD0556I are written to the CSLB transient data queue, to provide an audit log of changes to the LIBRARY configuration. The messages are issued to show the current LIBRARY search order each time a change has occurred that might affect the search order. Message DFHLD0555I is issued followed by one instance of message DFHLD0556I for each LIBRARY that is active in the search order. Use the audit log to determine changes to the LIBRARY configuration in a CICS system, such as the following:

- A new LIBRARY is installed.
- A LIBRARY is removed (discarded) from CICS.
- The ranking, critical status, or enablement status of a LIBRARY is changed.
- The overall LIBRARY search order is changed.

Procedure

1. Examine the audit log, which is written to the CSLB transient data queue, to see changes to a LIBRARY configuration in a CICS system and the resulting LIBRARY search order.

   The audit log also shows changes to the LIBRARY configuration for LIBRARY concatenations that are defined as part of an application installed on a platform. These LIBRARY configurations, which are private to the application, are shown in their search order for the application, but they are not part of the search order for the CICS system.
2. Optionally, use a utility, developed in-house or by a vendor, to analyze and interpret the audit log for this system or for multiple systems.

Results

DFHLD0555I and DFHLD0556I show the new LIBRARY search order resulting after a change has occurred that might affect the search order. There is one instance of message DFHLD0556I for each LIBRARY that is active in the search order. There are a number of scenarios in which these messages will be displayed even though the search order is unchanged. For example, changing the ranking value of a disabled LIBRARY will not affect the search order because a disabled LIBRARY does not participate in the search order, but the change will trigger these messages. Similarly, changing the ranking of an enabled LIBRARY to a number that still lies between the rankings of the LIBRARYs on either side of it in the search order will trigger these messages, but not change the order.

Tidying up the LIBRARY configuration

Tidy up LIBRARY concatenations that have been used to apply temporary fixes or that contain application that are no longer used.

Before you begin

- CICS is running.

Procedure

1. Study the names of LIBRARY resources installed in CICS and the audit log of LIBRARY changes, to discover any LIBRARY resources used to apply temporary fixes that are no longer required, or LIBRARY resources for applications that are no longer used.
2. Discard any LIBRARY resources that are no longer required.
3. Delete the definitions of these LIBRARY resources, unless you believe that they will be required in the future and operational procedures allow reuse of LIBRARY definitions in this way.

Results

CICS continues as before. The set of LIBRARY resources installed in the CICS system are only those that are required for current applications used in the system.

Defining MVS residence and addressing modes

The effect of the MVS residence and addressing modes on application programs, how you can change the modes, and how you can make application programs permanently resident, are described.

About this task

A command-level program can reside above 16 MB, and can address areas above 2 GB.

Establishing the addressing mode of a program

Every program that executes in MVS is assigned two attributes: an addressing mode (AMODE), and a residency mode (RMODE).

About this task

The AMODE attribute specifies the addressing mode in which your program is designed to receive control. Generally, your program is designed to execute in that addressing mode, although you can switch modes in the program, and have different AMODE attributes for different entry points in a load module.

The RMODE attribute indicates where in virtual storage your program can reside.

Table 62. Valid AMODE and RMODE specifications

Attribute	Meaning
AMODE(24)	Specifies 24-bit addressing mode.
AMODE(31)	Specifies 31-bit addressing mode.
AMODE(ANY)	Specifies 24-bit or 31-bit addressing mode.
AMODE(64)	Specifies 64-bit addressing mode.
RMODE(24)	The module must reside in virtual storage below 16 MB. You can specify RMODE(24) for 31-bit or 64-bit programs that have 24-bit dependencies.
RMODE(ANY)	The module can reside in virtual storage below 16 MB, or above 16 MB but below 2 GB.

Note: C or C++ language programs must be link-edited with AMODE(31).

CICS does not support 64-bit residency mode (RMODE(64)) and treats any RMODE(64) programs as RMODE(31). That is, RMODE(64) programs are loaded into 31-bit (above-the-line) storage, not 64-bit (above-the-bar) storage.

If you do not specify any AMODE or RMODE attributes for your program, MVS assigns the system defaults AMODE(24) and RMODE(24). To override these defaults, you can specify AMODE and RMODE in one or more of the following places. Assignments in this list overwrite assignments later in the list.

Procedure

1. You can specify AMODE and RMODE on the link-edit MODE control statement:

   ```
   MODE AMODE(31),RMODE(ANY)
   ```

2. You can specify AMODE and RMODE by using one of the following methods:
   - In the PARM string on the EXEC statement of the link-edit job step:

```
//LKED EXEC PGM=IEWL,PARM='AMODE(31),RMODE(ANY),..'
```

- On the LINK TSO command, which causes processing equivalent to that of the EXEC statement in the link-edit step.

3. You can specify AMODE or RMODE statements in the source code of an assembler program.

   You can also set these modes in COBOL by using the compiler options. See the relevant application programming information for your COBOL compiler.

CICS address space considerations

The valid combinations of the AMODE and RMODE attributes, and their effects, are listed.

Table 63. Valid AMODE and RMODE specifications and their effects

AMODE	RMODE	Residence	Addressing
24	24	Below 16 MB	24-bit mode
31	24	Below 16 MB	31-bit mode
ANY	24	Below 16 MB	31-bit mode
31	ANY	Above 16 MB	31-bit mode
64	24	Below 16 MB	64-bit mode
64	ANY	Above 16 MB	64-bit mode

The following example shows link-edit control statements for a program coded to 31-bit standards, where *anyname* is the name of the load module:

```
//LKED.SYSIN DD *
MODE AMODE(31),RMODE(ANY)
NAME anyname(R)
/*
//
```

Making programs permanently resident

If you define a program in the CSD with the resident attribute, RESIDENT(YES), it is loaded on first reference. This applies to programs link-edited with either RMODE(ANY) or RMODE(24). However, be aware that the storage compression algorithm that CICS uses does not remove resident programs.

About this task

If there is not enough storage for a task to load a program, the task is suspended until enough storage becomes available. If any of the DSAs get close to being short on storage, CICS frees the storage occupied by programs that are not in use. For more information about the dynamic storage areas in CICS, see CICS dynamic storage areas.

Instead of making RMODE(24) programs resident, you can make them non-resident and use the library lookaside (LLA) function. The space occupied by such a program is freed when its usage count reaches zero, making more virtual storage available. LLA keeps its library directory in storage and stages (places) copies of LLA-managed library modules in a data space that the virtual lookaside facility (VLF) manages. CICS locates a program module from the LLA library directory in storage, rather than searching program directories on DASD. When CICS requests a staged module, LLA gets it from storage without any input/output activity.

Running applications in the link pack area

Programs can reside in the link pack area (LPA) provided that they follow certain requirements.

Assembler language, C, COBOL, or PL/I programs must be read-only and adhere to the requirements as follows:

Assembler
Use the RENT assembler option.

C
Use the RENT compiler option.

COBOL
Do not overwrite WORKING STORAGE. (The CICS translator generates a CBL statement with the required compiler RENT option (unless you specify the translator option NOCBLCARD).

PL/I
Do not overwrite STATIC storage. (The CICS translator inserts the required REENTRANT option into the PROCEDURE statement.)

All programs must be link-edited with the RENT and REFR options.

If you want CICS to use modules that you have written to these standards, and installed in the LPA, specify USELPACOPY(YES) on the program resource definitions in the CSD.

Running application programs in the read-only DSAs

Programs that are eligible to reside above 16 MB, and are read-only, can reside in the CICS extended read-only DSA (ERDSA). Programs that are *not* eligible to reside above 16 MB, and are read-only, can reside in the CICS read-only DSA (RDSA) below 16 MB.

The following conditions apply for programs to be eligible for the ERDSA:

- Programs must be properly written to read-only standards.
- Programs must be written to 31-bit addressing standards.
- Programs must be link-edited with the RENT attribute and the RMODE(ANY) residency attribute.

The following conditions apply for programs to be eligible for the RDSA:

- Programs must be properly written to read-only standards.
- Programs must be link-edited with the RENT attribute and the RMODE(24) residency attribute.

Note: When you run CICS with RENTPGM=PROTECT specified as a system initialization parameter, the RDSAs are allocated from key-0 read-only storage.

ERDSA requirements for the specific languages are described in the following topics.

Assembler

For CICS to load your assembler programs in the ERDSA, assemble and link-edit them with the following options: the RENT assembler option; the link-edit RENT attribute; the RMODE(ANY) residency mode.

If you specify these options, ensure that the program is truly read-only (for example, it does not write to static storage), otherwise storage exceptions occur. The program must also be written to 31-bit or 64-bit addressing standards. For possible causes of storage protection exceptions in programs that are resident in the ERDSA, see Causes of protection exceptions.

AMODE(31) programs

The CICS-supplied procedure for AMODE (31) application programs, DFHEITAL, has a LNKPARM parameter that specifies the XREF and LIST options only. To link-edit an ERDSA-eligible program, override LNKPARM from the calling job, specifying the RENT and RMODE(ANY) options in addition to any others you require.

For example:

```
//ASMPROG JOB
1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//EITAL EXEC DFHEITAL,
    .
    (other parameters as necessary)
    .
// LNKPARM='LIST,XREF,RMODE(ANY),RENT'
```

The CICS EXEC interface module for assembler programs (DFHEAI) specifies AMODE(ANY) and RMODE(ANY). However, because the assembler defaults your application to AMODE(24) and RMODE(24), the resulting load module also becomes AMODE(24) and RMODE(24).

If you want your application program link-edited as AMODE(31) and RMODE(ANY), use appropriate statements in your assembler program. For example:

```
MYPROG CSECT
MYPROG AMODE 31
MYPROG RMODE ANY
```

You can set AMODE and RMODE in the following ways:

- You can set the required AMODE and RMODE specification by using link-edit (or binder) control information in the JCL PARM keyword. For example:

  ```
  //EITAL EXEC DFHEITAL,
  LNKPARM='LIST,XREF,RENT,AMODE(31),RMODE(ANY)'
  ```

- Alternatively, you can use the MODE control statement in the SYSIN data set in the link-edit, or the binder, step in your JCL.

When you use the binder, you might see unexpected warning messages about conflicting AMODE and RMODE specifications.

AMODE(64) programs

The CICS-supplied procedure for AMODE(64) application programs, DFHEGTAL, has a LNKPARM parameter that specifies the XREF and LIST options only. To link-edit an ERDSA-eligible program, override LNKPARM from the calling job, specifying the RENT and RMODE(ANY) options in addition to any others you require.

For example:

```
//ASMPROG JOB
1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//EGTAL EXEC DFHEGTAL,
    .
    (other parameters as necessary)
    .
// LNKPARM='LIST,XREF,RMODE(ANY),RENT'
```

The CICS EXEC interface module for AMODE(64) assembler programs (DFHEAG) specifies AMODE(64) and RMODE(ANY). However, because the assembler defaults your application to AMODE(24) and RMODE(24), the resulting load module also becomes AMODE(24) and RMODE(24).

If you want your application program link-edited as AMODE(64) and RMODE(ANY), use appropriate statements in your assembler program. For example:

```
MYPROG CSECT
MYPROG AMODE 64
MYPROG RMODE ANY
```

You can set AMODE and RMODE in the following ways:

- You can set the required AMODE and RMODE specification by using link-edit (or binder) control information in the JCL PARM keyword. For example:

```
//EITAL EXEC DFHEGTAL,
LNKPARM='LIST,XREF,RENT,AMODE(64),RMODE(ANY)'
```

- Alternatively, you can use the MODE control statement in the SYSIN data set in the link-edit, or the binder, step in your JCL.

When you use the binder, you might see unexpected warning messages about conflicting AMODE and RMODE specifications.

C and C/++

If you want CICS to load your C and C++ programs into the ERDSA, compile and link-edit them with the RENT compiler option.

The CICS-supplied procedures DFHYITDL or DFHYITFL (for C) and DFHYITEL or DFHYITGL (for C++) have a LNKPARM parameter that specifies a number of link-edit options. To link edit an ERDSA-eligible program, override this parameter from the calling job, and add RENT to the other options you require. You do not need to add the RMODE(ANY) option, because the CICS EXEC interface module for C (DFHELII) is link-edited with AMODE(31) and RMODE(ANY). Therefore, your program is link-edited as AMODE(31) and RMODE(ANY) automatically when you include the CICS EXEC interface stub, see "The CICS-supplied interface modules" on page 594.

The following sample job statements show the LNKPARM parameter with the RENT option added:

```
//CPROG JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//YITDL EXEC DFHYITDL,
    .
    (other parameters as necessary)
    .
// LNKPARM='LIST,MAP,LET,XREF,RENT'
```

If you want to compile C or C++ code that contains sequence numbers, for example the C or C++ sample programs distributed with CICS, then you must override the **CPARM** parameter to specify SEQ. For example:

```
//EXAMPLE EXEC DFHYITEL,
// CPARM='/CXX OPT(1) SEQ NOMAR SOURCE'
//TRN.SYSIN DD *
... source code goes here ...
/*
//LKED.SYSIN DD *
NAME EXAMPLE(R)
/*
```

COBOL

If you use the integrated CICS translator, the compilation requires the RENT compiler option, so no CBL card needs to be added during translation.

COBOL programs that use a separate translation step are automatically eligible for the ERDSA, for the following reasons:

- The CBLCARD translator option (the default) causes the required compiler option, RENT, to be included automatically on the CBL statement generated by the CICS translator. If you use the NOCBLCARD translator option, you can specify the RENT option either on the PARM statement of the compilation job step, or by using the COBOL macro IGYCOPT to set installation-defined options.
- The COBOL compiler automatically generates code that conforms to read-only and 31-bit addressing standards.
- The CICS EXEC interface module for COBOL (DFHELII) is link-edited with AMODE(31) and RMODE(ANY). Therefore, your program is link-edited as AMODE(31) and RMODE(ANY) automatically when you include the CICS EXEC interface stub. See "The CICS-supplied interface modules" on page 594.

You also need to specify the reentrant attribute to link-edit. The CICS-supplied procedure DFHYITVL has a LNKPARM parameter that specifies a number of link-edit options. To link-edit an ERDSA-eligible program, override this parameter from the calling job, and add RENT to any other options you require. For example:

```
//COBPROG JOB
1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//YITVL EXEC DFHYITVL,
    .
    (other parameters as necessary)
    .
// LNKPARM='LIST,XREF,RENT'
```

PL/I

CICS PL/I programs are generally eligible for the ERDSA, provided that they do not change static storage.

The following requirements are enforced, either by CICS or PL/I:

- The PL/I compilation procedure must specify the RENT compiler option. The CICS-supplied procedures for compiling PL/I, for example DFHYITPL, automatically include this option.
- The PL/I compiler automatically generates code that conforms to 31-bit addressing standards.
- The CICS EXEC interface module for PL/I (DFHELII) is link-edited with AMODE(31) and RMODE(ANY). Therefore, your program is link-edited as AMODE(31) and RMODE(ANY) automatically when you include the CICS EXEC interface stub. See "The CICS-supplied interface modules" on page 594.

You also need to specify the reentrant attribute to the link edit. The CICS-supplied procedure DFHYITPL has a LNKPARM parameter that specifies a number of link-edit options. To link edit an ERDSA-eligible program, override this parameter from the calling job, and add RENT to any other options you require. For example:

```
//PLIPROG JOB
1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//YITPL EXEC DFHYITPL,
    .
    (other parameters as necessary)
    .
// LNKPARM='LIST,XREF,RENT'
```

Do not specify the RENT attribute on the link-edit step unless you have ensured the program is truly read-only (for example, it does not write to static storage), otherwise storage exceptions occur. For possible causes of storage protection exceptions in programs that are resident in the ERDSA, see Causes of protection exceptions.

Using BMS map sets in application programs

The procedure to use BMS map sets in application programs is summarized.

About this task

Use the following procedure before you install an application program to run under CICS.

Procedure

- Create any BMS map sets used by the program, as described in "Installing map sets and partition sets" on page 706.
- Include the physical map sets (used by BMS in its formatting activities) in a data set that is in the DFHRPL or dynamic LIBRARY concatenation.
- Either include the symbolic map sets (copied into the application programs) in a user copy library, or insert them directly into the application program source.

  The DFHMAPS procedure writes the symbolic map set output to the library specified on the DSCTLIB parameter, which defaults to the CICSTS56.CICS.SDFHMAC library. To include symbolic map sets in a user copy library, use the following steps:

a) Specify the library name by the DSCTLIB= *name* operand on the EXEC statement for the DFHMAPS procedure used to install physical and symbolic map sets together.

b) Include a DD statement for the user copy library in the SYSLIB concatenation of the job stream used to assemble and compile the application program.

   If you choose to let the DFHMAPS procedure write the symbolic map sets to the CICSTS56.CICS.SDFHMAC library (the default), include a DD statement for the CICSTS56.CICS.SDFHMAC library in the SYSLIB concatenation of the job stream used to compile the application program. This is not necessary for the DFHEITAL procedure used to assemble assembler language programs, because these jobs already include a DD statement for the CICSTS56.CICS.SDFHMAC library in the SYSLIB concatenation.

c) For PL/I, specify a library that has a block size of 32760 bytes. This is necessary to overcome the blocksize restriction on the PL/I compiler.

What to do next

For more information about installing map sets, see "Installing map sets and partition sets" on page 706. For information about writing programs to use BMS services, see "Basic mapping support" on page 366.

Using the CICS-supplied procedures to install application programs

CICS supplies job control statements (JCL) for translate (if required), compile, and link-edit steps, in separate cataloged procedures for each supported programming language.

After CICS is installed, copy these procedures, which are installed in the CICSTS56.CICS.SDFHPROC library, into a procedure library. The name of each procedure is in the form DFH *wx* T *y* L, where the variables *w* , *x* , and *y* depend on the type of program (EXCI batch or CICS online), the type of compiler, and the programming language. The following tables show the procedure names.

Table 64. Procedures to install application programs: non-Language Environment-conforming compilers

Language	**Stand-alone translator**	**EXCI**
Assembler	DFHEITAL (AMODE(24) and AMODE(31) applications) DFHEGTAL (AMODE(64) applications)	DFHEXTAL

Table 65. Procedures for installing application programs: Language Environment-conforming compilers

Language	**Stand-alone translator**	**Integrated translator**	**EXCI with Stand-alone translator**	**EXCI with integrated translator**
C	DFHYITDL (see note "1" on page 691)	DFHZITDL (see note "1" on page 691)	DFHYXTDL	DFHZXTDL
C using the XPLINK compiler option	DFHYITFL (see note "2" on page 691)	DFHZITFL (see note "1" on page 691)	-	-
C++	DFHYITEL (see note "1" on page 691)	DFHZITEL (see note "1" on page 691)	DFHYXTEL	DFHZXTEL
C++ using the XPLINK compiler option	DFHYITGL (see note "2" on page 691)	DFHZITGL (see note "1" on page 691)	-	-

Table 65. Procedures for installing application programs: Language Environment-conforming compilers (continued)

Language	Stand-alone translator	Integrated translator	EXCI with Stand-alone translator	EXCI with integrated translator
COBOL (see note "3" on page 691)	DFHYITVL	DFHZITCL (see note "2" on page 691)	DFHYXTVL	DFHZXTCL
PL/I (see note "4" on page 691)	DFHYITPL (see note "2" on page 691)	DFHZITPL (see note "2" on page 691)	DFHYXTPL	DFHZXTPL

Notes:

1. DFHYITEL can also be used for C; you must specify the correct name of the C compiler on the **COMPILER** parameter.
2. The output library for the generated module is a PDSE (not a PDS).
3. DFHZITCL is the recommended procedure for compiling COBOL modules because it uses the version of the Enterprise COBOL compiler that includes the integrated CICS translator.
4. DFHZITPL is the recommended procedure for compiling PL/I modules because it uses the version of the Enterprise PL/I compiler that includes the integrated CICS translator.
5. For programs that issue EXEC DLI commands in a batch environment under Language Environment (IMS routines), use these special procedures:

   DFHYBTPL
   PL/I application programs

   DFHYBTVL
   COBOL application programs.

   Note: This procedure requires the macro DFHLI000.

Installing programs in load library secondary extents

CICS supports load library secondary extents that are created while CICS is running. If you define libraries in the DFHRPL or dynamic LIBRARY concatenation with primary and secondary extents, and secondary extents are added as a result of link-editing into the load library while CICS is running, the CICS loader detects the occurrence, closes, and then reopens the library. This means that you can introduce new versions using the CEMT NEWCOPY command, even if the new copy of the program has caused a new library extent.

Note: If you are using DFHXITPL, the SYSLMOD DD statement in the binder step must refer to a PDSE (not a PDS as for the older PL/I compilers).

Including the CICS-supplied interface modules

If you want to use CPI Communications or SAA® Resource Recovery in your application program then you must make the appropriate interface modules available to your program.

The CICS -supplied procedures to install your online application programs in a CICS library specify the CICS library member that contains the INCLUDE statement for the appropriate language EXEC interface module. For example, the DFHYITVL procedure uses the following statements:

```
//COPYLINK EXEC PGM=IEBGENER,COND=(7,LT,COB)
//SYSUT1 DD DSN=&INDEX..SDFHSAMP(&STUB),DISP=SHR
//SYSUT2 DD DSN=&&COPYLINK,DISP=(NEW,PASS),
// DCB=(LRECL=80,BLKSIZE=400,RECFM=FB),
// UNIT=&WORK,SPACE=(400,(20,20))
//SYSPRINT DD SYSOUT=&OUTC
//SYSIN DD DUMMY
```

```
//SYSLIN DD DSN=&&COPYLINK,DISP=(OLD,DELETE)
// DD DSN=&&LOADSET,DISP=(OLD,DELETE)
// DD DDNAME=SYSIN
```

In this COBOL example, the symbolic parameter STUB defaults to DFHEILID. The DFHEILID member contains the statement INCLUDE SYSLIB(DFHELII).

The supplied procedures for PL/I and C also refer to DFHEILID, which means that the DFHELII stub is used.

If your application program is to use CPI Communications or the SAA Resource Recovery facility, do one of the following:

- Add appropriate INCLUDE statements to the LKED.SYSIN override in the job used to call the CICS-supplied procedure to install your application program. Add the following INCLUDE statements:
  - INCLUDE SYSLIB(DFHCPLC) if your program uses CPI Communications
  - INCLUDE SYSLIB(DFHCPLRR) if your program uses SAA Resource Recovery

Warning messages can appear during the link-edit step, indicating DUPLICATE definitions for the DFHEI1 entry. You can ignore these messages.

For more information about link-edit requirements, see “Using your own job streams” on page 703.

Translating, assembling, and link-editing assembler language application programs

You can use the DFHEITAL or DFHEXTAL procedure to translate, assemble, and link-edit AMODE(24) and AMODE(31) application programs written in assembler language. You can use the DFHEGTAL procedure to translate, assemble, and link-edit AMODE(64) application programs written in assembler language.

About this task

You can use the sample job control statements shown in Figure 162 on page 692 to process application programs written in assembler language. In the procedure name, *x* depends on whether your programs are CICS application programs and the AMODE of those programs, or EXCI batch programs. For the names of the CICS-supplied procedures, see “Using the CICS-supplied procedures to install application programs” on page 690.

```
//jobname      JOB     accounting info,name,MSGLEVEL=1
//             EXEC    PROC=DFHExTAL                        1
//TRN.SYSIN    DD      *
*ASM      XOPTS(translator options . . .)                 2
               .
               assembler language source statements
               .
/*
//LKED.SYSIN   DD      *
               NAME    anyname(R)
/*
//
```

Figure 162. Sample job control statements to call the DFHExTAL procedures

anyname is the name of your load module.

Notes:

1. To install a program into a read-only DSA, see “Running application programs in the read-only DSAs” on page 686 for more details.

   To install a program to use from the LPA, add the following options:

   - RENT to the PARM options in the EXEC statement for the ASM step of the DFHE *x* TAL procedure
   - RENT and REFR options to the LNKPARM parameter on the call to the DFHE *x* TAL procedure

(See “Running applications in the link pack area” on page 686.)

2. For information about the translator options you can include on the XOPTS statement, see “Defining translator options” on page 592.

The following example shows the job control statements to translate, assemble and link-edit an AMODE(64) application program by using the CICS-supplied procedure DFHEGTAL.

```
//APPLPROG EXEC DFHEGTAL
//TRN.SYSIN  DD *
   .
   . Application program
   .
/*
//LKED.SYSIN DD *
   ENTRY program_name
   NAME program_name(R)
/*
```

program_name is the name of the AMODE(64) application.

Figure 163 on page 694 shows the Assembler source program processed by the command level translator, with reference to CICS.SDFHLOAD, to produce a translator listing and an output file. This output file is then processed by the Assembler, with reference to CICS.SDFHMAC, to produce an assembler listing and a further output file. This output file is then processed by the linkage editor to produce a linkage editor listing and a load module that is stored in an application library.

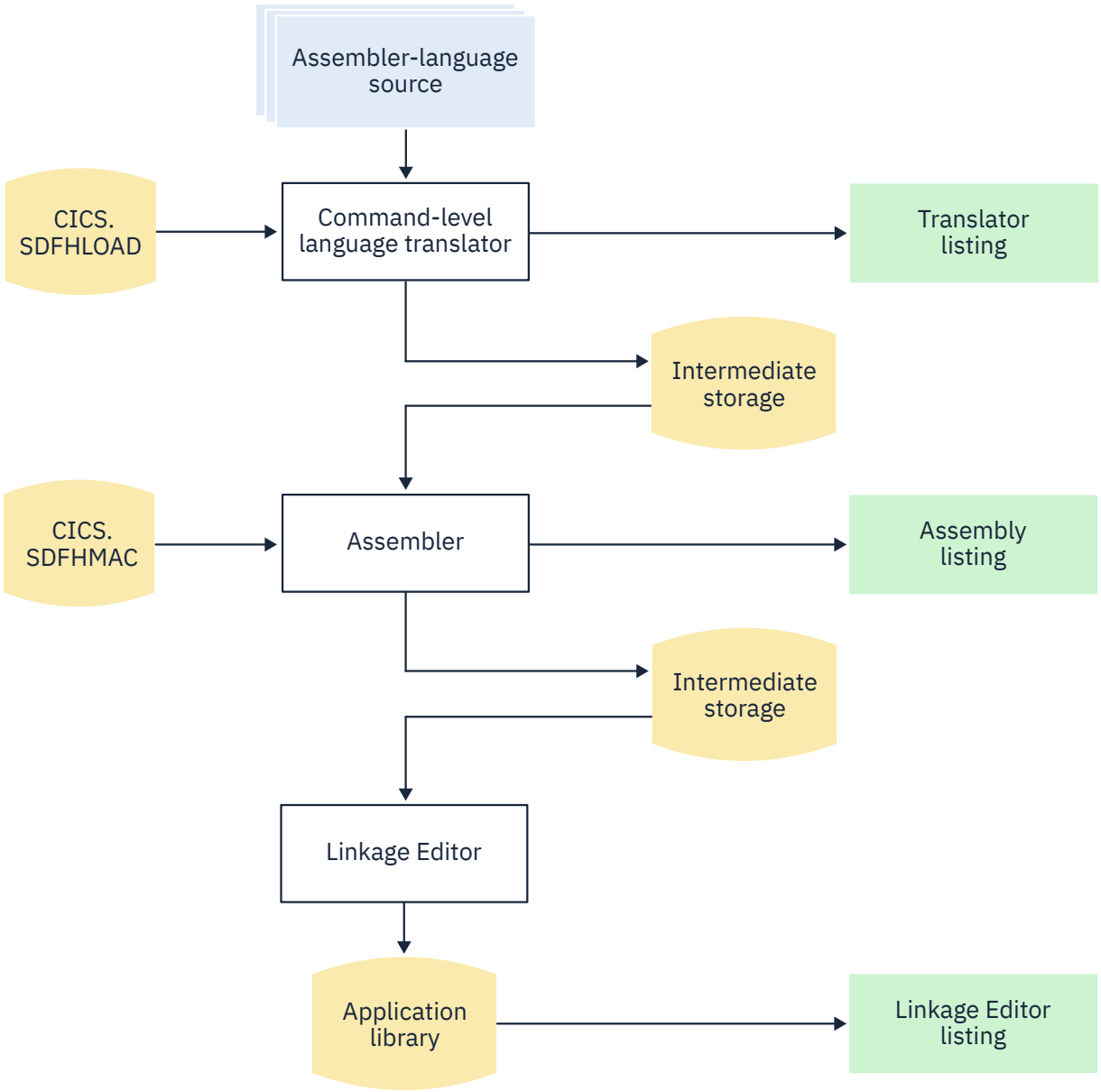


Figure 163. Installing assembler language programs using the DFHEITAL or DFHEGTAL procedure

Installing COBOL application programs

This diagram shows you the flow of control in the cataloged procedures for COBOL and PL/I programs that require a separate translator step. If you use an integrated translator, there is no separate translator step. The high-level language source and CICS.SDFHLOAD both input to the compiler, and a combined translator and compiler listing is produced.

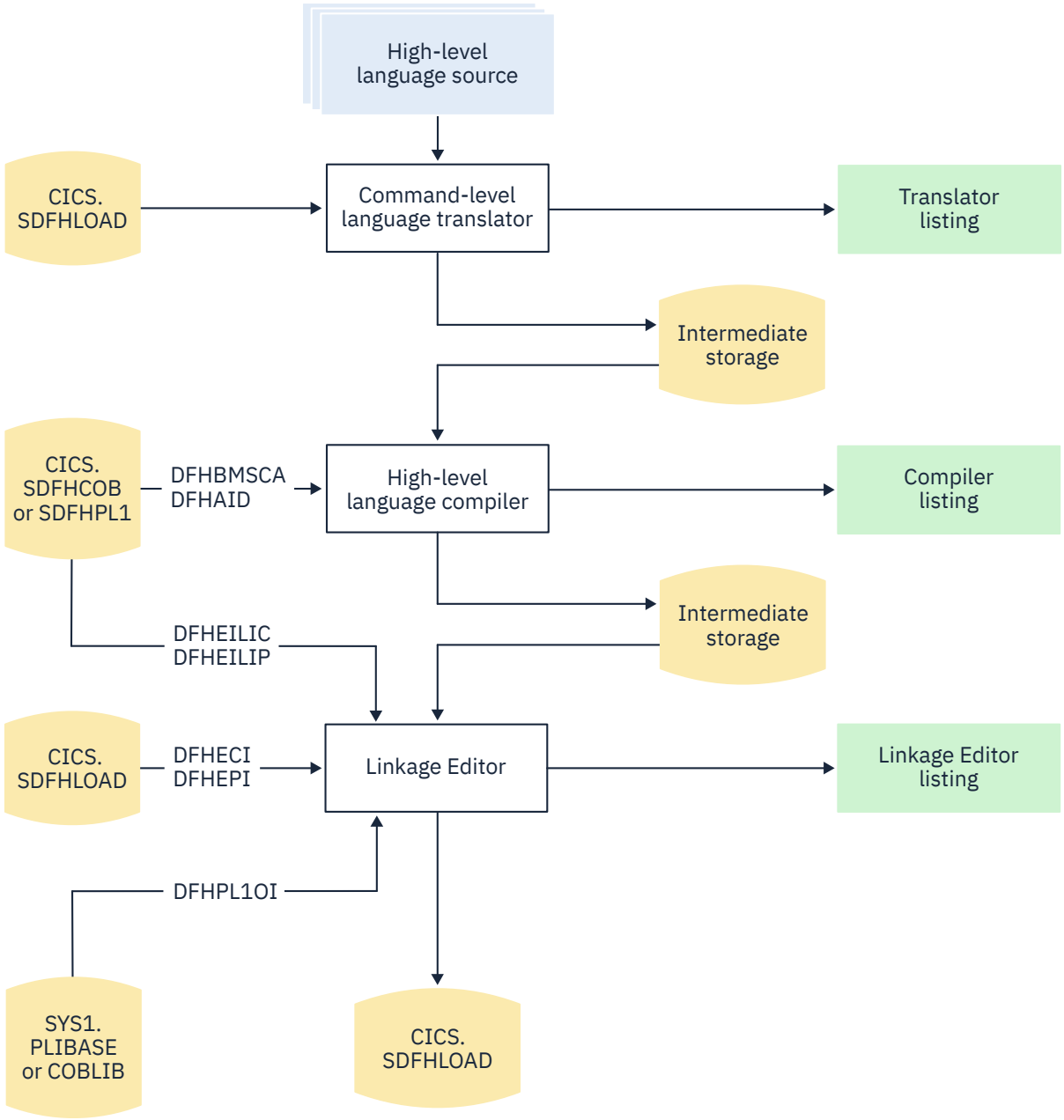


Figure 164. Installing COBOL and PL/I programs

Sample JCL to install COBOL application programs

You can use the job control statements that are shown in these examples to process COBOL application programs with a separate or integrated translator.

The procedure name depends on whether it is a CICS application program or an EXCI batch program. For the names of the COBOL procedures supplied with CICS, see Table 65 on page 690.

```
//jobname JOB accounting info,name,MSGLEVEL=1
// EXEC PROC=procname1
//TRN.SYSIN DD *2
CBL XOPTS(Translator options . . .)3.
COBOL source statements.
/*
//LKED.SYSIN DD *4
NAME anyname(R)
/*
//
```

Figure 165. Sample job control statements to call the DFHYITVL or DFHYXTVL procedures

where *procname* is the name of the procedure, and *anyname* is your load module name.

To use the procedure DFHZITCL or DFHZXTCL to invoke the integrated translator, you can use the job control statements that are shown in Figure 166 on page 697 :

```
//jobname JOB accounting info,name,MSGLEVEL=1
// EXEC PROC=procname,PROGLIB=dsnname1
//COBOL.SYSIN DD *
.. COBOL source statements.
/*
//LKED.SYSIN DD *
NAME anyname(R)
/*
//
```

Figure 166. Sample job control statements to use the DFHZITCL or DFHZXTCL procedure

where *anyname* is your load module name.

1. Translator options: specify the COBOL3 or COBOL2 translator option according to the version of the COBOL functionality that is required in the compile step.
2. Compiler options:

   To compile a COBOL program, you need the compiler options RENT and NODYNAM.

   Note: The compiler LIB option is no longer required for COBOL 5 and later, and has been removed. For earlier versions of COBOL, this option must be manually reinstated.

   If you use the translator option, CBLCARD (the default), the CICS translator automatically generates a CBL statement that contains these options. You can prevent the generation of a CBL or PROCESS card by specifying the translator option NOCBLCARD.

   The PARM statement of the COB step in the COBOL procedures supplied with CICS specifies values for the compiler options. For example:

   ```
   //COB EXEC PGM=IGYCRCTL,REGION=&REG,
   // PARM='NODYNAM,OBJECT,RENT,APOST,MAP,XREF'
   ```

   To compile a COBOL program with a compiler that has an integrated translator, you must use the CICS compiler option to indicate that you want the compiler to invoke the translator. The DFHZITCL procedure includes this compiler option:

   ```
   CBLPARM='NODYNAM,MAP,CICS(''COBOL3'')'
   ```

   Note: If you specify CICS translator options for the integrated translator in the PARM string, you need double apostrophes as shown in this example. If, however, you specify the options in your source

program, you need single apostrophes (for example, you might have `CBL CICS('COBOL3,SP') APOST` as the CBL statement in your source program.

The COBOL procedures supplied with CICS do not specify values for the SIZE and BUF options. The defaults are SIZE=MAX, and BUF=4K. SIZE defines the amount of virtual storage available to the compiler, and BUF defines the amount of dynamic storage to be allocated to each compiler buffer work file. You can change these options with a PARM.COB parameter in the EXEC statement that invokes the procedure. For example:

```
EXEC PROC=procname
,PARM.COB='SIZE=512K,BUF=16K,.,.,.'
```

Change compiler options by using any of the following methods:

- Override the PARM statement that is defined on the COB step of the COBOL procedures.

  If you specify a PARM statement in the job that calls the procedure, it overrides all the options that are specified in the procedure JCL. Ensure that all the options you want are specified in the override, or in a CBL statement.
- Specify a CBL statement at the start of the source statements in the job stream that is used to call the COBOL procedures.
- Use the COBOL installation defaults macro, IGYCOPT. This macro is required if you do not use a CBL statement (that is, you specify the translator option NOCBLCARD).
- Define a data set that contains the compiler options for your COBOL program, this data set must include the CICS compiler option and it subparameters.

  Code the SYSOPTF DD statement in one of the following ways:

  ```
  // SYSOPTF DD DSNAME=dsname,UNIT=SYSDA,VOLUME=(subparms),DISP=SHR
  ```

  In this code fragment, the compiler options are stored in the data set *dsname* .

  ```
  //COBOL EXEC PGM=IGYCRCTL,REGION=4M,PARM=(OPTFILE)
  //SYSOPTF DD *
  APOST
  TRUNC(OPT)
  CICS('COBOL3,SP')
  NODYNAM
  RENT
  LIST
  MAP
  XREF
  OPT
  TEST(ALL,SEPARATE)
  //STEPLIB DD DSN=PP.COBOL390.V410.SIGYCOMP,DISP=SHR
  ```

  In this code fragment, the compiler options are placed directly in the code, after the **OPTFILE** parameter.

  For more information about the SYSOPTF statement, see the Enterprise COBOL for z/OS Programming Guide.

For information about the translator option CBLCARD|NOCBLCARD, see "Defining translator options" on page 592. If you choose to use the NOCBLCARD option, also specify the COBOL compiler option ALOWCBL=NO to prevent an error message of IGYOS4006-E being issued. For more information about the ALOWCBL compiler option, see the relevant Installation and Customization documentation for your version of COBOL.

3. If you have no input for the translator, you can specify `DD DUMMY` instead of `DD *`. However, if you specify `DD DUMMY` , also code a suitable DCB operand. The translator does not supply all the data control block information for the SYSIN data set.
4. If the stand-alone translator supplied with CICS TS is used, the translator options on the XOPTS statement override similar options in the COBOL procedures.

For information about the translator options you can include on the XOPTS statement, see “Defining translator options” on page 592.

When the integrated CICS translator is used, the COBOL compiler recognizes only the keyword CICS for defining translator options, not XOPTS.

5. You can ignore weak external references unresolved by the link edit.

   The link-edit job step requires access to the libraries that contain the environment-specific modules for CICS , and the Language Environment link-edit modules, as appropriate. Override or change the names of these libraries if the modules and library subroutines are installed in libraries with different names.

   If you are installing a program into either of the read-only DSAs, see “Running application programs in the read-only DSAs” on page 686 for more details.

   If you are installing a program that is to be used from the LPA, add the RENT and REFR options to the LNKPARM parameter on the call to the COBOL procedures. For more information, see “Running applications in the link pack area” on page 686.

Installing PL/I application programs

You can use the DFHY*x*TPL procedures to process PL/I applications with a separate translator. The value of *x* depends on whether it is a CICS application program or an EXCI batch program. Alternatively, the DFHZ*x*TPL procedures can be used to invoke the integrated translator.

Figure 164 on page 696 illustrates the flow of control in the cataloged procedures for PL/I programs.

For more information about preparing PL/I programs, see the Enterprise PL/I for z/OS Programming Guide.

Installing C application programs

This diagram shows you the flow of control in the DFHYxTzL cataloged procedures for C command-level programs that require a separate translator step. If you use an integrated translator, there is no separate translator step. The high-level language source and CICS.SDFHLOAD both input to the compiler, and a combined translator and compiler listing is produced.

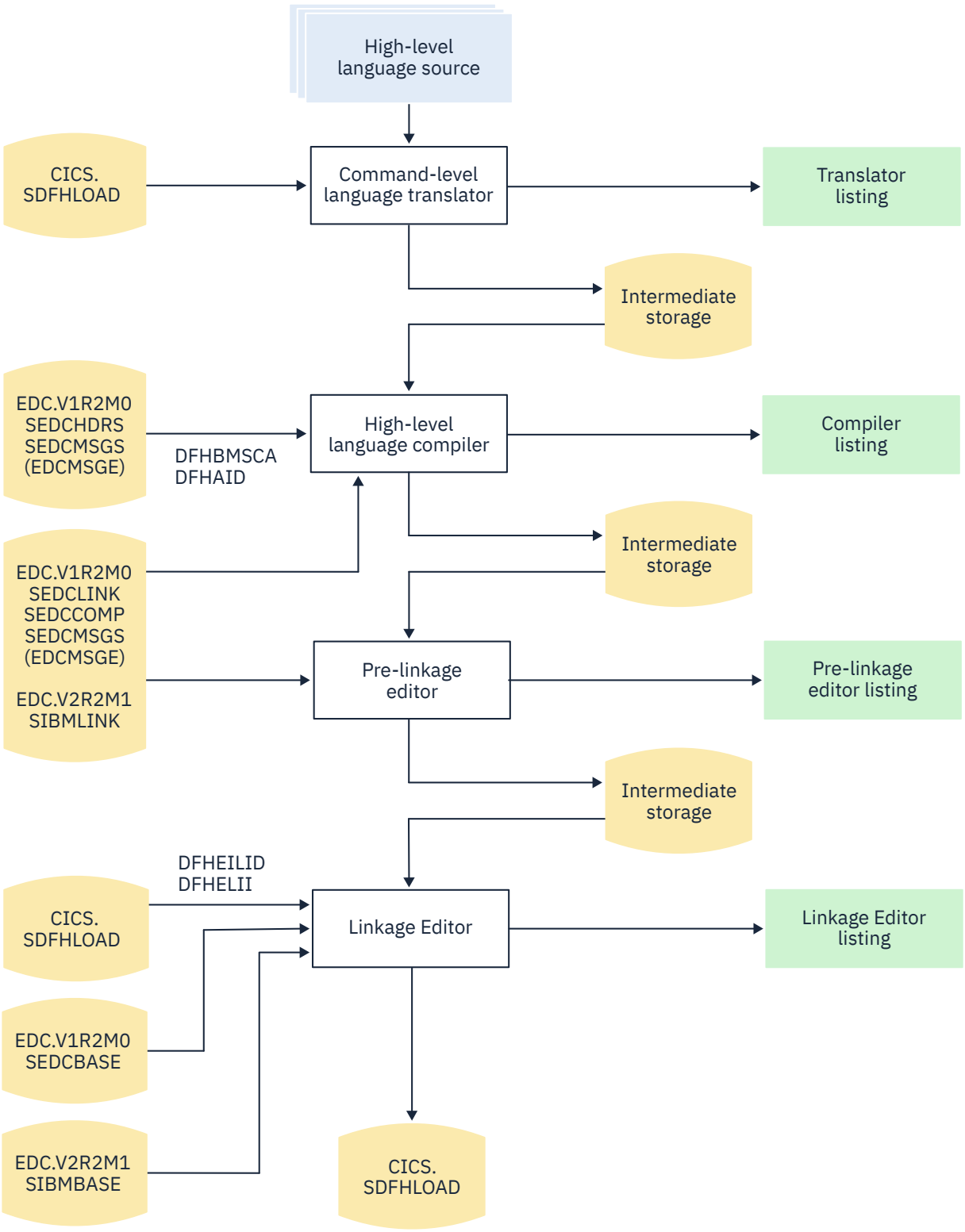


Figure 167. Installing C programs using the DFHYxTzL procedure

There are translator, compiler, pre-linkage editor and linkage editor steps, each producing a listing and an intermediate file that is passed to the next step. C libraries are referenced in the compiler, pre linkage editor, and linkage editor steps.

Note: When you choose the XPLINK compiler option, there is no pre-link step in this diagram.

Before you can install any C programs, you must have installed the C library and compiler and generated CICS support for C. (See Adding CICS support for programming languages.)

Sample JCL to install C application programs

You can use the job control statements shown in these examples to process C application programs. In the procedure name, x depends on whether your program is a CICS application program or an EXCI batch program.

For the names of the CICS-supplied procedures, see Table 65 on page 690.

```
//jobname JOB accounting info,name,MSGLEVEL=1
// EXEC PROC=DFHYxTzL1
//TRN.SYSIN DD *2
#pragma XOPTS(Translator options . . .)3
.C source statements.
/*
//LKED.SYSIN DD *4
NAME anyname(R)
/*
//
```

where
anyname
is your load module name.

Figure 168. Sample JCL to call the DFHYxTzL procedures

Notes for installing a C program

1. **Compiler options:** You can code compiler options by using the parameter override **CPARM** in the EXEC statement that invokes the procedure, or on a #pragma options directive.

   If you want to make your C or C++ source available to the DFHYITEL, DFHZITEL or DFHYXTEL JCL procedures you must restrict the source input margins. You can restrict the source input margins by:

   - Altering the C or C++ source directly by specifying the following pragma declaration:

     ```
     #pragma margins(m,n)
     ```

     where *m* and *n* are the columns where the C or C++ source is located. For example #pragma margins(1,72).
   - Altering the **CPARM** override options in the JCL procedure; specify SEQ or NOSEQ depending on whether the source has sequence numbers or not.
2. If you have no input for the translator, you can specify DD DUMMY instead of DD *. However, if you specify DD DUMMY , also code a suitable DCB operand. (The translator does not supply all the data control block information for the SYSIN data set.)
3. **Translator options:** For information about the translator options you can include on the XOPTS statement, see “Defining translator options” on page 592.
4. If you are installing a program into either of the read-only DSAs, see “Running application programs in the read-only DSAs” on page 686 for more details.

If you are installing a program that is to be used from the LPA, add the RENT and REFR options to the LNKPARM parameter on the call to the DFHYxTzL procedure. (See “Running applications in the link pack area” on page 686 for more information.)

C language programs must be link-edited with AMODE(31), so the DFHYxTzL procedures specify AMODE(31) by default.

Invoking the integrated CICS translator for XL C

To use the procedures to invoke the integrated translator for XL C, you can use the job control statements shown in Figure 169 on page 702 :

```
//jobname JOB accounting info,name,MSGLEVEL=1
// EXEC DFHZITxL,PROGLIB=dsnname1
//C.SYSIN DD *
.. C source statements.
//LKED.SYSIN DD *
NAME anyname(R)

//
```

where
anyname
is your load module name.

Figure 169. Sample JCL for integrated translator for XL C

1. **Translator name:** Specify DFHZITDL for C programs without XPLINK, or DFHZITFL for C programs with XPLINK.

Invoking the integrated CICS translator for XL C++

To use the procedures to invoke the integrated translator for XL C++, you can use the job control statements shown in Figure 170 on page 702 :

```
//jobname JOB accounting info,name,MSGLEVEL=1
// EXEC DFHZITxL,PROGLIB=dsnname1
//CPP.SYSIN DD *
.. C++ source statements.
//LKED.SYSIN DD *
NAME anyname(R)

//
```

where
anyname
is your load module name.

Figure 170. Sample JCL for integrated translator for XL C++

1. **Translator name:** Specify DFHZITEL for C++ programs without XPLINK, or DFHZITGL for C++ programs with XPLINK.

Including pre-translated code with your C source code

The translator can generate dfhexec or DFHEXEC. If both versions are present in your program, error message IEW2456E is displayed. There are two ways to prevent this error either recompile the old code containing dfhexec or use prelinker RENAME control statement in the job.

About this task

The following sample JCL shows you how to use the RENAME control statement.

```
//jobname JOB accounting info,name,MSGLEVEL=1
// EXEC PROC=DFHYxTzL
//TRN.SYSIN DD *
#pragma XOPTS(Translator options . . .).
C source statements.
/*
//PLKED.SYSLIN DD *
RENAME dfhexec DFHEI1
//LKED.SYSLIN DD *
NAME anyname(R)
/*
//
```

where
anyname
is your load module name

Figure 171. Sample JCL to rename dfhexec

Using your own job streams

You can use the supplied cataloged procedures as a model to write your own JCL to translate, assemble (or compile), and link-edit your application programs.

The procedures are installed in the CICSTS56.CICS.SDFHPROC library.

The following information summarizes the important points about the translator and each main category of programs. For simplicity, the descriptions assume that programs are loaded into CICSTS56.CICS.SDFHLOAD or IMS.PGMLIB. You can load programs into any libraries, but only when they are either included in the DFHRPL or dynamic LIBRARY concatenation in the CICS job stream, or included in the STEPLIB library concatenation in the batch job stream (for a stand-alone IMS batch program).

Note: The IMS libraries referred to in the job streams are identified by IMS.libnam (for example IMS.PGMLIB). If you use your own naming convention for IMS libraries, you must rename the IMS libraries accordingly.

Translator requirements

The CICS translator requires a minimum of 256 KB of virtual storage. You might need to use the translator options CICS and DLI.

Online programs that use EXEC CICS or EXEC DLI commands

1. Always use the translator option CICS. If the program issues EXEC DLI commands, use the translator option DLI.
2. The link-edit input (defined by the SYSLIN DD statement) must include the correct interface module **before** the object deck. Therefore, place an INCLUDE statement for the interface module before the object deck. Also put ORDER statements before the INCLUDE statements, and an ENTRY statement after all the INCLUDE statements.

   The interface modules are as follows:

DFHEAI
Assembler

DFHELII
All HLL languages

In the CICS-supplied procedures, the input to the link-edit step (defined by the SYSLIN DD statement) concatenates a library member with the object deck. This member contains an INCLUDE statement for the required interface module. For example, the DFHYITVL procedure concatenates the library member DFHEILID, which contains the following INCLUDE statement:

```
INCLUDE SYSLIB(DFHELII)
```

3. Place the load module output from the link-edit (defined by the SYSLMOD DD statement) in CICSTS56.CICS.SDFHLOAD, or your own program library.

Figure 172 on page 704 shows sample JCL and an inline procedure, based on the CICS-supplied procedure DFHYITVL, that can be used to install COBOL application programs. The procedure does not include the COPYLINK step and concatenation of the library member DFHEILID that contains the INCLUDE statement for the required interface module (as included in the DFHYITVL procedure). Instead, the JCL provides the following INCLUDE statement:

```
INCLUDE SYSLIB(DFHELII)
```

If this statement was not provided, the link-edit would return an error message for unresolved external references, and the program output would be marked as not executable.

Figure 172. Sample user-defined JCL to install a COBOL program

```
//* The following JCL could be used to execute this procedure
//*
//APPLPROG EXEC MYYITVL,
// INDEX='CICSTS56.CICS
// PROGLIB='CICSTS56.CICS.SDFHLOAD',
// DSCTLIB='CICSTS56.CICS.SDFHCOB',
// INDEX2='user.qualif'
// OUTC=A, Class for print output
// REG=4M, Region size for all steps
// LNKPARM='LIST,XREF', Link edit parameters
// WORK=SYSDA Unit for work data sets

//TRN.SYSIN DD *
//* .
//* . Application program
//* .
//*
//LKED.SYSIN DD *
INCLUDE SYSLIB(DFHELII)
NAME anyname(R)
//*
//MYYITVL PROC SUFFIX=1$, Suffix for translator module
// INDEX='CICSTS56.CICS
', Qualifier(s) for CICS libraries
// PROGLIB='CICSTS56.CICS.SDFHLOAD', Name of o/p library
// DSCTLIB='CICSTS56.CICS.SDFHCOB', Private macro/dsect
// AD370HLQ='SYS1', Qualifier(s) for AD/Cycle compiler
// LE370HLQ='SYS1', Qualifier(s) for Language Environment libraries
// OUTC=A, Class for print output
// REG=4M, Region size for all steps
// LNKPARM='LIST,XREF', Link edit parameters
// WORK=SYSDA Unit for work data sets
//*
```

```
//* This procedure contains 3 steps
//* 1. Exec the COBOL translator (using the supplied suffix 1$)
//* 2. Exec the COBOL compiler
//* 3. Linkedit the output into data set &PROGLIB

//TRN EXEC PGM=DFHECP &SUFFIX,,
// PARM='COBOL3',
// REGION=&REG
```

```
//STEPLIB DD DSN=&INDEX..SDFHLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSPUNCH DD DSN=&&SYSCIN,
// DISP=(,PASS),UNIT=&WORK,
// DCB=BLKSIZE=400,
// SPACE=(400,(400,100))
//*
//COB EXEC PGM=IGYCRCTL,REGION=&REG,
// PARM='NODYNAM,OBJECT,RENT,APOST,MAP,XREF'
//*     Note:
//*     The compiler LIB option is no longer required for
//*     COBOL 5 and later, and the option has been removed.
//*     For earlier versions of COBOL, this option must be
//*     manually reinstated.
//*
//STEPLIB DD DSN=&AD370HLQ..SIGYCOMP,DISP=SHR
//SYSLIB DD DSN=&DSCTLIB,DISP=SHR
// DD DSN=&INDEX..SDFHCOB,DISP=SHR
// DD DSN=&INDEX..SDFHMAC,DISP=SHR
// DD DSN=&INDEX..SDFHSAMP,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSIN DD DSN=&&SYSCIN,DISP=(OLD,DELETE)
//SYSLIN DD DSN=&&LOADSET,DISP=(MOD,PASS),
// UNIT=&WORK,SPACE=(80,(250,100))
//SYSUT1 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT2 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT3 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT4 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT5 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT6 DD UNIT=&WORK,SPACE=(460,(350,100))
//*
//LKED EXEC PGM=IEWL,REGION=&REG,
// PARM='&LNKPARM',COND=(5,LT,COB)
//SYSLIB DD DSN=&INDEX..SDFHLOAD,DISP=SHR
// DD DSN=&LE370HLQ..SCEELKED,DISP=SHR
//SYSLMOD DD DSN=&PROGLIB,DISP=SHR
//SYSUT1 DD UNIT=&WORK,DCB=BLKSIZE=1024,
// SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=&OUTC
//SYSLIN DD DSN=&&COPYLINK,DISP=(OLD,DELETE)
// DD DSN=&&LOADSET,DISP=(OLD,DELETE)
// DD DDNAME=SYSIN
//PEND
//*
```

Online programs that use the CALL DLI interface

1. Specify the translator option CICS , but not the translator option DLI.

   Note: For a program that does not use CICS commands and is invoked only by a running transaction (and never directly by CICS task initiation), no translator step is needed.
2. The interface module, DFHDLIAI, is automatically included by the link-edit. If you use an INCLUDE statement in the link-edit input, place it **after** the object deck.
3. Include copybook DLIUIB in your program.
4. Place the load module output from the link-edit (defined by the SYSLMOD DD statement) in CICSTS56.CICS.SDFHLOAD, or a user-defined application program library.

Batch or BMP programs that use EXEC DLI commands

1. The translator option DLI is required. Do not specify the translator option CICS.
2. The INCLUDE statement for the interface module must **follow** the object deck in the input to the link-edit (defined by the SYSLIN DD statement). The interface module, DFSLI000, which resides on IMS.RESLIB, is the same for all programming languages. If you include CICSTS56.CICS.SDFHLOAD in the input to the link-edit (defined by the SYSLIB DD statement), concatenate it **after** IMS.RESLIB.
3. Place the load module output from the link-edit (defined by the SYSLMOD DD statement) in IMS.PGMLIB, or a library concatenated in the STEPLIB DD statement of the batch job stream.

Batch or BMP programs that use DL/I CALL commands

If you want to prepare assembler, COBOL, or PL/I programs that use the DL/I CALL interface, do not use the CICS-supplied procedures. Programs that contain CALL ASMTDLI, CALL CBLTDLI, or CALL PLITDLI should be assembled or compiled, and link-edited, as IMS applications, and are not subject to any CICS requirements. See the relevant IMS manual for information about how to prepare application programs that use the DL/I CALL interface.

Installing map sets and partition sets

Use the basic mapping support (BMS) facility of CICS to assemble and link-edit map sets and partition sets. You can use the BMS macros to install HTML templates generated from BMS maps.

If your program uses BMS maps, you need to create the maps. The traditional method for doing this is to code the map in BMS macros and assemble these macros. You do the assembly twice, with different output options.

- One assembly creates a set of definitions. You copy these definitions into your program using the appropriate language statement, and they allow you to refer to the fields in the map by name.
- The second assembly creates an object module that is used when your program executes.

The process is illustrated in the following diagram:

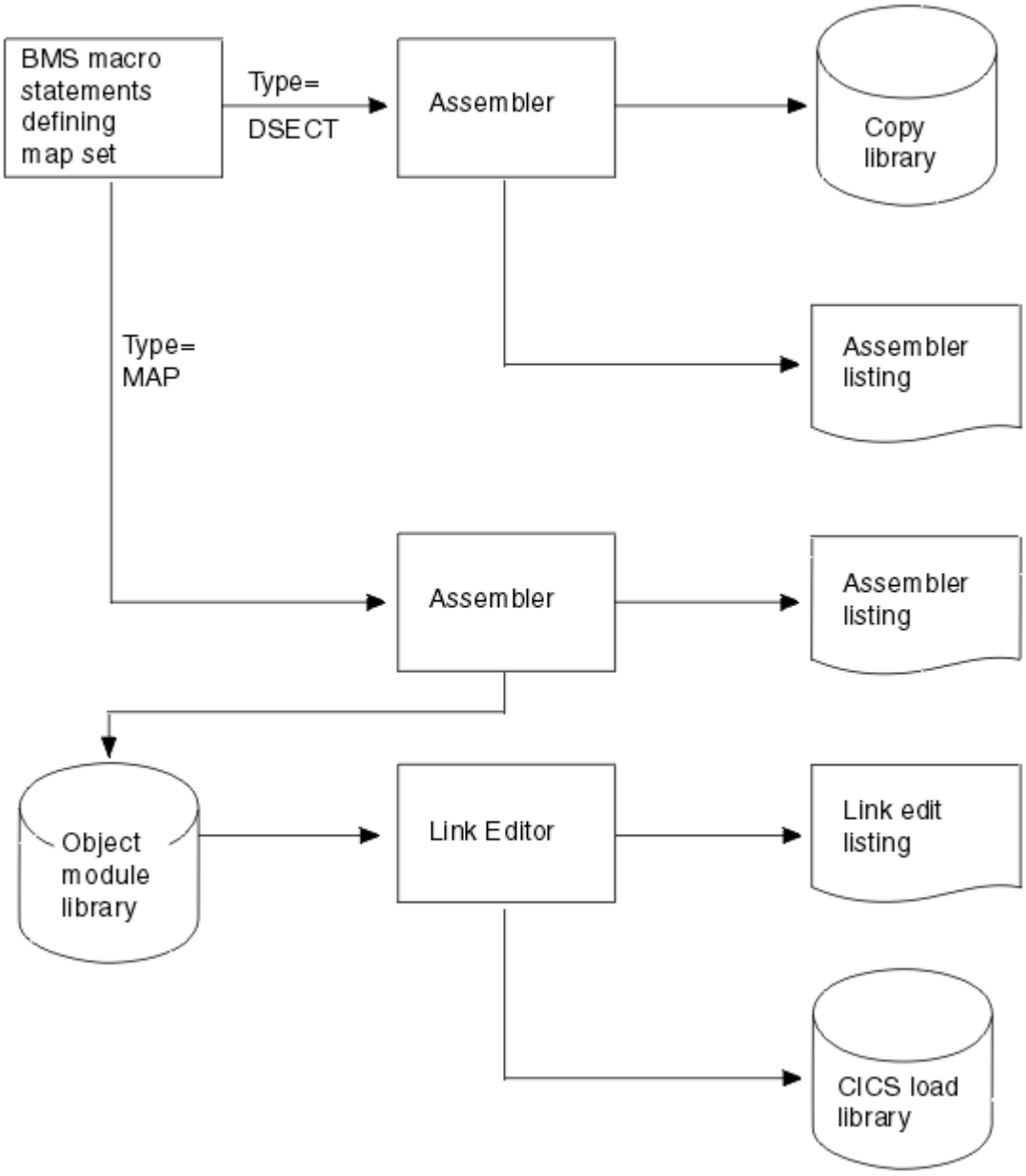


Figure 173. Preparing a map

Whatever way you produce maps, you need to create a map before you compile (assemble) any program that uses it. In addition, if you change the map, you typically need to recompile (reassemble) all programs that use it. Some changes affect only the physical map and are not reflected in the corresponding symbolic map used by the program. One of these is a change in field position that does not alter the order of the fields. However, changes in data type, field length, field sequence, and others do affect the symbolic map, and it is always safest to recompile (reassemble).

CICS also supports the definition of BMS map sets and partition sets interactively by using licensed programs such as the IBM Screen Definition Facility II (SDF II). For more information about SDF II, see Screen Definition Facility II home site.

CICS loads BMS map sets and partition sets above the 16 MB line if you specify the residency mode for the map set or partition set as RMODE(ANY) in the link-edit step. If you are using either map sets or partition sets from earlier releases of CICS, you can load them above the 16 MB line by link-editing them again with RMODE(ANY). For examples of link-edit steps specifying RMODE(ANY), see the sample job streams in this section.

Installing map sets

Use these examples to learn how to install physical map sets and symbolic description map sets both separately and together.

This section first describes the types of map sets, how you define them, and how CICS recognizes them. This is followed by a description of how to prepare physical map sets and symbolic description map sets separately. Finally, there is a description of how to prepare both physical and symbolic description map sets in one job. In these descriptions, it is assumed that the SYSPARM parameter is used to distinguish the two types of map sets.

See:

- "Types of map sets" on page 708
- "Installing physical map sets" on page 709
- "Installing symbolic description map sets" on page 711
- "Installing physical and symbolic description maps together" on page 713

Types of map sets

To install a map set, you must prepare two types of map sets; a physical map set and a symbolic description map set.

- A physical map set is used by BMS to translate data from the standard device independent form used by application programs to the device-dependent form required by terminals.
- A symbolic description map set is used in the application program to define the standard device-independent form of the user data. This is a DSECT in assembler language, a data definition in COBOL, a BASED or AUTOMATIC structure in PL/I, and a "struct" in C/370.

Physical map sets must be cataloged in the CICS load library. Symbolic description map sets can be cataloged in a user copy library, or inserted directly into the application program itself.

The map set definition macros are assembled twice; once to produce the physical map set used by BMS in its formatting activities, and once to produce the symbolic description map set that is copied into the application program.

Defining the type of map set you require

You can distinguish the two types of map set by using either the TYPE operand of the DFHMSD macro or the SYSPARM operand on the EXEC statement of the job used to assemble the map set.

If you use the SYSPARM operand for this purpose, the TYPE operand of the DFHMSD macro is ignored. Using SYSPARM allows both the physical map set and the symbolic description map set to be generated from the same unchanged set of BMS map set definition macros.

Map sets can be assembled as either unaligned or aligned (an aligned map is one in which the length field is aligned on a halfword boundary). Use unaligned maps except in cases where an application package needs to use aligned maps.

The SYSPARM value alone determines whether the map set is aligned or unaligned, and is specified on the EXEC PROC=DFHMAPS statement. The SYSPARM operand can also be used to specify whether a physical map set or a symbolic description map set (DSECT) is to be assembled, in which case it overrides the TYPE operand. If neither operand is specified, an unaligned DSECT is generated.

The TYPE operand of the DFHMSD macro can only define whether a physical or symbolic description map set is required.

For the possible combinations of operands to generate the various types of map set, see Table 66 on page 709.

Table 66. SYSPARM and DFHMSD operand combinations for map assembly		
Type of map set	**SYSPARM operand of EXEC DFHMAPS statement**	**TYPE operand of DFHMSD macro**
Aligned symbolic description map set (DSECT)	A A ADSECT	Not specified DSECT Any (takes SYSPARM)
Aligned physical map set	A AMAP	MAP Any (takes SYSPARM)
Unaligned symbolic description map set (DSECT)	Not specified Not specified DSECT	Not specified DSECT Any (takes SYSPARM)
Unaligned physical map set	Not specified MAP	MAP Any (takes SYSPARM)

The physical map set indicates whether it was assembled for aligned or unaligned maps. This information is tested at execution time, and the appropriate map alignment used. Thus aligned and unaligned map sets can be mixed.

Using extended data stream terminals

You can use fixed extended data stream attributes by reassembling the physical map set or for dynamic attribute medication, by reassembling both the physical, and symbolic description map sets.

Applications and maps designed for the 3270 Information Display System run unchanged on devices supporting extensions to the 3270 data stream such as color, extended highlighting, programmed symbols, and validation. To use fixed extended attributes such as color, you only need to reassemble the physical map set. If dynamic attribute modification by the application program is needed, you must reassemble both the physical and symbolic description map sets, and you must reassemble or recompile the application program.

Installing physical map sets

These examples show you the assembler and linkage editor steps for installing physical map sets and an example job stream.

Figure 174 on page 710 shows the assembler and linkage editor steps for installing physical map sets.

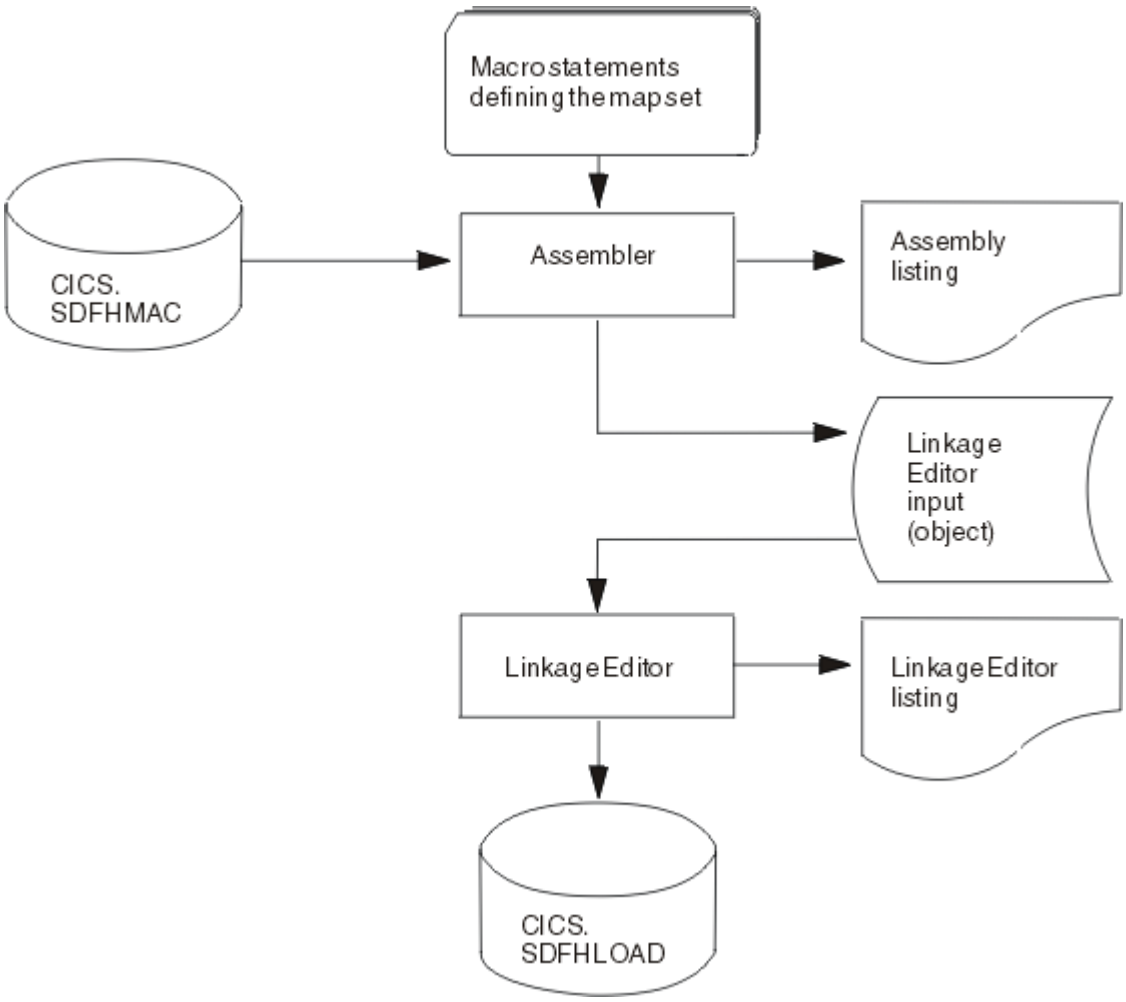


Figure 174. Installing physical map sets

Figure 175 on page 710 gives an example job stream for the assembly and link-editing of physical map sets.

```
//PREP JOB 'accounting information',CLASS=A,MSGLEVEL=1
//STEP1 EXEC PROC=DFHASMVS,PARM.ASSEM='SYSPARM(MAP)'1
//SYSPUNCH DD DSN=&&TEMP,DCB=(RECFM=FB,BLKSIZE=2960),
// SPACE=(2960,(10,10)),UNIT=SYSDA,DISP=(NEW,PASS)
//SYSIN DD *

Macro statements defining the map set

/*
//STEP2 EXEC PROC=DFHLNKVS,PARM='LIST,LET,XREF'2
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
// DD *
MODE RMODE(ANY|24)3
NAME mapsetname(R)4
/*
//
```

Figure 175. Assembling and link-editing a physical map set

Notes

1. For halfword-aligned length fields, specify the option SYSPARM(AMAP) instead of SYSPARM(MAP).
2. Physical map sets are loaded into CICS-key storage, unless they are link-edited with the RMODE(ANY) and RENT options. If they are link-edited with these options, they are loaded into key-0 protected storage, if RENTPGM=PROTECT is specified on the RENTPGM initialization parameter. However, it is

recommended that map sets (except for those that are only sent to 3270 or LU1 devices) should not be link-edited with the RENT or the REFR options because, in some cases, CICS modifies the map set. Generally, use the RENT or REFR options for map sets that are only sent to 3270 or LU1 devices. For more information about the storage protection facilities available in CICS, see Storage protection.

3. The MODE statement specifies whether the map set is to be loaded above (RMODE(ANY)) or below (RMODE(24)) the 16 MB line. RMODE(ANY) indicates that CICS can load the map set anywhere in virtual storage, but tries to load it above the 16 MB line, if possible.
4. Use the NAME statement to specify the name of the physical map set that BMS loads into storage. If the map set is device-dependent, derive the map set name by appending the device suffix to the original 1- to 7-character map set name used in the application program. The suffixes to be appended for the various terminals supported by CICS BMS depend on the parameter specified in the TERM or SUFFIX operand of the DFHMSD macros used to define the map set.

   To use a physical map set, you must define and install a resource definition for it. You can do this either by using the program autoinstall function or by using the **CEDA DEFINE MAPSET** and **INSTALL** commands. as described in "Defining programs, map sets, and partition sets to CICS" on page 718.

Installing symbolic description map sets

These examples show you the steps for installing symbolic description map sets using the DFHASMVS procedure.

Symbolic description map sets enable the application programmer to make symbolic references to fields in the physical map set. Figure 176 on page 712 shows the preparation of symbolic description map sets for BMS.

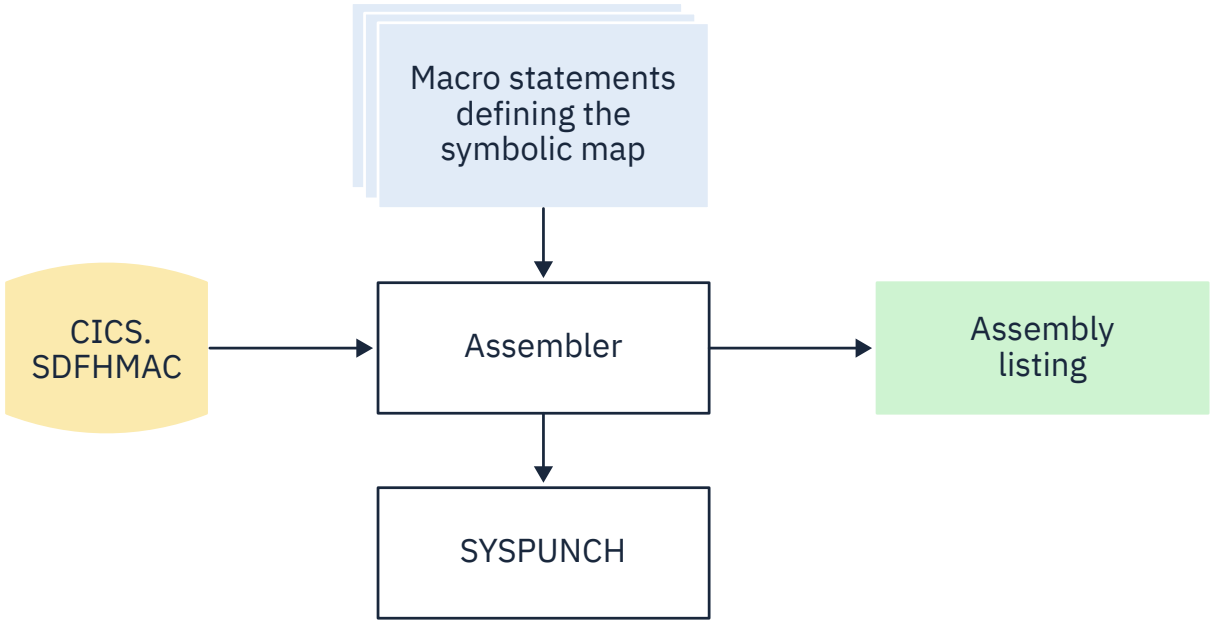


Figure 176. Installing symbolic description map sets using the DFHASMVS procedure

To use a symbolic description map set in a program, you must assemble the source statements for the map set and obtain a punched copy of the storage definition through SYSPUNCH. The first time this is done, you can direct the SYSPUNCH output to SYSOUT=A to get a listing of the symbolic description map set. If many map sets are to be used at your installation, or there are multiple users of common map sets, establish a private user copy library for each language that you use.

When a symbolic description is prepared under the same name for more than one programming language, a separate copy of the symbolic description map set must be placed in each user copy library. You must ensure that the user copy libraries are correctly concatenated with SYSLIB.

You need only one symbolic description map set corresponding to all the different suffixed versions of the physical map set. For example, to run the same application on terminals with different screen sizes, you would:

1. Define two map sets each with the same fields, but positioned to suit the screen sizes. Each map set has the same name but a different suffix, which would match the suffix specified for the terminal.
2. Assemble and link-edit the different physical map sets separately, but create only one symbolic description map set, because the symbolic description map set would be the same for all physical map sets.

You can use the sample job stream in Figure 177 on page 713 to obtain a listing of a symbolic description map set. It applies to all the programming languages supported by CICS.

```
//DSECT JOB 'accounting information',CLASS=A,MSGLEVEL=1
//ASM EXEC PROC=DFHASMVS,PARM.ASSEM='SYSPARM(DSECT)'
//SYSPUNCH DD SYSOUT=A
//SYSIN DD *

Macro statements defining the map set

/*
//
```

Figure 177. Listing of a symbolic description map set

If you want to assemble symbolic description map sets in which length fields are halfword-aligned, change the EXEC statement of the sample job in Figure 177 on page 713 to the following:

```
//ASSEM EXEC PROC=DFHASMVS,PARM.ASSEM='SYSPARM(ADSECT)'
```

To obtain a punched copy of a symbolic description map set, code the //SYSPUNCH statement in the previous example to direct output to the punch data stream. For example:

```
//SYSPUNCH DD SYSOUT=B
```

To store a symbolic description map set in a private copy library, use job control statements similar to the following:

```
//SYSPUNCH DD DSN=USER.MAPLIB.ASM(map set name),DISP=OLD
//SYSPUNCH DD DSN=USER.MAPLIB.COB(map set name),DISP=OLD
//SYSPUNCH DD DSN=USER.MAPLIB.PLI(map set name),DISP=OLD
```

Installing physical and symbolic description maps together

These examples show you the steps for installing physical and symbolic description map sets using the DFHMAPS procedure.

Figure 178 on page 715 shows the DFHMAPS procedure for installing physical and symbolic description maps together. The DFHMAPS procedure consists of the following four steps, shown in Figure 178 on page 715 :

1. The BMS macros that you coded for the map set are added to a temporary sequential data set.
2. The macros are assembled to create the physical map set. The MAP option is coded in the SYSPARM global variable in the EXEC statement (PARM='SYSPARM(MAP)').

3. The physical map set is link-edited to the CICS load library.
4. Finally, the macros are assembled again, this time to produce the symbolic description map set. In this step, DSECT is coded in the SYSPARM global variable in the EXEC statement (PARM='SYSPARM(DSECT)'). Output is directed to the destination specified in the //SYSPUNCH DD statement. In the DFHMAPS procedure, that destination is the CICSTS56.CICS .SDFHMAC library.

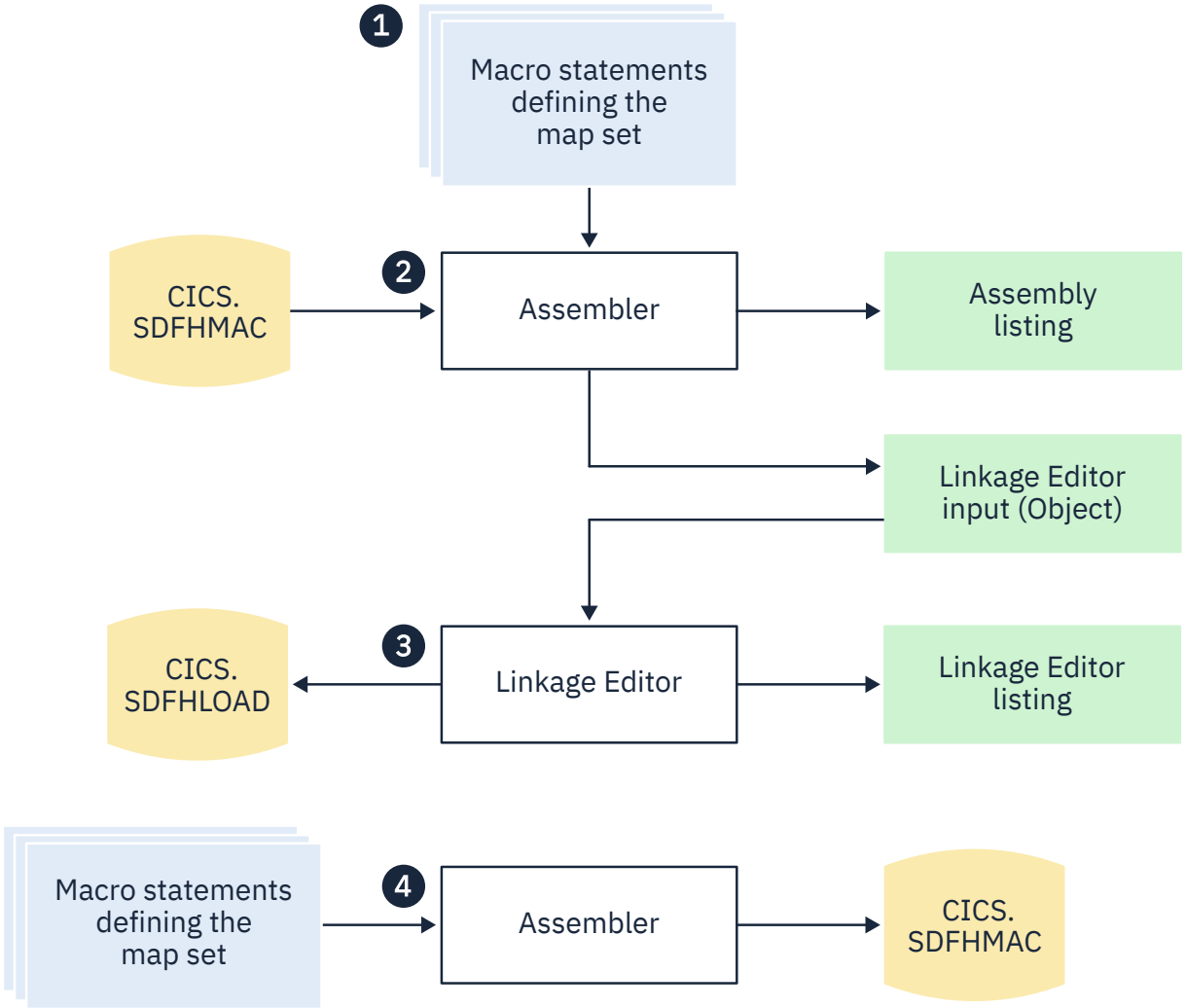


Figure 178. Installing a physical map set and a symbolic description map set together

Using the DFHMAPT procedure to install HTML templates from BMS maps

The DFHMAPT procedure is similar to DFHMAPS, with an additional step that installs HTML templates generated from the BMS maps.

About this task

In this step, TEMPLATE is coded in the SYSPARM global variable in the EXEC statement (PARM='SYSPARM(TEMPLATE)'). In the DFHMAPT procedure, the output is directed to CICSTS56.CICS.SDFHHTML.

If you want to use your own macro to customize HTML templates, and you do not want to add your macro to the BMS source you should modify step ASMTEMPL:

1. Change the PARM parameter of the EXEC statement to

```
PARM='SYSPARM(TEMPLATE,macro_name),DECK,NOOBJECT'
```

2. Add the library that contains your macro to the SYSLIB concatenation.

JCL to install physical and symbolic description maps

This example shows you the JCL job stream needed to install the physical map sets and the symbolic description map sets together.

The load module from the assembly of the physical map set and the source statements for the symbolic description map set can be produced in the same job by using the sample job stream in Figure 179 on page 716.

```
//PREPARE JOB 'accounting
information',CLASS=A,MSGLEVEL=1
//ASSEM EXEC PROC=DFHMAPS,MAPNAME=mapsetname,RMODE=ANY|24 (see note)
//SYSUT1 DD *

Macro statements defining the map set

/*
//
```

Figure 179. Installing physical and symbolic description maps together

Note: The RMODE statement specifies whether the map set is to be loaded above (RMODE=ANY) or below (RMODE=24) the 16MB line. RMODE=ANY indicates that CICS can load the map set anywhere in virtual storage, but tries to load it above the 16MB line, if possible.

The DFHMAPS procedure produces map sets that are not halfword-aligned. If you want the length fields in input maps to be halfword-aligned, you have to code A=A on the EXEC statement. In the sample job in Figure 179 on page 716 , change the EXEC statement to:

```
//ASSEM EXEC PROC=DFHMAPS,MAPNAME=mapsetname,A=A
```

This change results in the SYSPARM operands in the assembly steps being altered to SYSPARM(AMAP) and SYSPARM(ADSECT) respectively.

The DFHMAPS procedure directs the symbolic description map set output (SYSPUNCH) to the CICSTS56.CICS.SDFHMAC library. Override this by specifying DSCTLIB=name on the EXEC statement, where "name" is the name of the chosen user copy library.

Adding a CSECT to your map assembly

This example shows you how to add both a CSECT name and AMODE and RMODE statements to your map assembly.

It is possible that you might need to generate your BMS maps with a CSECT. For example, you might need to specify AMODE and RMODE options to ensure that your maps reside above 16 MB, or you might need to use the DFSMS binder IDENTIFY statement for reasons of change management. In this case, you need not only include the appropriate CSECT at the front of your BMS macro statements, but also add some

conditional assembler statements to ensure that the CSECT statement is not included in the symbolic description map. The following example shows how you can add both a CSECT name and AMODE and RMODE statements:

```
//PREPARE JOB 'accounting
information',CLASS=A,MSGLEVEL=1
//ASSEM EXEC PROC=DFHMAPS,MAPNAME=mapsetname,RMODE=ANY|24
//SYSUT1 DD *.
AIF ('&SYSPARM' EQ 'DSECT').SKIPSD
AIF ('&SYSPARM' EQ 'ADSECT').SKIPSD
ANYNAME CSECT Binder IDENTIFY requires CSECT name
ANYNAME AMODE 31
ANYNAME RMODE ANY
.SKIPSD ANOP ,
DFH0STM DFHMSD TYPE=DSECT,MODE=INOUT,CTRL=FREEKB,LANG=COBOL, C
TIOAPFX=YES,TERM=3270-2,MAPATTS=(COLOR,HILIGHT), C
DSATTS=(COLOR,HILIGHT)
SPACE
DFH0STM DFHMDI SIZE=(24,80)...
SPACE
DFHMSD TYPE=FINAL
END.
/*
//
```

Figure 180. Adding a CSECT to the map assembly

Installing partition sets

You can install partition sets in the same way as physical map sets. There is no concept of a symbolic description partition set.

The job stream in Figure 181 on page 717 is an example of the assembly and link-edit of partition sets.

```
//PREP JOB 'accounting information',CLASS=A,MSGLEVEL=1
//STEP1 EXEC PROC=DFHASMVS
//SYSPUNCH DD DSN=&&TEMP,DCB=(RECFM=FB,BLKSIZE=2960),
// SPACE=(2960,(10,10)),UNIT=SYSDA,DISP=(NEW,PASS)
//SYSIN DD *.
Macro statements defining the partition set.
/*
//STEP2 EXEC PROC=DFHLNKVS,PARM='LIST,LET,XREF'1
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
// DD *
MODE RMODE(ANY|24)2
NAME partitionsetname(R)3
/*
//
```

Figure 181. Assembling and link-editing a partition set

Notes

1. A partition set is loaded into CICS -key storage, unless it is link-edited with the RMODE(ANY) and RENT options. If it is link-edited with these options, it is loaded into key-0 protected storage, if RENTPGM=PROTECT is specified on the **RENTPGM** system initialization parameter.

   For more information about the storage protection facilities available in CICS, see Storage protection.
2. The MODE statement specifies whether the partition set is to be loaded above (RMODE(ANY)) or below (RMODE(24)) the 16 MB line. RMODE(ANY) indicates that CICS can load the partition set anywhere in virtual storage, but tries to load it above the 16MB line, if possible.
3. Use the NAME statement to specify the name of the partition set which BMS loads into storage. If the partition set is device-dependent, derive the partition set name by appending the device suffix to the original 1- to 7-character partition set name used in the application program. The suffixes that BMS appends for the various terminals depend on the parameter specified in the SUFFIX operand of the DFHPSD macro that defined the partition set.

For programming information giving a complete list of partition-set suffixes, see LENGTH options in CICS commands.

To use a partition set, you must define and install a resource definition for it. You can do this either by using the program autoinstall function or by using the **CEDA DEFINE PARTITIONSET** and **INSTALL** commands, as described in The CEDA DEFINE command and The CEDA INSTALL command

Defining programs, map sets, and partition sets to CICS

To be able to use a program that you have installed in one of the load libraries specified in your CICS startup JCL, the program, and any map sets and partition sets that it uses, must be defined to CICS. To do this, CICS uses the resource definitions MAPSET (for map sets), PARTITIONSET (for partition sets), and PROGRAM (for programs).

About this task

You can create and install such resource definitions in any of the following ways:

- CICS can dynamically create, install, and catalog a definition for the program, map set, or partition set when it is first loaded, by using the autoinstall for programs function.
- You can create a specific resource definition for the program, map set, or partition set and install that resource definition in your CICS region.

  You can install resource definitions in either of the following ways:

  – At CICS initialization, by including the resource definition group in the group list specified on the GRPLIST system initialization parameter.
  – While CICS is running, by the CEDA INSTALL command.

For information about defining programs to CICS, see PROGRAM resources.

Testing applications

You can use the following methods to test CICS application programs. This guidance does not relate to testing Java applications.

Single-thread testing

A **single-thread** test takes one application transaction at a time, in an otherwise "empty" CICS system, and sees how it behaves. This enables you to test the program logic, and also shows whether the basic CICS information (such as resource definition) is correct. It is feasible to test this single application in one CICS region while your normal, online production CICS system is active in another.

Multithread testing

A **multithread** test involves several concurrently active transactions. Naturally, all the transactions are in the same CICS region, so you can readily test the ability of a new transaction to coexist with them.

You might find that a transaction that works perfectly in its single-thread testing still fails in the multithread test. It might also cause other transactions to fail, or even terminate CICS.

Regression testing

A **regression** test is used to make sure that all the transactions in a system continue to do their processing in the same way both before and after changes are applied to the system. This is to ensure that fixes applied to solve one problem do not cause further problems. It is a good idea to build a set of miniature files to perform your tests on, because it is much easier to examine a small data file for changes.

A good regression test exercises all the code in every program; that is, it explores all tests and possible conditions. As your system develops to include more transactions, more possible conditions, and so on, add these to your test system to keep it in step. The results of each test should match

those from the previous round of testing. Any discrepancies are grounds for suspicion. You can compare terminal output, file changes, and log entries for validity.

Sequential terminal support (described in “Using sequential terminal support” on page 270), can be useful for regression testing. When you have a module that has worked for some time and is now being modified, you need to rerun your old tests to ensure that the function still works. Sequential terminal support makes it easy to maintain a "library" of old test cases and to rerun them when needed.

Sequential terminal support allows you to test programs without having to use a telecommunication device. System programmers can specify that sequential devices be used as terminals (using the terminal control table (TCT)). These sequential devices can be card readers, line printers, disk units, or magnetic tape units. They can also be combinations of sequential devices such as:

- A card reader and line printer (CRLP)
- One or more disk or tape data sets as input
- One or more disk or tape data sets as output

You can prepare a stream of transaction test cases to do the basic testing of a program module. As the testing progresses, you can generate additional transaction streams to validate the multiprogramming capabilities of the programs or to allow transaction test cases to run concurrently.

You have to do two main tasks before you can test and debug an application:

1. “Preparing the application for testing” on page 719
2. “Preparing the system for testing” on page 719

Preparing the application for testing

This list shows you what you need to consider when preparing the application and system table entries.

1. Translate, assemble or compile, and link-edit each program. Make sure that there are no error messages on any of these three steps for any program before you begin testing.
2. Use the DEBUG and EDF options on your translator step, so that you can use translator statement numbers with execution diagnostic facility (EDF) displays.
3. Use the COBOL compiler options CLIST and DMAP so that you can relate storage locations in dumps and EDF displays to the original COBOL source statements, and find your variables in working storage.
4. Create a PROFILE resource definition for your transactions to use, and make sure that the definition is installed.
5. Create a TRANSACTION resource definition for each transaction in your application, and make sure that the definitions are installed.
6. If your system does not use program autoinstall, create a PROGRAM resource definition for each program used in the application, and make sure that the definitions are installed.
7. If your system does not use program autoinstall, create a MAPSET resource definition for each map set in the application, and make sure that each definition is installed.
8. Create a FILE resource definition for each file used, and make sure that each definition is installed.
9. Build at least a test version of each of the files required.
10. Define each of the transient data queues to be used by the application.
11. Put job control DD cards in the startup job stream for each file used in the application.
12. Prepare some test data.

Preparing the system for testing

This list shows you what you need to consider when preparing the system for debugging.

1. Make sure that EDF is available in your system, by including group DFHEDF in the list you specify in the GRPLIST system initialization parameter.

2. Set up appropriate tracing options for your application. For details about setting up tracing options, see Using CICS trace.
3. Make sure that transaction dumping is enabled for all transaction dump codes, and that system dumping is enabled for all system dump codes. These are, anyway, the default settings. For information about setting up dump options, see Using dumps in problem determination.
4. Be prepared to print the dumps. Have a DFHDU730 job stream or procedure ready, and have the CICS dump data sets defined in your startup procedure.
5. Contact your system programmer for information about SDUMP data sets available on your system and access to JCL for processing them.
6. Enable CICS to detect loops, by setting the **ICVR** parameter in the SIT to a number greater than zero. Something between five and ten seconds (ICVR=5000 to ICVR=10000) is typically a workable value.
7. Generate statistics. For more information about using statistics, see Introduction to CICS statistics.

Preparing CICS Db2 programs for execution and production

This section discusses program preparation in a CICS Db2 environment.

For information on support for Java programs in the CICS Db2 environment, see Using JDBC and SQLJ to access Db2 data from Java programs .

These topics contains diagnosis, modification or tuning information.

The CICS Db2 test environment

When setting up your CICS Db2 test environment, consider how many CICS systems you want to connect to your Db2 systems.

You can connect more than one CICS system to the same Db2 system. However, the CICS Db2 attachment facility does not allow you to connect one CICS system to more than one Db2 system at a time.

You can set up production and test environments with:

- A single CICS system connected to one Db2 system
- Two or more CICS systems for production and test, connected to the same Db2 system
- Two or more CICS systems connected to two or more different Db2 systems

The first alternative, using a single CICS system for both production and test, is not recommended because applications in test could affect the performance of the production system.

The second alternative, with just one Db2 system, could be used for both test and production. Whether it is suitable depends on the development and production environments involved. Running a test CICS system and a production CICS system separately allows test failures without impacting production.

The third alternative, with, for example, one test and one production Db2 system, is the most flexible. Two CICS subsystems can run with one or more Db2 systems. Where the CICS systems are attached to different Db2 systems:

- User data and the Db2 catalog are not shared. This is an advantage if you want to separate test data from production data.
- Wrong design or program errors in tested applications do not affect the performance in the production system.
- Authorization within the test system can be less strict because production data is not available. When two CICS systems are connected to the same Db2 system, authorization must be strictly controlled, both in terms of the functions and the data that are available to programmers.

CICS Db2 program preparation

You can prepare your CICS Db2 program by using the Db2 Interactive (DB2I) interface, or you can submit your own JCL for batch processing.

About this task

The steps shown in Figure 182 on page 721 summarize how to prepare your program for execution after your application program design and coding is complete.

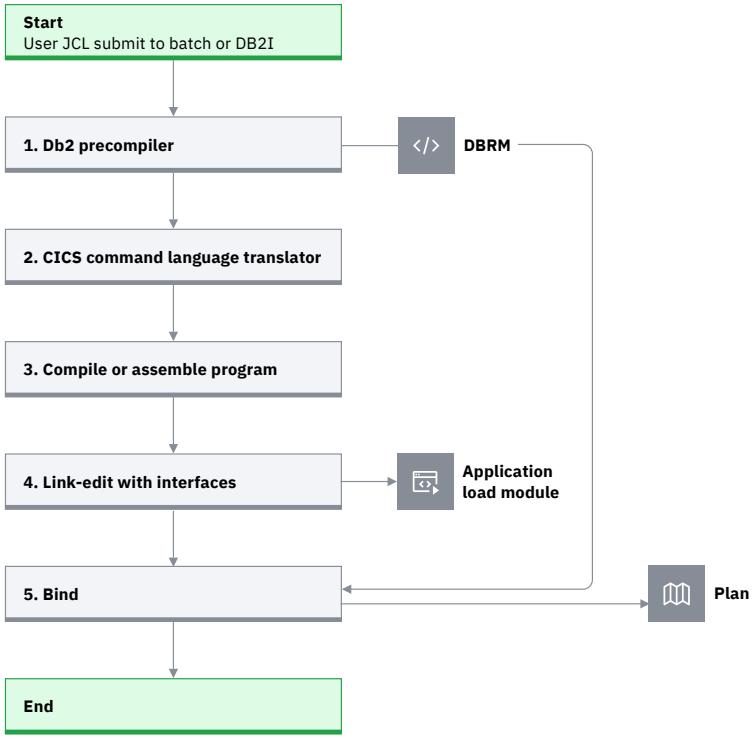


Figure 182. Steps to prepare a CICS application program that accesses Db2

For an overview of the stages in this process, see Preparing a CICS application program that accesses Db2.

When you prepare CICS application programs that access Db2:

- The Db2 precompiler (Step 1) builds a DBRM that contains information about each of the program's SQL statements. It also validates SQL statements in the program. For more information about using the Db2 precompiler, see Programming for Db2 for z/OS in Db2 for z/OS product documentation.
- If the source program is written in PL/I, the input to Step 1, the Db2 precompiler, is the output from the PL/I macro phase (if used).
- You can run Step 1, the Db2 precompiler, and Step 2, the CICS command language translator, in either sequence. The sequence shown is the preferred method, and it is the method supported by the DB2I program preparation panels. If you run the CICS command language translator first, it produces a warning message for each EXEC SQL statement it encounters, but these messages do not affect the result.
- If you use one of the Language Environment-conforming compilers (COBOL and PL/I) that has integrated the CICS translator, translation of the EXEC CICS commands (Step 2) takes place during program compilation (Step 3). For more information on the integrated CICS translator and the compilers that support it, see Translation and compilation.

- If you are running DB2 Version 7 or later and preparing a COBOL or PL/I program using one of the Language Environment-conforming COBOL or PL/I compilers, the compiler also provides an SQL statement coprocessor (which produces a DBRM), so you do not need to use the separate Db2 precompiler (Step 1). See Programming for Db2 for z/OS in Db2 for z/OS product documentation for more information on using the SQL statement coprocessor.
- If you are running DB2 Version 6 or earlier and preparing a COBOL or PL/I program, use the separate Db2 precompiler. For a COBOL program, ensure that you specify a string delimiter that is the same for the Db2 precompiler and the integrated CICS translator. The default delimiters are not compatible.
- In the link edit of the program (Step 4), include both the appropriate CICS EXEC interface module, or stub, for the language in which you are coding, and the CICS Db2 language interface module DSNCLI. The CICS EXEC interface module *must* be included first in the load module. You can link DSNCLI with your program in either 24-bit or 31-bit addressing mode (AMODE=31). If your application program runs in 31-bit addressing mode, you should link-edit the DSNCLI stub to your application with the attributes AMODE=31 and RMODE=ANY so that your application can run above 16MB.
- The bind process (Step 5) requires Db2. The bind process uses the DBRM to produce an application plan (often just called a plan) that enables the program to access Db2 data. See The bind process for more information on the bind process. A group of transactions that use the same entry thread (in other words, specified in the same DB2ENTRY) must use the same application plan. Their DBRMs must be bound into the same application plan, or bound into packages that are then listed in the same application plan.

Table 67 on page 722 shows the tasks that you need to perform to prepare a CICS Db2 program, depending on the language of the program and on your version of Db2:

Table 67. Tasks to prepare a CICS program that accesses Db2

Db2 version and program language	**Step 1 (SQL statement processing)**	**Step 2 (CICS command translation)**	**Step 3 (Program compile)**	**Step 4 (Link-edit)**	**Step 5 (Bind)**
DB2 Version 6 and Assembler	Db2 precompiler	CICS-supplied separate translator	Language compiler	Link-edit with EXEC interface and DSNCLI	Bind process
DB2 Version 6 and PL/I	Db2 precompiler	Language compiler that supports integrated CICS translator		Link-edit with EXEC interface and DSNCLI	Bind process
DB2 Version 6 and COBOL	Db2 precompiler	Language compiler that supports integrated CICS translator		Link-edit with EXEC interface and DSNCLI	Bind process
DB2 Version 6 and other languages	Db2 precompiler	CICS-supplied separate translator	Language compiler	Link-edit with EXEC interface and DSNCLI	Bind process
DB2 Version 7 (or later) and Assembler	Db2 precompiler	CICS-supplied separate translator	Language compiler	Link-edit with EXEC interface and DSNCLI	Bind process
DB2 Version 7 (or later) and PL/I	Language compiler that supports integrated CICS translator and SQL statement coprocessor			Link-edit with EXEC interface and DSNCLI	Bind process
DB2 Version 7 (or later) and COBOL	Language compiler that supports integrated CICS translator and SQL statement coprocessor			Link-edit with EXEC interface and DSNCLI	Bind process
DB2 Version 7 (or later) and other languages	Db2 precompiler	CICS-supplied separate translator	Language compiler	Link-edit with EXEC interface and DSNCLI	Bind process

You prepare your CICS Db2 program by using the Db2 Interactive (DB2I) interface or by submitting your own JCL for batch execution.

- Db2 Interactive (DB2I) interface: DB2I provides panels to precompile, compile or assemble, and link-edit an application program and to bind the plan. For details about application program preparation, see the Programming for Db2 for z/OS in Db2 for z/OS product documentation.
- User JCL submitted to batch execution: Members DSNTEJ5C and DSNTEJ5P in the Db2 library, SDSNSAMP, contain samples of the JCL required to prepare COBOL and PL/I programs for CICS.

If you prepare your program for execution while CICS is running, you might need to issue a CEMT NEWCOPY command to make the new version of the program known to CICS.

CICS SQLCA formatting routine

DSNTIAR, the IBM-supplied SQLCODE message formatting procedure, lets you send "SQL messages online" to your application.

With DB2 Version 3.1, DSNTIAR was split into two front-end modules (DSNTIAC and DSNTIAR) and a runtime module (DSNTIA1). DSNTIAC is used for CICS applications and DSNTIAR for other Db2 interfaces. This change removed the need, previous to DB2 Version 3.1, to relink-edit your application modules every time a change is made to DSNTIAR, either by change of release or by applying maintenance. If you have applications that have previously been link-edited with DSNTIAR, you should consider link-editing them again using DSNTIAC instead, which will provide performance improvements and isolate them from changes to DSNTIAR.

The CICS front-end part, DSNTIAC, is supplied as a source member in the Db2 library SDSNSAMP.

The necessary program definitions for DSNTIAC and DSNTIA1 are provided in IBM supplied group DFHDB2 on the CSD. You must add the SDSNLOAD library to the CICS DFHRPL concatenation (after the CICS libraries) so that DSNTIA1 can be loaded.

What to bind after a program change

If you change a program, you must prepare it and rebind it before it can be used.

About this task

For an overview of the bind process, see The bind process. For an overview of plans and packages, see Plans, packages and dynamic plan exits.

Imagine you have a CICS transaction that consists of four program modules: module 1 is the main module. Module 1 calls module 2. Module 1 also calls module 3, and module 3 calls module 4. It is not unusual that the number of modules is high in a real transaction. Assuming that at least one SQL statement changed in one of the modules, you must perform the following procedure to prepare the program and to make the transaction executable again.

Procedure

1. Precompile the program on Db2.
2. Translate the program using the CICS translator.
3. Compile the host language source statements.
4. Link-edit.
5. **If the DBRM for program C was bound into a package**, bind that package using the new DBRM, and all the application plans that use program C will automatically locate the new package.
6. **If the DBRM for program C was bound directly into any application plans**, locate all the application plans that include the DBRM for program C. Bind all the application plans again, using the DBRMs for all the programs directly bound into them, to get new application plans. For the programs that were not changed, use their old DBRMs. Note that you *cannot* use the REBIND subcommand, because input to REBIND is the plan and *not* the DBRMs.

Note: If you have not used packages before, note that using packages simplifies the rebinding process. You can bind each separate DBRM as a package and include them in a package list. The package list can be included in a PLAN. You can then use the BIND PACKAGE command to bind the DBRMs for any changed programs, instead of using the BIND PLAN command to bind the whole application plan. This provides increased transaction availability and better performance. See Using Db2 packages for more information on using packages.

Bind options and considerations for programs

When binding multiple programs into an application plan, be aware of the way in which Db2 uses time stamps.

For each program, the Db2 precompiler creates the following:

- The Db2 precompiler creates a DBRM with a time stamp of Td *x*. For example, Td1 for the first program, Td2 for the second program, and so on.
- The Db2 precompiler creates a modified source program with a time stamp of Ts *x* in the SQL parameter list. For example, Ts1 and Ts2, if two programs are involved.

At bind time, the DBRM for each program is bound into the package or plan that you have specified. In addition, Db2 updates its catalog table SYSIBM.SYSDBRM with one line for each DBRM, together with its time stamp. At execution time, Db2 checks the time stamps for each SQL statement, and returns a -818 SQL code if the time stamp for the DBRM and the time stamp it has placed in the source program are different (in our example, if Td1 and Ts1 are different, or Td2 and Ts2 are different). To avoid -818 SQL codes, use one of the following strategies:

- Bind all programs into packages, and list these packages in the application plan. When a program changes, precompile, compile, and link-edit the program, and bind it into a package again.
- If you bind any programs directly into application plans, ensure that for every new or changed program, you precompile, compile, and link-edit the program, then bind all the application plans that involve that program, using the DBRMs from all the programs directly bound into those plans. Use the BIND command, not the REBIND command, to do this.

When you bind a plan, a number of options are available. Almost all bind options are application dependent and should be taken into account during the application design. You should develop procedures to handle different BIND options for different plans. Also, the procedures should be able to handle changes in BIND options for the same plan over time.

The following sections describe some specific recommendations for BIND options with CICS.

RETAIN

RETAIN means that BIND and EXECUTE authorities from the old plan are not changed.

When the RETAIN option is not used, all authorities from earlier GRANTs are REVOKED. The user executing the BIND command becomes the creator of the plan, and all authorities must be reestablished by new GRANT commands.

This is why it is recommended that you use the RETAIN option when binding your plans in the CICS environment.

Isolation level

The isolation level is specified for the complete plan. You are recommended to use Cursor Stability (CS) unless there is a specific need for using Repeatable Read (RR). Using CS allows a high level of concurrency and reduces the risk of deadlocks.

Note that the isolation level is specified for the complete plan. This means that if RR is necessary for a specific module in CICS, then all the DBRMs included in the plan must also use RR.

Also, if for performance reasons you decide to group a number of infrequently used transactions together to use the same DB2ENTRY and let them use a common plan, then this new plan must also use RR, if just one of the transactions requires RR.

Plan validation time

A plan is bound with VALIDATE(RUN) or VALIDATE(BIND). VALIDATE(RUN) is used to determine how to process SQL statements that cannot be bound.

If a statement must be bound at execution time, it is rebound for each execution. This means that the statement is rebound for every new unit of work (UOW).

Binding a statement at execution time can affect performance. A statement bound at execution time is rebound for each execution. That is, the statement must be rebound after each syncpoint. It is not recommended that you use this option with CICS.

Note that using dynamic SQL does not require VALIDATE(RUN). Nevertheless, dynamic SQL implies that a statement is bound at execution.

You should use VALIDATE(BIND) in a CICS Db2 environment.

ACQUIRE and RELEASE

The ACQUIRE and RELEASE parameters change from plan to plan over time because they are related to the transaction rate.

The general recommendations for these parameters are described in Selecting BIND options for optimum performance.

CICS Db2 program testing and debugging

The tools that are typically used in a CICS environment can be used to test and debug a CICS application program that accesses Db2. These include: the execution diagnostic facility (EDF), the CICS auxiliary trace, and transaction dumps.

For information about these and other problem determination processes, see Troubleshooting Db2.

Going into production: checklist for CICS Db2 applications

This checklist shows the tasks you need to perform after designing, developing, and testing an application, in order to put the application into production.

About this task

These tasks are highly dependent on the standards you have used in the test system. For example, the tasks to be performed are different if:

- There are separate Db2 systems for test and production.
- Only one Db2 is used for both test and production.

The following discussion assumes that you use separate Db2 and CICS subsystems for test and production.

Going into production implies performing the following activities:

Use DDL to prepare production databases
All DDL operations must run on the production Db2 system, using DDL statements from the test system as a base. Some modifications are probably needed, for example, increasing the primary and secondary allocations, as well as defining other volume serial numbers and defining a new VCAT in the CREATE STOGROUP statements.

Prepare DCLGEN
For COBOL and PL/I programs, you may have to run DCLGEN operations on the production Db2 system, using DCLGEN input from the test Db2 system.

Depending on the option taken for compilations (if no compilations are run on the production system), an alternative could be to copy the DCLGEN output structures from the test libraries into the production libraries. This keeps all information separate between test and production systems.

Precompile for the production system

If you have bound your programs into packages on the test system, you do not need to perform this step. You can move the packages straight to the production system. See "Produce an application plan for the production system" for details of how to do this. However, if you want to bind your programs directly into application plans, or if you want to bind the programs into packages on the production system, you need to put the DBRMs for the programs on the production system. You can either:

- Precompile CICS modules containing EXEC SQL statements on the production system, or
- Copy DBRMs from the test system to the production system libraries.

Compile and link-edit for the production system

To produce load modules:

- If the DBRMs were produced by precompiling on the production system, compile and link-edit the CICS modules on the production system; or
- If the DBRMs were copied, or if you are moving packages from the test system to the production system, copy the load modules from the test system to the production system libraries.

Table 68 on page 726 shows a procedure you can use to copy a changed load module from a test system to a production system library, replacing the old version of the load module.

Table 68. Moving a changed program from the test environment to the production environment

Test system	Production system	Notes
	USER.PROD.LOADLIB(PGM3)	The original load module
USER.TEST.LOADLIB(PGM3)		The test load module
	USER.OLD.PROD.LOADLIB(PGM3)	The old version of the program is placed in other production library
	USER.PROD.LOADLIB(PGM3)	The new version of the program is placed in the production library

By selecting the production run library using the proper JCL, you can run either the old version or the new version of the program. Then the correct version of the package is run, determined by the consistency token embedded in the program load module.

Produce an application plan for the production system

If you have bound your programs into packages on the test system, you can copy the packages to the production system, including them in a collection that is listed in the application plan. When you copy a package to the production system, you do not need to bind the application plan on the production system again, as long as the package is included in a collection that is already listed in the application plan.

Table 69 on page 726 shows a procedure you can use to copy a changed package from a test system to a production system library, replacing the old version of the package. This example uses the VERSION keyword at precompile time to distinguish the different versions of the packages. For a full explanation and use of the VERSION keyword, see Programming for Db2 for z/OS in Db2 for z/OS product documentation.

Table 69. Moving a changed package from the test environment to the production environment

Test system	Production system	Notes
	location_name. PROD_COLL.PRG3.VER1	The old version of the package
location_name. TEST_COLL.PRG3.VER2		A new version of the package is bound on the test system and then copied to the production system

Table 69. Moving a changed package from the test environment to the production environment (continued)		
Test system	**Production system**	**Notes**
	location_name. PROD_COLL.PRG3.VER1	The old version is still in the production collection
	location_name. PROD_COLL.PRG3.VER2	The new version is placed in the production collection

If you want to bind your programs directly into application plans, or if you want to bind the programs into packages on the production system, you must perform the bind process on the DBRMs that you have placed on the production system. If you are binding your programs directly into application plans, you must then bind all the application plans on the production system that involve those programs. See The bind process for more information on the bind process. Note that due to various factors, such as the sizes of tables and indexes, comparing the EXPLAIN output between test and production systems can be useless. Nevertheless, it is recommended that you run EXPLAIN when you first bind a plan on the production system, to check the Db2 optimizer decisions.

GRANT EXECUTE
You must grant users EXECUTE authority for the Db2 application plans on the production system.

Tests
Although no further tests should be necessary at this point, stress tests are useful and recommended to minimize the occurrence of resource contention, deadlocks, and timeouts and to check that the transaction response time is as expected.

CICS definitions
To have new application programs ready to run, update the following RDO definitions on the CICS production system.

- RDO transaction definitions for new transaction codes
- RDO program definitions for new application programs and maps
- SIT for specific Db2 requirements, if it is the first Db2-oriented application going into production
- RDO DB2ENTRY and DB2TRAN definitions for the applications. RDO DB2CONN definition if it is the first Db2-oriented application going into production. When defining the new transactions and application plans in the DB2ENTRY you can use unprotected threads to get detailed accounting and performance information in the beginning. Later, you can use protected threads as needed.

In addition, if RACF is installed, you need to define new users and Db2 objects.

Tuning a CICS application that accesses Db2

CICS applications that access Db2 must be tuned before they move to production and periodically whist they are in production.

About this task

When moving a CICS application that accesses Db2 to production, add these checks to those already performed for CICS :

- Check that all the application programs that make Db2 requests are threadsafe. If they are, you will be exploiting the open transaction environment (OTE), and improving the performance of the application. See Enabling CICS Db2 applications to use OTE through threadsafe programming for an explanation of how application programs work in the open transaction environment.
- Ensure that the number and type of SQL statements used meet the program specifications (use the Db2 accounting facility).
- Check if the number of get and updated pages in the buffer pool is higher than expected (use the Db2 accounting facility).

- Check that planned indexes are being used (use EXPLAIN), and that inefficient SQL statements are not being used.
- Check if DDL is being used and, if so, the reasons for using it (use the Db2 accounting facility).
- Check if conversational transactions are being used.

  Determine whether pseudoconversational transactions can be used instead. If conversational design is needed, check the Db2 objects that are locked across conversations. Check also that the number of new threads needed because of this conversational design is acceptable.
- Check the locks used and their duration.

  Make sure that tablespace locks are not being used because of incorrect or suboptimal specification of, for example:

  – LOCK TABLE statement
  – LOCKSIZE=TS specification
  – ISOLATION LEVEL(RR) specification
  – Lock escalation.

  This information is available in the catalog tables, except for lock escalation, which is an installation parameter (DSNZPARM).
- Check the plans used and their sizes. Even though the application plans are segmented, the more DBRMs used in the plan, the longer the time needed to BIND and REBIND the plans in case of modification. Try to use packages whenever possible. Packages were designed to solve the problems of:

  – Binding the whole plan again after modifying your SQL application. (This was addressed by dynamic plan selection, at the cost of performance.)
  – Binding each application plan if the modified SQL application is used by many applications.

When this tuning is complete, use the expected transaction load to decide on the DB2ENTRY definitions required, and the number of threads required. Check also the impact of these transactions on the Db2 and CICS subsystems.

When tuning a CICS application that accesses Db2 in production:

- Check that the CICS applications use the planned indexes by monitoring the number of GET PAGES in the buffer pool (use the Db2 accounting facility). The reasons for an index not being used may be that the index has been dropped, or that the index was created after the plan was bound.
- Use the lock manager data from the accounting facility to check on suspensions, deadlocks, and timeouts.

CICS application build automation with the CICS build toolkit

The CICS TS build toolkit (CICS build toolkit) provides a command line interface for build automation of CICS projects, including CICS bundles, CICS applications, CICS application bindings, and CICS platforms.

The CICS build toolkit also supports projects that are referenced by CICS bundles, including OSGi Application Projects, OSGi Bundle Projects, Enterprise Application Projects, and Dynamic Web Projects, and can take pre-built OSGi bundles and Liberty applications as input.

Java 11Java 17Java 21 The CICS build toolkit only supports Java 8. If your applications must be compiled against Java 11 or Java 17, consider building them using the CICS-provided Gradle or Maven plug-ins.

Build automation

In a continuous integration environment, a build script runs automatically when a developer checks in updates to their application. The script checks out the latest application from source control, and calls the CICS build toolkit to build the projects that form the application. The script checks the result of the build for errors and, if appropriate, copies the built projects to a suitable location, such as an artifact repository or a staging area on zFS.

Variable substitution

To facilitate the deployment of the same application to different environments (for example, development, quality assurance, and production), you can use the CICS build toolkit to resolve variables in CICS bundles. A script typically uses the built projects together with a properties file that defines values for variables in the target environment.

Supported operating systems

The CICS build toolkit is supported on z/OS, Linux®, and Microsoft Windows operating systems.

Preparing to use the CICS build toolkit

How to install, upgrade, and uninstall the CICS build toolkit, including the Java requirements and Eclipse workspace considerations for running the CICS build toolkit.

Installing the CICS build toolkit

1. Download the most recent version of the *cicsbt-#######.zip* file from the IBM CICS Transaction Server build toolkit downloads website.
2. Transfer the *cicsbt-#######.zip* file in binary to the system on which you run the CICS build toolkit.
3. Extract the *cicsbt-#######.zip* file to create the *cicsbt* directory. Issue the command **unzip cicsbt-#######.zip** or, if unzip is unavailable, issue the command **jar -xf cicsbt-#######.zip**.
4. Ensure that all users requiring access to CICS build toolkit have read permissions for all directories and subdirectories in the `cicsbt` path, and execute permission for the shell scripts *cicsbt*, *cicsbt.bat*, or *cicsbt_zos*. For example on z/OS:

```
chmod -R 755 cicsbt_install_dir
```

Upgrading the CICS build toolkit

To upgrade the CICS build toolkit to a newer version, delete the *cicsbt* directory and follow the installation instructions that are contained in the readme file for the new version.

Uninstalling the CICS build toolkit

To uninstall the CICS build toolkit, delete the *cicsbt* directory.

Running the CICS build toolkit

The CICS build toolkit requires a Java 8 compatible Software Development Kit (SDK), either 32-bit (31-bit on z/OS) or 64-bit.

You must set the following environment variables: *JAVA_HOME* to refer to the SDK directory, and either *HOME* on z/OS and UNIX, or *USERPROFILE* on Windows to refer to the users home directory.

For an overview of the available command line options, on Windows run cicsbt.bat, on Linux run cicsbt, and on z/OS run cicsbt_zos.

Managing Eclipse workspaces

When you run the CICS build toolkit, it creates a temporary Eclipse workspace in your user home directory, in which it builds the specified CICS bundles or applications. Do not delete the workspace that is created by the CICS build toolkit while it is in use. This action causes the build process to fail and corrupts your built artifacts.

If you want to run multiple instances of the CICS build toolkit simultaneously from one user account, you can use the **--workspace** option to specify a workspace directory.

Note: Ensure that you delete the workspace directory after use, and that you use a different source directory for each workspace. This is important because the CICS build toolkit writes to the source directory during the build process.

Building a CICS bundle, application, application binding, or platform

You can automate the build of CICS projects, including CICS bundle projects, application projects, application binding projects, and platform projects by writing a build script that calls the CICS build toolkit.

Before you begin

CICS build toolkit needs write access to the source projects as it might modify them as part of the build process. Make sure that you use a temporary copy of the source projects that can be discarded after the build completes. For more information about preparation, see "Preparing to use the CICS build toolkit" on page 729.

If you are building Java projects from source, you need to know the target CICS release and have any additional Java library dependencies available. Alternatively, you can copy built versions of those Java projects, for example *.jar .wab .eba* or *.ear* files, into the CICS bundle root directory before you run the CICS build toolkit.

About this task

Use this procedure after you develop your applications and before you deploy your applications as part of a continuous delivery model. Run *cicsbt* as part of your build automation. Alternatively, run *cicsbt* at the shell on UNIX or Linux, or *cicsbt.bat* at the command prompt on Windows.

Procedure

1. Specify the **--input** option to indicate the location of the CICS projects and any referenced projects to be built.
2. Specify the **--build** option to indicate which CICS bundle projects, application projects, application binding projects, and platform projects to build from the source location.

   When you build an application binding project, the CICS build toolkit looks for associated applications and platforms in the input directory. If an associated application is found, it is also built. If no associated applications or platforms are found, the build completes with warnings.
3. Specify the **--output** option to indicate the output directory.
4. To build CICS projects that reference Java projects, specify a target platform by using the **--target** option. It is not necessary to build Java projects if they are already built and included in the CICS bundle root directory.

   There are several ways to specify a target platform:

   a. Specify a built-in target that corresponds to the lowest version of CICS that your application runs on. You can view a list of available built-in targets by specifying **--target** with no arguments.

   b. Provide the path to an Eclipse `.target` file created by CICS Explorer or Eclipse. Ensure that the user that runs the CICS build toolkit has access to all libraries and directories that are referenced by the `.target` file.
5. To use an alternative character set for your built artifacts, specify the **--encoding** option.

   Some CICS bundle artifacts must be in the character set that is specified by **LOCALCCSID** CICS SIT parameter. CICS build toolkit uses character set *cp037* by default.

Results

The CICS projects and any referenced projects are built in the output directory within subdirectories that follow the naming convention *symbolic.name_version*, for example, `com.ibm.cics.test.bundle_1.0.0`.

Example

This example shows an output directory where only applications and bundles are built, with no associated bindings:

```
/output/dir/
    applications/
        my.application_1.0.0
        my.other.application_1.0.0
    bundles/
        my.application.bundle_1.0.0
```

This example shows an output directory where a binding that is associated with the platform *testplatform* is built. Application *my.application_1.0.0* and bundle *my.application.bundle_1.0.0* are also built, but are not associated with the binding:

```
/output/dir/
    applications/
        my.application_1.0.0
    bundles/
        my.application.bundle_1.0.0
    testplatform/
        applications/
            my.other.application_1.0.0
        bindings/
            my.other.application.test.binding_1.0.0
        bundles/
            my.other.application.bundle_1.0.0
            my.other.application.another.bundle_1.0.0
```

The output directory that is created by the CICS build toolkit has the same structure as the platform home directory (PLATHOME). For more information about the platform home directory structure, see Platform directory structure in z/OS UNIX.

This example shows a CICS build toolkit call that builds the CICS application binding `my.application.binding(1.0.1)` and any associated applications that are found in the input directory:

```
cicsbt --input my/source/dir/* \
       --build my.application.binding(1.0.1) \
       --output /my/output/dir
```

To use this example on Windows, enter all parameters on the same line as the CICS build toolkit call. Paths that contain spaces must be within quotation marks.

What to do next

If your built projects contain variables, you must resolve the variables before they are installed into CICS. For more information about variable substitution, see “Resolving variables in a CICS bundle” on page 732.

After you run the CICS build toolkit, use your build script to copy the built CICS projects to an artifact repository or a staging location on zFS, ready for deployment.

Note: If the artifacts are transferred to zFS by FTP, you must make the transfers in binary mode to ensure that their contents are preserved.

Resolving variables in a CICS bundle

If your built artifacts contain unresolved variables, use the CICS build toolkit to resolve these variables before you deploy these artifacts to your target environment.

Before you begin

Ensure that you build the targeted projects in a previous run of the CICS build toolkit before you attempt to resolve variables in your built artifacts. For more information about the build process, see “ Building a CICS bundle, application, application binding, or platform ” on page 730.

About this task

Use this procedure after you build your applications and as part of your deployment phase in a continuous delivery model. Run *cicsbt* as part of your automated deployment process. Alternatively, run *cicsbt* at the shell (UNIX or Linux) or *cicsbt.bat* at the command prompt (Windows).

Procedure

1. Specify the **--resolve** option to indicate the location of the CICS projects you want to resolve.

   Note: The path specified by the **--resolve** option should match the top-level directory that was specified by the **--output** option when the project was initially built.
2. Optional: If you are resolving variables in a stand-alone bundle, specify the **--properties** option to indicate the location of a stand-alone `variables.properties` file.

   This step is not required if you are resolving variables in a bundle associated with a platform or application binding. Variables will be resolved using the *variables.properties* file in the platform or application binding root folder.
3. Optional: If you specified the **--encoding** option when building the artifacts, specify the same **--encoding** value when resolving to ensure that the CICS build toolkit can read the artifacts correctly.
4. Specify the **--output** option to indicate the output directory.

   The directory structure of the resolved output will match the directory structure of the input that is supplied by the **--resolve** option.

Results

Variables in the CICS project are fully resolved and resolved projects are placed in the specified output directory.

Building and resolving in the same script

In most cases, building artifacts and resolving variables will occur at different points in the delivery cycle. For simpler applications, it is possible to build and resolve within the same script.

This example calls the CICS build toolkit to build the latest versions of the CICS bundles, with ID of *OSGiBundleProject* and *AnotherBundleProject*, from two different input directories. This call uses the CICS Transaction Server Version 5.1 target to build the referenced Java projects. The second call then resolves the variables within the targeted bundles:

```
cicsbt --input my/source/dir/* other/source/dir/* \
       --build OSGiBundleProject AnotherBundleProject \
       --target com.ibm.cics.explorer.sdk.runtime51.target \
       --output unresolved/output/dir

cicsbt --resolve unresolved/output/dir \
       --output resolved/output/dir
```

To use this example on Windows, enter all options on the same line as the CICS build toolkit call. If your paths contain spaces, ensure that they are within quotation marks.

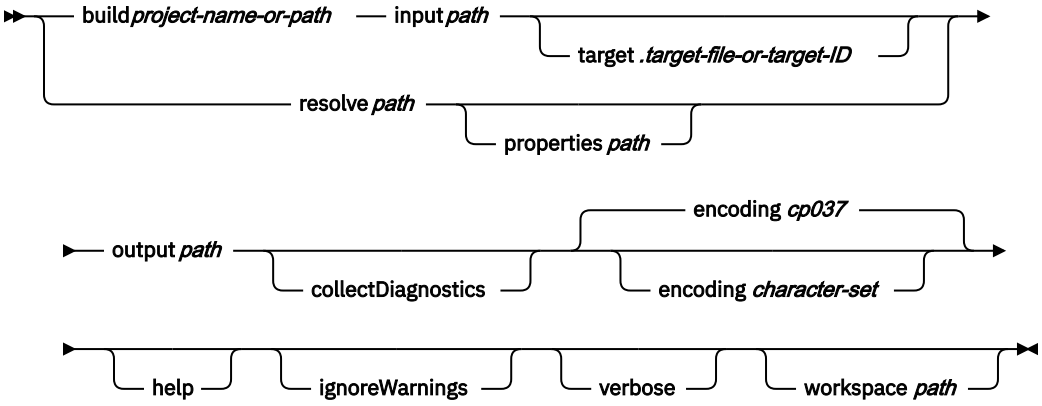
What to do next

Proceed to the next step of your deployment process. For example, write a script to move the resolved projects to the CICS platform home directory and deploy the bundle or application resources in CICS by using the DFHDPLOY utility.

CICS build toolkit command line options

The CICS build toolkit command line options are listed, with examples of building CICS bundles and resolving variables in built artifacts.

Summary of command line options



List of command line options

--build, -b *project-name-or-path*

Specifies which CICS bundle projects, application projects, application binding projects, and platform projects to build from the directories that are specified by the **`--input`** option.

A project can be specified with either the symbolic name and version, for example `MyBundleProject(1.0.1)`, or path to the project, for example `/u/user/applications/testapplication`. If you specify a symbolic name but do not specify a version, the CICS build toolkit builds the highest version of the project that is available. To build multiple projects in a single call, specify **`--build`** with multiple arguments, delimited by spaces. Alternatively, specify **`--build`** with no argument to build all projects in the path specified by **`--input`**.

The **`--build`** and **`--resolve`** options are mutually exclusive.

--collectDiagnostics, -c

Requests the collection of diagnostics data after a build completes.

The diagnostics data is stored in a *.zip* file. This data is written to the working directory in use when the build was started, or to your system's temporary directory if the working directory is read-only. You can send this diagnostics data to IBM to help diagnose problems.

Alternatively, you can set the environment variable *JAVA_TOOL_OPTIONS* to `-Dcollect.diagnostic.data` to request the collection of diagnostics data.

--encoding, -e *character-set*

The IANA character set name that represents the default CCSID for the CICS region into which the bundle will be installed. If the option is not specified, the IANA character set *cp037* is used. The encoding is used to convert Atom configuration files, JVM profiles and Node.js profiles that need representing in EBCIDIC.

For example, if the **LOCALCCSID** system initialization parameter for the CICS region is *00285*, specify `--encoding ibm285`. For a list of CCSIDs and corresponding character set names, see CICS-supported conversions.

--help, -h
Prints a list of the available options.

--input, -i *path*
Specifies the path of the project that you want to build.

Multiple paths can be specified by using a space delimiter. You can also use an asterisk at the end of a path to specify all projects in that directory to build multiple projects in a single call, for example */path/to/top/level/directory/**.

--ignoreWarnings, -g
Specifies that warnings are to be ignored. RC=0 is returned if code warnings are found.

If both `-v` and `-g` are specified, this message is returned when warnings are found: `Warnings were generated in the build, which have been ignored.`

--output, -o *path*
Specifies the path in which to place built or resolved projects.

The CICS build toolkit creates subdirectories for applications and bundles, built according to their symbolic name and version. If only applications or bundles are built, they are placed in `/applications` and `/bundles` subdirectories. If one or more application bindings are built, then all associated artifacts are placed in a directory structure that includes the platform name. Any applications or bundles not associated with a binding are placed in subdirectories as normal.

--properties, -p *path*
Specifies the path of a properties file outside of the built projects' directories.

You can use the **`--properties`** option to resolve variables in stand-alone bundles. Bundles within applications or platforms are not supported by this option. A properties file specified through the **`--properties`** option takes precedence over properties in the bundle.

For more information about how to define and use variables, see Variables and properties files definition.

--resolve, -r *path*
Specifies the path of the directory structure that is produced by a previous run of the CICS build toolkit.

This option replaces variables within the CICS bundle parts with corresponding values specified in the *variables.properties* file in the bundle root folder. For a stand-alone CICS bundle, values can also be specified in a *variables.properties* file that is specified by the **`--properties`** option, and take precedence. For a CICS bundle that is part of an application, values can also be specified in a *variables.properties* file in the application binding root folder, and take precedence. The *variables.properties* file must be in ISO-8859-1 (Latin 1) character set encoding.

The **`--build`** and **`--resolve`** options are mutually exclusive.

--target, -t *.target-file-or-target-ID*
Specifies which target platform to use.

The target platform defines the Java libraries (APIs) that are required to build referenced OSGi Bundle Projects and OSGi Application Projects. The target platform can be predefined, such as a CICS release, or can be an Eclipse *.target* file that is created by the user.

The following target platforms are available by default:

- *com.ibm.cics.explorer.sdk.runtime41.target*
- *com.ibm.cics.explorer.sdk.runtime42.target*
- *com.ibm.cics.explorer.sdk.runtime51.target*
- *com.ibm.cics.explorer.sdk.runtime52.target*
- *com.ibm.cics.explorer.sdk.runtime53.target*
- *com.ibm.cics.explorer.sdk.runtime54.target*
- *com.ibm.cics.explorer.sdk.runtime55.target*

- *com.ibm.cics.explorer.sdk.runtime56.target*
- *com.ibm.cics.explorer.sdk.web.liberty51.target*
- *com.ibm.cics.explorer.sdk.web.liberty52.target*
- *com.ibm.cics.explorer.sdk.web.liberty53.target*
- *com.ibm.cics.explorer.sdk.web.liberty54.target*
- *com.ibm.cics.explorer.sdk.web.liberty55.target*
- *com.ibm.cics.explorer.sdk.web.liberty56.target*

--verbose, -v
Provides extra details to the console about the progress of the CICS build toolkit.

If both `-v` and `-g` are specified, this message is returned when warnings are found: `Warnings were generated in the build, which have been ignored.`

--workspace, -w *path*
Specifies the path of the Eclipse workspace that the CICS build toolkit uses.

If this option is omitted, the CICS build toolkit creates a temporary Eclipse workspace that is deleted once processing is complete. If you use this option to specify a workspace, it is not automatically deleted on completion. For more information, see Managing Eclipse workspaces.

Example usage

This example builds the CICS bundle with ID *com.ibm.cics.server.examples.jcics* and version *1.0.1*, and uses the CICS Transaction Server Version 5.4 target to build the referenced Java projects:

```
cicsbt --input my/source/dir/* \
--build "com.ibm.cics.server.examples.jcics(1.0.1)" \
--target com.ibm.cics.explorer.sdk.runtime54.target \
--output my/output/dir
```

This example builds the latest versions of the CICS bundles with ID of *OSGiBundleProject* and *AnotherBundleProject*, from two different input directories, and uses the CICS Transaction Server Version 5.1 target to build the referenced Java projects:

```
cicsbt --input my/source/dir/* other/source/dir* \
--build OSGiBundleProject AnotherBundleProject \
--target com.ibm.cics.explorer.sdk.runtime51.target \
--output my/output/dir
```

This example resolves variables for a previously built project:

```
cicsbt --resolve unresolved/output/dir \
--output resolved/output/dir
```

This example resolves variables for a previously built bundle project by using a stand-alone properties file:

```
cicsbt --resolve unresolved/output/dir \
--output resolved/output/dir \
--properties props/my.properties
```

To use these examples on Windows, enter all options on the same line as the CICS build toolkit call. If your paths contain spaces, ensure that they are within quotation marks.

CICS build toolkit return codes

The CICS build toolkit provides return codes to indicate the outcome of a build, with a return code of 0 for successful completion, 1 or 2 for warnings, and 9-16 inclusive for errors.

When an error occurs, the CICS build toolkit can halt processing or complete, depending on the severity of the error. More diagnostic information can be obtained by specifying the **-v** parameter.

CICS build toolkit return code	Description
0	The operation completed successfully, or artifacts were built with warnings when the `--ignorewarnings` option was specified on the CICS build toolkit command.
1	Warning. Some source projects were not imported.
2	Warning. Artifacts were built, with warnings.
9	Error. Invalid options were supplied.
10	Error. No source projects were imported.
11	Error. The artifact that is specified to be built was not found in the source projects.
12	Error. The build step failed.
13	Error. A problem occurred with the supplied target platform.
14	Error. No source projects were found.
15	Error. The resolve step failed.
16	Error. No unresolved projects were found.

Automate the deployment and undeployment of CICS bundles and applications with the DFHDPLOY utility

The DFHDPLOY utility provides a set of commands that you can use in a script to deploy, undeploy, and set the state of CICS bundles and cloud enabled CICS applications.

Overview

The DFHDPLOY utility can be started from JCL and integrated with your existing deployment and automation procedures. Use the DFHDPLOY utility with the CICS build toolkit to automate build and deployment of CICS bundles and cloud enabled CICS applications and as part of a continuous delivery environment. See “CICS application build automation with the CICS build toolkit” on page 728.

The DFHDPLOY utility has commands that perform the following functions:

- Connecting to a CICSplex
- Deploying, undeploying, and setting the state of a CICS bundle
- Deploying, undeploying, and setting the state of a cloud enabled CICS application

The DFHDPLOY utility can be used in conjunction with UrbanCode® Deploy to create a continuous delivery environment. For more information, see the CICS TS plug-in for UrbanCode Deploy documentation on the UrbanCode website.

Usage

DFHDPLOY does not support the installation of a bundle or application defined by an existing resource definition.

Note: Specify `REGION=100M` or more when you run the DFHDPLOY job to avoid running out of memory while the job is processing.

Security

The user running the DFHDPLOY utility requires read access to the zFS directories specified by the BINDDIR , BUNDLEDIR , and APPLDIR command options for any LPAR in which the utility might run.

If security is enabled in the CICSplex, the user running the DFHDPLOY utility requires access to various CICSPlex SM resources protected by the external security manager, such as RACF.

For deploying and undeploying bundles, the following access levels are required:

- ALTER access for BAS.APPLICTN, OPERATE.APPLICTN.
- UPDATE access for BAS.DEF, CSD.DEF.
- READ access for CONFIG.DEF, OPERATE.FILE, OPERATE.REGION.

For deploying and undeploying applications, the following access levels are required:

- ALTER access for CLOUD.APPLICATION.
- UPDATE access for CLOUD.DEF.
- READ access for CLOUD.PLATFORM, CONFIG.DEF, OPERATE.TASK.

For more information about implementing CICSPlex SM security, see Determining who requires access to CICSPlex SM resources.

Compatibility

DFHDPLOY can connect to a CICSPlex SM CMAS at version 5.1 and above, but some functions are limited in earlier releases of CICS TS for z/OS, Version 5.

- Application support was added in CICS TS for z/OS, Version 5.1.
- The AVAILABLE and UNAVAILABLE states for bundles and applications were added in CICS TS for z/OS, Version 5.2.
- Bundle PHASEIN support was added in CICS TS for z/OS, Version 5.3.

Sample program DFH$DPLY

The DFH$DPLY sample program contains annotated DFHDPLOY JCL to deploy, undeploy, and optionally set a sample bundle and application in a CICSplex. The sample is supplied in CICSTS56.CICS.SDFHSAMP.

DFHDPLOY utility commands

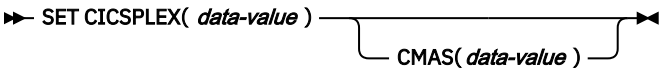
Commands are read from the JCL SYSIN data definition and messages are written to the SYSTSPRT data definition. You can specify a command across multiple lines. A semicolon (;) indicates the end of a command. Comments starting with an asterisk (*) as the first non-blank character are ignored. Leading and trailing spaces are ignored.

When an application is transitioned from AVAILABLE or UNAVAILABLE to any other state, DFHDPLOY waits for tasks associated with the application operation to complete during the UNAVAILABLE stage.

If a task fails to complete within the specified TIMEOUT value, the job fails with warnings and a return code of 4 or 8. For more information about the differences between return codes, see DFHDPLOY utility return codes.

The SET CICSPLEX command

Define the CICSplex to connect to and use to manage applications and bundles in subsequent commands. DFHDPLOY can connect to a CICSPlex SM CMAS at version 5.1 and above.



Options

CICSPLEX ***(data-value)***
Specifies the CICSplex (up to 8 characters) to connect to and use for subsequent commands in the input stream.

CMAS ***(data-value)***
Optionally specifies the CMAS (up to 8 characters) that processes DFHDPLOY commands.

Use this parameter to separate CICSPlex SM API calls over multiple CMASes within a management network. For example, you can use this parameter to distribute workload to other CMASes in addition to the maintenance point CMAS.

Note: The maintenance point CMAS must be active for DFHDPLOY to run successfully.

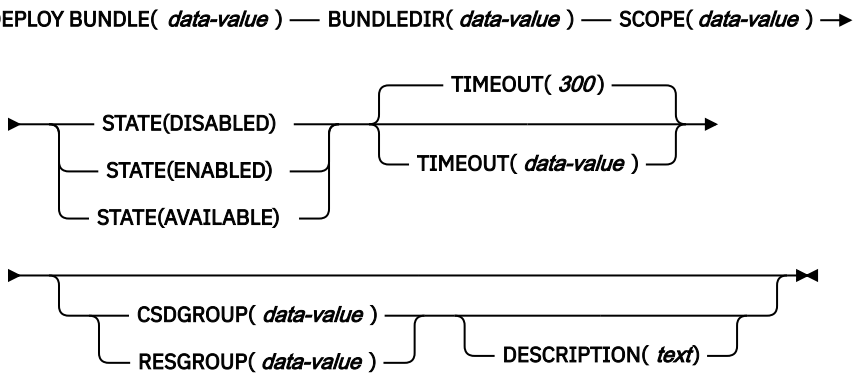
If this option is not specified, the CMAS that processes DFHDPLOY commands is determined in the same manner as in an EXEC CPSM CONNECT API call: typically, the most recently started CMAS within the CICSplex that is running on the same MVS image and the same version of CICSPlex SM. For more information about connecting to a CMAS, see The connection process.

Note: The DFHDPLOY JOBSTEP containing the SET CICSPLEX command must reference the same SEYUAUTH library as the CMAS within the CICSplex. If this library is not referenced, the following message is displayed:

```
DFHRL2004E Unable to connect to CICSPLEX(). SEYUAUTH library mismatch between
DFHDPLOY and CMAS JCL.
```

The DEPLOY BUNDLE command

Define a CICS bundle resource and change its state to DISABLED, ENABLED, or AVAILABLE in a CICS system or group of CICS systems.



Options

BUNDLE ***(data-value)***
Specifies the name of the CICS bundle (up to 8 characters) to deploy.

BUNDLEDIR ***(data-value)***
Specifies the location of the CICS bundle (up to 255 characters) on zFS.

CSDGROUP ***(data-value)***
Specifies the CSD group (8 characters) to add the bundle resource to. The bundle resource is defined in the CSD of each CICS system that is specified by the **SCOPE** option. If **CSDGROUP** is not specified, the bundle is defined in BAS during the deployment and then removed. **CSDGROUP** and **RESGROUP** are mutually exclusive.

Note: In cases where **SCOPE** specifies a group of CICS regions from more than one sysplex, and the CSDs of those CICS regions use the same data set names, the bundle definition is created only on the CSD of the sysplex where the DFHDPLOY utility is running.

DESCRIPTION ***(text)***
An optional value that specifies a description of the bundle definition (up to 58 characters).

RESGROUP ***(data-value)***
Specifies the BAS resource group (8 characters) to add the bundle resource to. The bundle resource is defined in BAS. If **RESGROUP** is not specified, the bundle is defined in BAS during the deployment action and then removed. **RESGROUP** and **CSDGROUP** are mutually exclusive.

SCOPE ***(data-value)***
Specifies the name of the CICS System, or CICS System Group (up to 8 characters) in which to install the CICS bundle.

STATE ***(DISABLED | ENABLED | AVAILABLE)***
Specifies the target state of the bundle. The following options are valid:

- **DISABLED** : Creates the bundle definition, installs the definition, then changes the bundle state to disabled and unavailable.
- **ENABLED** : Creates the bundle definition, installs the definition, then changes the bundle state to enabled and unavailable.
- **AVAILABLE** : Creates the bundle definition, installs the definition, then changes the bundle state to enabled and available.

TIMEOUT (300 | ***data-value)***
An optional numerical value that specifies the maximum amount of time in seconds (1 - 1800 inclusive) for the command to complete. For more information about how to choose an appropriate **TIMEOUT** value, see Troubleshooting the DFHDPLOY utility.

Installing bundles automatically during a COLD start

You can configure your CICS bundles to be installed automatically when a CICS system initializes:

- For a CSD group, add the group to a list using the **DFHCSDUP ADD** command , then specify the list on the **GRPLIST** system initialization parameter .
- For a BAS resource group, associate the **RESGROUP** with a **RESDESC** object. For more information, see Installing resources automatically in BAS.

Example usage

This example connects to *MYPLEX* , removes the existing bundle *WEBSITE* if it exists, and deploys a new bundle *WEBSITE* to an enabled state:

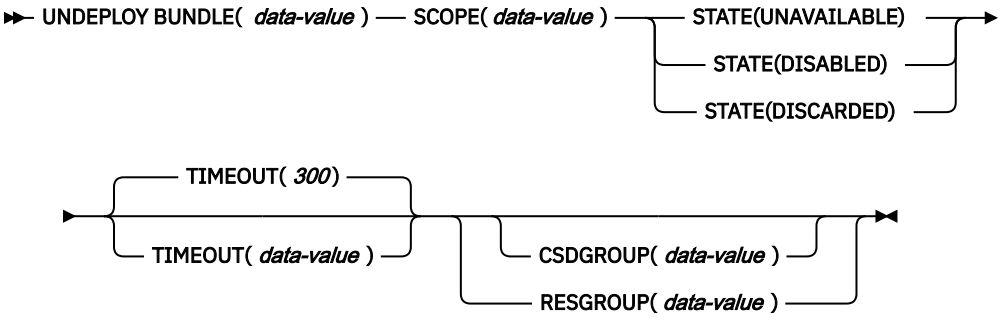
```
//DFHDPLOY JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY
//*
//STEPLIB  DD DISP=SHR,DSN=CICSTS56.CICS.SDFHLOAD
//         DD DISP=SHR,DSN=CICSTS56.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN    DD *
SET CICSPLEX(MYPLEX);
*
UNDEPLOY BUNDLE(WEBSITE)
  CSDGROUP(BANKING) SCOPE(SYS1) STATE(DISCARDED);
*
DEPLOY BUNDLE(WEBSITE)
  BUNDLEDIR(/var/cicsts/bundles/Website_1.0.0/)
  CSDGROUP(BANKING) SCOPE(SYS1) STATE(ENABLED) TIMEOUT(60);
/*
```

The UNDEPLOY BUNDLE command

Change a CICS bundle resource to a specified target state of UNAVAILBLE, DISABLED, or DISCARDED in a CICS system or a group of CICS systems.

Note: If you specify **CSDGROUP** or **RESGROUP** when you deploy a bundle, but do not specify the same option again when you undeploy the bundle, the bundle definition is not removed and remains linked to

the CSD group or BAS resource group. DFHDPLOY does not support the installation of a bundle defined by an existing resource definition.



Options

BUNDLE ***(data-value)***
Specifies the name of the CICS bundle (up to 8 characters) to undeploy.

CSDGROUP ***(data-value)***
Optionally specifies which CSD group to remove the bundle definition from. If it is not specified, the bundle definition is not removed from the CSD. **CSDGROUP** and **RESGROUP** are mutually exclusive.

RESGROUP ***(data-value)***
Optionally specifies which BAS resource group to remove the bundle definition from. If it is not specified, the bundle definition is not removed from the data repository. **RESGROUP** and **CSDGROUP** are mutually exclusive.

SCOPE ***(data-value)***
Specifies the name of the CICS System, or CICS System Group (up to 8 characters) that the CICS bundle is removed from.

STATE ***(UNAVAILABLE | DISABLED | DISCARDED)***
Specifies the target state of the bundle. The following options are valid:

- UNAVAILABLE: Makes the bundle unavailable.
- DISABLED: Makes the bundle unavailable and then disabled.
- DISCARDED: Makes the bundle unavailable, then disabled, and then discards the bundle run time object and removes the definition.

TIMEOUT (300 | *data-value*)
An optional numerical value that specifies the maximum amount of time in seconds (1 - 1800 inclusive) for the command to complete. For more information about how to choose an appropriate **TIMEOUT** value, see Troubleshooting the DFHDPLOY utility.

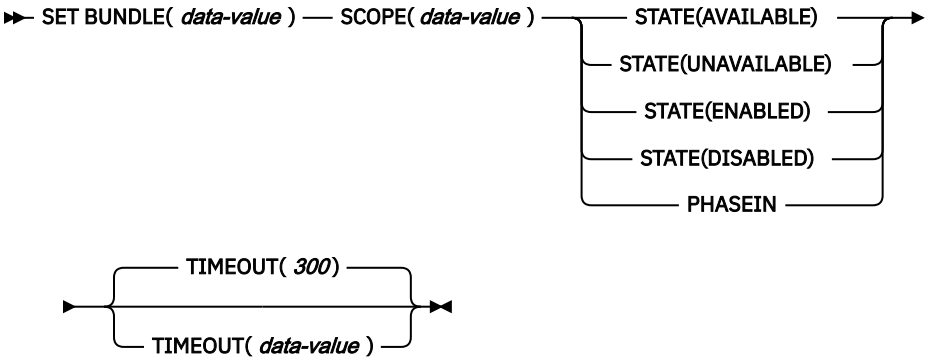
Example

This example connects to *MYPLEX* and undeploys bundle *BUND1* to a disabled state:

```
//DFHDPLY1 JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY
//*
//STEPLIB  DD DISP=SHR,DSN=CICSTS56.CICS.SDFHLOAD
//         DD DISP=SHR,DSN=CICSTS56.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN    DD *
SET CICSPLEX(MYPLEX);
*
UNDEPLOY BUNDLE(BUND1) SCOPE(AOR1)
  STATE(DISABLED) TIMEOUT(10) CSDGROUP(MYCSDGRP);
/*
```

The SET BUNDLE command

Change the state of an installed bundle to the specified target state, in a CICS system or group of CICS systems. For example, the **SET BUNDLE** command can, within a specified SCOPE, change the state of a bundle to DISABLED from any other state. Use this command with the **PHASEIN** option to phase in a higher version of an OSGi bundle without disrupting active tasks.



Options

BUNDLE ***(data-value)***
Specifies the name of the bundle (up to 8 characters) to change state.

PHASEIN
Determines the highest semantic version of all OSGi bundles in the bundle's root directory and registers that version with the OSGi framework, if not already registered. Previously registered versions are removed from the OSGi framework. The new version is used for all subsequent requests, but any active tasks continue to use the old version until the task completes. The bundle must be in an ENABLED state, or the phase-in operation fails.

PHASEIN and **STATE** options are mutually exclusive.

Note: The **PHASEIN** option is valid only for CICS TS V5.3 or later.

SCOPE ***(data-value)***
Specifies the CICS System, or CICS System Group (up to 8 characters) where the bundle is installed.

STATE (*AVAILABLE* | *UNAVAILABLE* | *ENABLED* | *DISABLED)*
Specifies the target state of the bundle. The following options are valid:

AVAILABLE
Enables the bundle, and then makes it available.

UNAVAILABLE
Makes the bundle unavailable.

ENABLED
Enables the bundle.

DISABLED
Makes the bundle unavailable, and then disables it.

STATE and **PHASEIN** options are mutually exclusive.

TIMEOUT (300 | *data-value)*
An optional numerical value that specifies the maximum amount of time in seconds (1 - 1800 inclusive) for the command to complete. For more information about how to choose an appropriate **TIMEOUT** value, see Troubleshooting the DFHDPLOY utility.

For more information about CICS processing of the **SET BUNDLE** command, see SET BUNDLE.

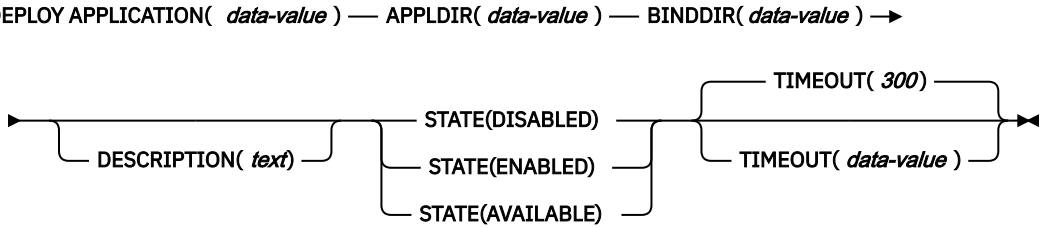
Example

The following example connects to *MYPLEX* and sets the state of *BUND1* to ENABLED:

```
//DFHDPLY1 JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY
//*
//STEPLIB  DD DISP=SHR,DSN=CICSTS56.CICS.SDFHLOAD
//         DD DISP=SHR,DSN=CICSTS56.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN    DD *
SET CICSPLEX(MYPLEX);
*
SET BUNDLE(BUND1) SCOPE(AOR1) STATE(ENABLED) TIMEOUT(10);
/*
```

The DEPLOY APPLICATION command

Define a CICS application and change its state to DISABLED, ENABLED, or AVAILABLE in a platform within the current CICSplex.



Options

APPLICATION *(data-value)*
Specifies the name of the application definition (up to 8 characters) that is used by CICS.

APPLDIR *(data-value)*
Specifies the location of the application bundle (up to 255 characters) on zFS.

BINDDIR *(data-value)*
Specifies the location of the application binding bundle (up to 255 characters) on zFS.

DESCRIPTION *(text)*
An optional value that specifies a description of the application definition (up to 58 characters).

STATE *(DISABLED | ENABLED | AVAILABLE)*
Specifies the target state of the application. The following options are valid:

- DISABLED: Creates the application definition, installs the definition, then changes the application state to disabled and unavailable.
- ENABLED: Creates the application definition, installs the definition, then changes the application state to enabled and unavailable.
- AVAILABLE: Creates the application definition, installs the definition, then changes the application state to enabled and available.

TIMEOUT (300 | *data-value)*
An optional numerical value that specifies the maximum amount of time in seconds (1 - 1800 inclusive) for the command to complete. For more information about how to choose an appropriate **TIMEOUT** value, see Troubleshooting the DFHDPLOY utility.

Example usage

This example connects to *MYPLEX* and deploys application *APP1* to an available state:

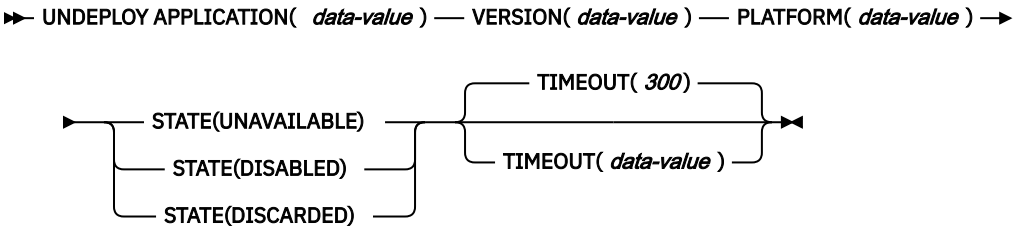
```
//DFHDPLY1 JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
```

```
//DFHDPLOY EXEC PGM=DFHDPLOY
//*
//STEPLIB  DD DISP=SHR,DSN=CICSTS56.CICS.SDFHLOAD
//         DD DISP=SHR,DSN=CICSTS56.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN    DD *
SET CICSPLEX(MYPLEX);
*
DEPLOY APPLICATION(APP1) STATE(AVAILABLE)
  APPLDIR(/var/cicsts/MYPLEX/platform1/applications/application1/)
  BINDDIR(/var/cicsts/MYPLEX/platform1/bindings/binding1/);
/*
```

For more information about the states of an application during deployment, see Checking the status of an application in the CICS Explorer product documentation.

The UNDEPLOY APPLICATION command

Change a CICS application to a specified target state of UNAVAILABLE, DISABLED, or DISCARDED in a platform within the current CICSplex.



Options

APPLICATION *(data-value)*
Specifies the name of the application definition (up to 8 characters) to undeploy.

PLATFORM *(data-value)*
Specifies the platform definition name (up to 8 characters) of the CICS platform to undeploy the application from.

STATE *(UNAVAILABLE | DISABLED | DISCARDED)*
Specifies the target state of the application. The following options are valid:

- UNAVAILABLE: Makes the application unavailable and waits for tasks that are associated with the current application operation to complete.
- DISABLED: Makes the application unavailable, waits for tasks that are associated with the current application operation to complete, and then disables the application.
- DISCARDED: Makes the application unavailable, waits for tasks that are associated with the current application operation to complete, disables the application, and then discards the application object and definition.

Note: A CICS task in one application context can link to an entry point in another CICS application, such that the current application context is stacked. DFHDPLOY does not wait for tasks that have a stacked application context to complete.

TIMEOUT (300 | *data-value)*
An optional numerical value that specifies the maximum amount of time in seconds (1 - 1800 inclusive) for the command to complete. For more information about how to choose an appropriate **TIMEOUT** value, see Troubleshooting the DFHDPLOY utility.

VERSION *(data-value)*
Denotes the cloud style version number of the application that is being undeployed, in the form (x.y.z) where x, y, and z specify a number 0-255.

Example

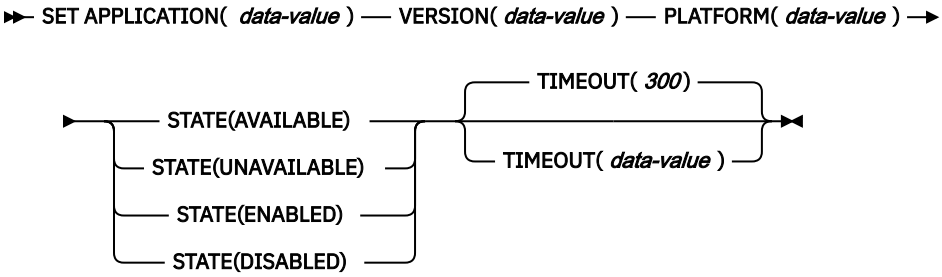
The following example connects to *MYPLEX* and undeploys application *APP1* to a discarded state:

```
//DFHDPLY1 JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY
//*
//STEPLIB  DD DISP=SHR,DSN=CICSTS56.CICS.SDFHLOAD
//         DD DISP=SHR,DSN=CICSTS56.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN    DD *
SET CICSPLEX(MYPLEX);
*
UNDEPLOY APPLICATION(APP1) VERSION(1.0.1)
  PLATFORM(MYPLAT1) STATE(DISCARDED) TIMEOUT(20);
/*
```

For more information about the states of an application during undeployment, see Checking the status of an application in the CICS Explorer product documentation.

The SET APPLICATION command

Change the state of an installed application to the specified target state, in a platform within the current CICSplex. For example, the SET APPLICATION command can change the state of an application from AVAILABLE to UNAVAILABLE.



Options

APPLICATION *(data-value)*
Specifies the application definition name (up to 8 characters) of the CICS application to change state.

PLATFORM *(data-value)*
Specifies the platform definition name (up to 8 characters) of the CICS platform on which the application changes state.

STATE *(AVAILABLE | UNAVAILABLE | ENABLED | DISABLED)*
Specifies the target state of the application. The following options are valid:

- AVAILABLE: Enable the application, and make it available.
- UNAVAILABLE: Make the application unavailable and wait for tasks that are associated with the current application operation to complete.
- ENABLED: Enable the application.
- DISABLED: Make the application unavailable, waits for tasks that are associated with the current application operation to complete, and then disables the application.

Note: A CICS task in one application context can link to an entry point in another CICS application, such that the current application context is stacked. DFHDPLOY does not wait for tasks that have a stacked application context to complete.

TIMEOUT (300 | *data-value)*
An optional numerical value that specifies the maximum amount of time in seconds (1 - 1800 inclusive) for the command to complete. For more information about how to choose an appropriate **TIMEOUT** value, see Troubleshooting the DFHDPLOY utility.

VERSION ***(data-value)***
Denotes the cloud style version number of the application to change state. This is in the form (x.y.z) where x, y and z specify a number 0-255.

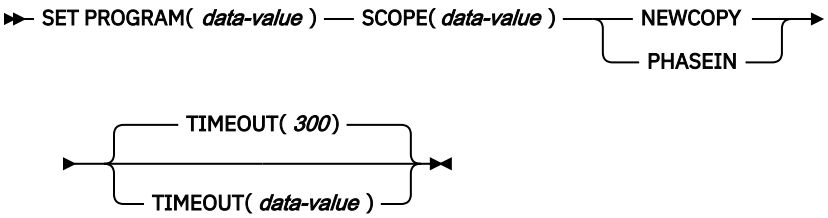
Example

The following example connects to *MYPLEX* and sets the state of *APP1* to DISABLED:

```
//DFHDPLY1 JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY
//*
//STEPLIB  DD DISP=SHR,DSN=CICSTS56.CICS.SDFHLOAD
//         DD DISP=SHR,DSN=CICSTS56.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN    DD *
SET CICSPLEX(MYPLEX);
*
SET APPLICATION(APP1) VERSION(1.0.1)
  PLATFORM(MYPLAT2) STATE(DISABLED) TIMEOUT(20);
/*
```

The SET PROGRAM command

Change the instance of an installed PROGRAM, in a CICS system or group of CICS systems. Use this command with the **PHASEIN** option to phase in a new version of a PROGRAM without disrupting active tasks or **NEWCOPY** option to use a new copy of the PROGRAM when the PROGRAM ceases to be in use by any transaction.



Options

PROGRAM ***(data-value)***
Specifies the name of the PROGRAM (up to 8 characters). Wildcards are supported.

NEWCOPY

CICS is to use a new copy of the program when the program ceases to be in use by any transaction. You can determine whether a module is in use from the RESCOUNT option in an INQUIRE PROGRAM command. A value of zero means the program is not in use. It is possible for CICS to replace the program with the new version during a single transaction, at a point when one use of the program has completed, and a subsequent use has yet to start.

CICS loads the new version either from the DFHRPL or dynamic LIBRARY concatenation, or uses an LPA-resident version, depending on the PRIVATE or SHARED options. PRIVATE is the default setting.

An error will be returned if you specify NEWCOPY for a PROGRAM which has the HOLD option.

An error will be returned if you specify NEWCOPY for a PROGRAM resource that was defined and installed in a CICS bundle.

PHASEIN

CICS uses a new copy of the program now for all new transaction requests. CICS continues to use the old copy for all currently running transactions until they finish (RESCOUNT equal to zero). CICS loads the new version either from the DFHRPL or dynamic LIBRARY concatenation, or uses an LPA-resident version, depending on the PRIVATE or SHARED options. PRIVATE is the default setting.

PHASEIN performs a REFRESH PROGRAM function to inform the loader domain that a new version of the program is cataloged and that this version of the named program must be used in all future ACQUIRE requests.

An error will be returned if you PHASEIN for a program which has the HOLD option.

An error will be returned if you specify PHASEIN for a Java program that runs in a JVM.

An error will be returned if you specify PHASEIN for a PROGRAM resource that was defined and installed in a CICS bundle. Use the SET BUNDLE PHASEIN command instead.

SCOPE ***(data-value)***
Specifies the CICS System, or CICS System Group (up to 8 characters) where the PROGRAM is installed.

TIMEOUT (**300** **|** ***data-value)***
An optional numerical value that specifies the maximum amount of time in seconds (1 - 1800 inclusive) for the command to complete. For more information about how to choose an appropriate **TIMEOUT** value, see Troubleshooting the DFHDPLOY utility.

Note: The **NEWCOPY** and **PHASEIN** options are mutually exclusive.

For more information about CICS processing of the **SET PROGRAM** command, see SET PROGRAM.

Example

The following example connects to *MYPLEX* and phases in a new version of PROGRAM *PROG1* and then performs a NEWCOPY against PROGRAM PROG2:

```
//DFHDPLY1 JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY
//*
//STEPLIB  DD DISP=SHR,DSN=CICSTS56.CICS.SDFHLOAD
//         DD DISP=SHR,DSN=CICSTS56.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN    DD *
SET CICSPLEX(MYPLEX);
*
SET PROGRAM(PROG1) SCOPE(AOR1) PHASEIN TIMEOUT(300);
*
SET PROGRAM(PROG2) SCOPE(AOR2) NEWCOPY TIMEOUT(300);
/*
```

The PERFORM PIPELINE command

Use this command with the **SCAN** option command to initiate a scan of the web service binding directory that is specified in the WSDIR attribute of the PIPELINE.

Options

PIPELINE ***(data-value)***
Specifies the name of the pipeline (up to 8 characters). Wildcards are supported.

SCAN
If the WSDIR attribute is not specified in the PIPELINE, there is nothing to scan. If the directory location specified is valid, CICS examines the Web service binding files in the directory to determine if they should be installed into the system.

SCOPE ***(data-value)***
Specifies the CICS System, or CICS System Group (up to 8 characters) where the pipeline is installed.

TIMEOUT (300 | *data-value)*
An optional numerical value that specifies the maximum amount of time in seconds (1 - 1800 inclusive) for the command to complete. For more information about how to choose an appropriate **TIMEOUT** value, see Troubleshooting the DFHDPLOY utility.

For more information about CICS processing of the **PERFORM PIPELINE** command, see SET PROGRAM.

Example

The following example connects to *MYPLEX* and scans all PIPELINEs in AOR1 and then only PIPELINE *PIPE1* in AOR2:

```
//DFHDPLY1 JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY
//*
//STEPLIB  DD DISP=SHR,DSN=CICSTS56.CICS.SDFHLOAD
//         DD DISP=SHR,DSN=CICSTS56.CPSM.SEYUAUTH
//SYSTSPRT DD SYSOUT=*
//SYSIN    DD *
SET CICSPLEX(MYPLEX);
*
PERFORM PIPELINE(*) SCAN SCOPE(AOR1) TIMEOUT(300);
*
PERFORM PIPELINE(PIPE1) SCAN SCOPE(AOR2) TIMEOUT(300);
/*
```

DFHDPLOY utility script examples

You can use the following examples to create your own JCL to deploy and undeploy your CICS applications and CICS bundles. This topic also includes examples of DFHDPLOY output for successful and failed scripts.

Sample program DFH$DPLY

Undeploying a bundle to a DISCARDED state

Deploying an application to an AVAILABLE state

Using conditional processing in your DFHDPLOY script

Sample program DFH$DPLY

The DFH$DPLY sample program contains annotated DFHDPLOY JCL to deploy, undeploy, and optionally set a sample bundle and application in a CICSplex. The sample is supplied in CICSTS56.CICS.SDFHSAMP.

Undeploying a bundle to a DISCARDED state

The following example script connects to *MYPLEX* and undeploys bundle *MYBUND* to a DISCARDED state in the CICS regions within *MYSCOPE* :

```
//DFHDPLY1
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHPLOY EXEC PGM=DFHDPLOY,REGION=100M
//*
//STEPLIB DD DISP=SHR,DSN=CICSTS56.CICS.SDFHLOAD
//        DD DISP=SHR,DSN=CICSTS56.CPSM.SEYUAUTH
*
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
*
SET CICSPLEX(MYPLEX);
*
UNDEPLOY BUNDLE(MYBUND) SCOPE(MYSCOPE)
TIMEOUT(40) STATE(DISCARDED);
/*
```

The following is the DFHDPLOY output for this script when it runs successfully:

```
DFHRL2132I Analyzing CICS regions and CSD attributes.
DFHRL2093I BUNDLE(MYBUND) found in SCOPE(MYSCOPE).
DFHRL2129I BUNDLE(MYBUND) state is INSTALLED on 2 CICS regions in
SCOPE(MYSCOPE).
DFHRL2129I BUNDLE(MYBUND) state is ENABLED on 2 CICS regions in
SCOPE(MYSCOPE).
DFHRL2054I Setting BUNDLE state to DISABLED. DFHRL2042I Discarding
BUNDLE(MYBUND).
DFHRL2077I BUNDLE(MYBUND) has been discarded from SCOPE(MYSCOPE).
DFHRL2037I UNDEPLOY command successful.
```

The following is an example of potential DFHDPLOY output for this script when it fails due to timing out:

```
DFHRL2132I Analyzing CICS regions and CSD attributes.
DFHRL2093I BUNDLE(MYBUND) found in SCOPE(MYSCOPE).
DFHRL2129I BUNDLE(MYBUND) state is INSTALLED on 2 CICS regions in
SCOPE(MYSCOPE).
DFHRL2129I BUNDLE(MYBUND) state is ENABLED on 2 CICS regions in
SCOPE(MYSCOPE).
DFHRL2054I Setting BUNDLE state to DISABLED.
DFHRL2039W The command failed to complete within the specified time out period
of 40 seconds.
DFHRL2129I BUNDLE(MYBUND) state is INSTALLED on 2 CICS regions in
SCOPE(MYSCOPE).
DFHRL2129I BUNDLE(MYBUND) state is ENABLED on 1 CICS regions in
SCOPE(MYSCOPE).
DFHRL2129I BUNDLE(MYBUND) state is DISABLED on 1 CICS regions in
SCOPE(MYSCOPE).
DFHRL2041I TIMEOUT has occurred before BUNDLE(MYBUND) has reached
STATE(DISABLED)

Status of BUNDLE(MYBUND) in SCOPE(MYSCOPE):
CICS ENABLESTATUS AVAILSTATUS
CICSSYS1 ENABLED NONE MYBUND
CICSSYS2 DISABLED NONE MYBUND

All Bundle part records for BUNDLE(MYBUND):
CICS BUNDLE ENABLESTATUS AVAILSTATUS BUNDLEPART
CICSSYS1 MYBUND DISABLED NONE SAMP
http://www.ibm.com/xmlns/prod/cics/bundle/TRANSACTION
CICSSYS1 MYBUND ENABLED NONE SAMPFILE
http://www.ibm.com/xmlns/prod/cics/bundle/FILE
CICSSYS2 MYBUND DISABLED NONE SAMP
http://www.ibm.com/xmlns/prod/cics/bundle/TRANSACTION
CICSSYS2 MYBUND DISABLED NONE SAMPFILE
http://www.ibm.com/xmlns/prod/cics/bundle/FILE

DFHRL2055I Errors have occurred, processing terminated.
```

Deploying an application to an AVAILABLE state

The following example script connects to *MYPLEX* and deploys application *MYAPP* to an AVAILABLE state:

```
//DFHDPLY1
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHPLOY EXEC PGM=DFHDPLOY,REGION=100M
//*
//STEPLIB DD DISP=SHR,DSN=CICSTS56.CICS.SDFHLOAD
//        DD DISP=SHR,DSN=CICSTS56.CPSM.SEYUAUTH
//SYSIN DD *
*
SET CICSPLEX(MYPLEX);
*
DEPLOY APPLICATION(MYAPP)
APPLDIR(/var/cicsts/MYPLEX/myplatform/myapplications/myapplication)
BINDDIR(/var/cicsts/MYPLEX/myplatform/mybindings/mybinding)
TIMEOUT(400) STATE(AVAILABLE);
/*
```

The following is the DFHDPLOY output for this script when it runs successfully:

```
DFHRL2044I CICS application definition(MYAPP) created.
DFHRL2045I CICS application definition(MYAPP) installed.
DFHRL2046I Setting application state to ENABLED.
DFHRL2046I Setting application state to AVAILABLE.
DFHRL2012I DEPLOY command completed successfully.
```

```
DFHRL2007I Processing complete.
DFHRL2014I Disconnecting from CICSPLEX(MYPLEX).
```

The following is an example of potential DFHDPLOY output for this script when it fails:

```
DFHRL2044I CICS application definition MYAPP created.
DFHRL2045I CICS application definition MYAPP installed.
DFHRL2046I Setting application state to ENABLED.
DFHRL2064I The state of application MYAPP version 0.0.0 is SOMEDISABLED and
availability is UNAVAILABLE.

Application Management part records which did not reach desired state:
ENABLESTATUS AVAILSTATUS BUNDLE(version) REGIONTYPE
SOMEDISABLED NONE BUNDMIX2(0.0.0) RegionType1
SOMEDISABLED NONE BUNDMIX2(0.0.0) RegionType2

DFHRL2064I The state of application MYAPP version 0.0.0 is SOMEDISABLED and
availability is UNAVAILABLE.

Showing all Application Management part records. Record count = 4
ENABLESTATUS AVAILSTATUS BUNDLE(version) REGIONTYPE
SOMEDISABLED NONE BUNDMIX2(0.0.0) RegionType1
SOMEDISABLED NONE BUNDMIX2(0.0.0) RegionType2
ENABLED UNAVAILABLE ProgramEP(1.0.0) RegionType1
ENABLED UNAVAILABLE ProgramEP(1.0.0) RegionType2

DFHRL2055I Errors have occurred. Processing terminated.
DFHRL2014I Disconnecting from CICSplex MYPLEX.
```

Using conditional processing in your DFHDPLOY script

You can use the return code from your DFHDPLOY job to conditionally process later steps by using the JCL COND parameter and the IF/THEN, ELSE, and ENDIF statements. See COND Parameter in the z/OS MVS JCL Reference and IF/THEN/ELSE/ENDIF statement construct in z/OS MVS JCL Reference.

The following script uses the SET APPLICATION command to reattempt the transition of application *MYAPP* to an AVAILABLE state, after the first attempt to deploy the application failed with a return code of 4:

```
//DPLYAPPL
JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY EXEC PGM=DFHDPLOY,REGION=100M
//*
//STEPLIB DD DISP=SHR,DSN=CICSTS56.CICS.SDFHLOAD
//        DD DISP=SHR,DSN=CICSTS56.CPSM.SEYUAUTH
*
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
*
SET CICSPLEX(MYPLEX);
*
DEPLOY APPLICATION(MYAPP)
APPLDIR(/var/cicsts/MYPLEX/myplatform/myapplications/myapplication)
BINDDIR(/var/cicsts/MYPLEX/myplatform/mybindings/mybinding)
TIMEOUT(10)
STATE(AVAILABLE);
*
//DPLYFAIL IF (DFHDPLOY.RC = 4) THEN
//DFHDPLY2 EXEC PGM=DFHDPLOY,REGION=100M
//*
//STEPLIB DD DISP=SHR,DSN=CICSTS56.CICS.SDFHLOAD
//        DD DISP=SHR,DSN=CICSTS56.CPSM.SEYUAUTH
*
//SYSTSPRT DD SYSOUT=*
//SYSIN DD *
* If we TIMED out with an RC=4 then retry making MYAPP
* AVAILABLE.
*
SET CICSPLEX(MYPLEX);
*
SET APPLICATION(MYAPP)
VERSION(4.4.4)
PLATFORM(MYPLATFM)
STATE(AVAILABLE);
```

```
//*
// ENDIF
```

DFHDPLOY utility return codes

If an error occurs while you run the DFHDPLOY utility, messages are written to the SYSTSPRT data definition destination and the step completes with a nonzero return code. When the return code is 8 or higher, no further commands in the input stream are processed.

There might be related messages in the CICSPlex SM address space (CMAS), connected to as a result of the SET CICSPLEX command, and in CICS systems in which the application or bundle is being deployed or undeployed.

DFHDPLOY return code	Description
0	All commands in the input stream completed successfully.
4	Either one or more warnings occurred during processing, or some commands did not complete successfully within the time that is specified by **TIMEOUT**. The following examples complete with a return code of 4: • An attempt to transition a bundle without any entry points to an ENABLED and AVAILABLE state. • An application, which has a target state of DISABLED, ends in an INCOMPLETE state. In cases where the script times out during the transition of a resource to its final target state, this return code does not reflect the success of the transition. Further processing of the application or bundle might occur. No further commands in the input stream are processed.
8	Error. The last command that was processed did not complete successfully, or processing timed out before the transition to the final target state began. The final target state was not reached. No further commands in the input stream are processed.
12	Error. The last command that was processed could not be validated. No further commands in the input stream are processed.

You can use the return code from your DFHDPLOY job to conditionally process later steps by using the JCL COND parameter and the IF/THEN, ELSE, and ENDIF statements. See COND Parameter in the z/OS MVS JCL Reference and IF/THEN/ELSE/ENDIF statement construct in z/OS MVS JCL Reference. For an example, see “Using conditional processing in your DFHDPLOY script” on page 749.

Receiving output messages in Japanese or Simplified Chinese

To receive DFHDPLOY output messages in Japanese or Simplified Chinese, use the DFHDPLOY **EXEC** command. Ensure that you set the appropriate client code page in your client in order to view the messages.

Procedure

1. Set the appropriate **PARM** parameter on the DFHDPLOY **EXEC** command.

   For output messages in Japanese, set PARM='K'.

   For output messages in Simplified Chinese, set PARM='S'.

Here is an example of the DFHDPLOY **EXEC** command configured to receive output messages in Japanese:

```
//DFHDPLOY EXEC PGM=DFHDPLOY,PARM='K'
```

2. Set the appropriate client code page in your client in order to view the messages.

   The following table lists the appropriate client code page for each language:

Language	PARM keyword	Client code page	Host code page
Japanese	K	1390/1399	939
Simplified Chinese	S	GB18030/GB2312	935

Deployment with UrbanCode Deploy

You can use IBM UrbanCode Deploy and the CICS TS plug-in to automate the deployment and undeployment of CICS resources. Used in conjunction with other CICS tooling, UrbanCode Deploy can improve workflow efficiency and contribute to a continuous delivery environment.

Requirements

UrbanCode Deploy is supported for CICS TS for z/OS Version 4 and above, and requires a configured CICS management client interface (CMCI) port.

Capabilities

The plug-in includes steps that can automate the following actions:

- Install CSD resources, groups, and lists
- Install BAS resources, resource descriptions, and groups
- Discard resources
- Enable and disable resources
- Open and close resources
- Newcopy and phase in resources
- Scan pipelines
- Check the enabled or open status of resources

The plug-in also provides component templates, which allow you to reuse component processes and properties across similar deployment scenarios.

For more detailed information, including full usage instructions, see the CICS TS plug-in for UrbanCode Deploy website.

For more information about IBM UrbanCode Deploy, including typical use cases, features, and to download an evaluation, see the UrbanCode Deploy website.

Deployment with Zowe CLI CICS deploy plug-in

You can use the Zowe™ CLI CICS deploy plug-in for Zowe CLI to deploy applications and resources developed on a workstation as CICS bundles to IBM CICS Transaction Server for z/OS (CICS). It also provides low-level commands for performing individual steps of the deployment process that can be used as part of a Continuous Integration / Continuous Delivery (CI/CD) pipeline.

Requirements

The Zowe CLI CICS deploy plug-in requires the following servers and facilities be set up on z/OS:

- z/OS Management Facility

- z/OS Secure SHell daemon
- DFHDPLOY
- CICSPlex System Manager (CPSM)

For more information, see Requirements on z/OS.

Capabilities

The plug-in allows you to:

- Generate a CICS bundle from an existing workstation directory structure.
- Deploy a CICS bundle to a CPSM managed group of CICS regions.
- Undeploy a CICS bundle from a CPSM managed group of CICS regions.

The plug-in provides a command line interface and is easily scripted. The plug-in is especially well suited to deploying Node.js applications to CICS. It builds upon the capabilities of DFHDPLOY allowing deployment of CICS bundles from a workstation directly to CICS using simple command line interactions.

For more detailed information, including full usage instructions, see the Zowe CLI CICS deploy plug-in documentation.

Notices

This information was developed for products and services offered in the United States of America. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property rights may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119 Armonk,
NY 10504-1785
United States of America

© Copyright IBM Corp. 1974, 2023

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Client Relationship Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Programming interface information

IBM CICS supplies some documentation that can be considered to be Programming Interfaces, and some documentation that cannot be considered to be a Programming Interface.

Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 5 Release 6 (CICS TS 5.6) are included in the following sections of the online product documentation:

- Developing applications
- Developing system programs
- CICS TS security
- Developing for external interfaces
- Application development reference
- Reference: system programming
- Reference: connectivity

Information that is NOT intended to be used as a Programming Interface of CICS TS 5.6, but that might be misconstrued as Programming Interfaces, is included in the following sections of the online product documentation:

- Troubleshooting and support
- CICS TS diagnostics reference

If you access the CICS documentation in manuals in PDF format, Programming Interfaces that allow the customer to write programs to obtain the services of CICS TS 5.6 are included in the following manuals:

- Application Programming Guide and Application Programming Reference
- Business Transaction Services

- Customization Guide
- C++ OO Class Libraries
- Debugging Tools Interfaces Reference
- Distributed Transaction Programming Guide
- External Interfaces Guide
- Front End Programming Interface Guide
- IMS Database Control Guide
- Installation Guide
- Security Guide
- CICS Transactions
- CICSPlex System Manager (CICSPlex SM) Managing Workloads
- CICSPlex SM Managing Resource Usage
- CICSPlex SM Application Programming Guide and Application Programming Reference
- Java Applications in CICS

If you access the CICS documentation in manuals in PDF format, information that is NOT intended to be used as a Programming Interface of CICS TS 5.6, but that might be misconstrued as Programming Interfaces, is included in the following manuals:

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSPlex SM Problem Determination Guide

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Apache, Apache Axis2, Apache Maven, Apache Ivy, the Apache Software Foundation (ASF) logo, and the ASF feather logo are trademarks of Apache Software Foundation.

Gradle and the Gradlephant logo are registered trademark of Gradle, Inc. and its subsidiaries in the United States and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Red Hat®, and Hibernate® are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Spring Boot is a trademark of Pivotal Software, Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Zowe, the Zowe logo and the Open Mainframe Project™ are trademarks of The Linux Foundation.

The Stack Exchange name and logos are trademarks of Stack Exchange Inc.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability
These terms and conditions are in addition to any terms of use for the IBM website.

Personal use
You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use
You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights
Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM online privacy statement

IBM Software products, including software as a service solutions, (*Software Offerings*) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information (PII) is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect PII. If this Software Offering uses cookies to collect PII, specific information about this offering's use of cookies is set forth below:

For the CICSPlex SM Web User Interface (main interface):
Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name and other PII for purposes of session management, authentication, enhanced user usability, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface (data interface):
Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's user name and other PII for purposes of session management, authentication, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface ("hello world" page):
Depending upon the configurations deployed, this Software Offering may use session cookies that do not collect PII. These cookies cannot be disabled.

For CICS Explorer:

Depending upon the configurations deployed, this Software Offering may use session and persistent preferences that collect each user’s user name and password, for purposes of session management, authentication, and single sign-on configuration. These preferences cannot be disabled, although storing a user's password on disk in encrypted form can only be enabled by the user's explicit action to check a check box during sign-on.

If the configurations deployed for this Software Offering provide you, as customer, the ability to collect PII from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM Privacy Policy and IBM Online Privacy Statement, the section entitled *Cookies, Web Beacons and Other Technologies* and the IBM Software Products and Software-as-a-Service Privacy Statement.

Index

Special Characters

#echo command 318
#include command 318
#set command 308, 318

Numerics

10/63 magnetic slot reader 435
31-bit addressing
    assembler language 449
    COBOL 513
31-bit mode transaction 152
3262 printer 346
3270 bridge
    ADS descriptor 377
3270 display 260
3270 family
    AID 284
    attention identifier 284
    attention keys 284
    attributes, extended 278
    base color 277
    buffer 275
    color, base 277
    data stream 274
    data stream orders 279
    data stream, outbound 282
    display characteristics 276
    extended attributes 278
    field attributes 277
    field format, inbound 286
    fields 276
    inbound field format 286
    input from 284
    intensity 277
    MDT 277
    modified data tag 277
    orders in data stream 279
    outbound data stream 282
    protection 277
    reading from 285
    terminal, writing to 275
    unformatted mode 287
    write control character 275
    writing to terminal 275
3270 Information Display System 709
3270 printer
    options 348
3270 screen field 400
3289 printer 346
3290 display 427
3601 logical unit 433
3770 batch data interchange logical unit 433
3770 batch logical unit 433
3790 batch data interchange logical unit 433

A

abend
    in Parameter level 506
abend codes 501
ABEND command 204
abend exit facility 204
abend exit program 205
abend exit routine 205
abend handling 564
abend user task, EDF 644
abendTask
    in CICS conditions 504
abnormal termination recovery 204
ACCEPT statement, COBOL 513
access to system information
    EXEC interface block (EIB) 12
Accessing start data
    in Starting transactions asynchronously 490
    in Using CICS Services 490
ACCUM option 348, 406
ACK 259
acknowledgment 259
active partition 431
ACTPARTN option 406, 431
adding records 252
ADDRESS command 11
ADDRESS COMMAREA command 149
ADDRESS special register 518
addressing mode (AMODE)
    options for CICS applications 684
addressing of CICS areas 439
ADS descriptor 377
affinity
    recommended programming techniques 158
    safe programming techniques 158
    suspect programming techniques 158
    unsafe programming techniques 158
AFFINITY attribute 179
affinity groups 179
affinity relations
    BAPPL 180
    LOCKED 182
    LUname 184
    userid 185
AFFLIFE attribute 179
AFTER option 288
agile service delivery 31
AIX, CICS for
    in Platform differences 505
ALARM option 389
ALLOCATE command
    inhibit wait, NOSUSPEND option 101
ALLOCERR condition 361
alternate
    index 224
    key 224

ALTPAGE value 409
AMODE (addressing mode)
  options for CICS applications 684
APAK transaction 351
APCG 97
API
  subset for DPL 218
APOST option 582
APPLCTN 673
APPLDEF 673
application
  management part 58
  status 58
application binding 49
application bundle 625, 673
application context 53
application context propagation 53
application design
  BIND options 724
  storm drain effect 202
application entry point 50, 53
application processing unit
  designing 556
application program logical levels 520
application programs
  asynchronous processing 222
  design 73
  design for performance 96
  distributed program link 211
  distributed transaction processing 222
  function shipping 210
  installing 674
  intercommunication considerations 209
  logical levels 147
  testing 718
  transaction routing 210
  using BMS map sets 689
  writing 8
application project 673
application/x-www-form-urlencoded 308
applications
  deploying 673
  division into units of work 557
  enabling 673
  installing 673
area, dynamic storage 97
argc 468
argv 468
ASIS option 400
ASKTIME command 287
assembler language
  31-bit addressing 449
  64–bit registers 449
  applications 98
  CALL statement 458
  Language Environment requirements 453
  mixing with other languages 458, 538
  non-conforming routines 453
  programming techniques 449, 453, 458
  restrictions 449
  working storage 449
Assembler language
  DFHECALL macro 595
assembly 577
assembly language
  programming techniques 458
  working storage 449
assembly, TYPE=DSECT 376
assign
  in Example of file control 488
ASSIGN command
  DESTCOUNT option 424
  MAPCOLUMN option 413
  MAPHEIGHT option 413
  MAPLINE option 413
  MAPWIDTH option 413
  MSR option 435
  options 265
  PAGENUM option 424
async api 17–20, 22
Asynchronous API 443
asynchronous journal output 187
asynchronous processing 209, 222
asynchronous requests 103
AT option 288
ATI 73, 258, 305
ATNI 210
attention field 437
attention identifier 401
ATTENTION key 259
Automatic condition handling (callHandleEvent)
  in CICS conditions 504
  in Conditions, errors, and exceptions 504
automatic creation 475
automatic deletion 475
automatic task initiation 73, 258
automatic transaction initiation 305
automatic transaction initiation (ATI) 561
AUTOPAGE option 408
auxiliary temporary storage 91
auxiliary trace 101
AZI6 210

B

backout of resources 188
BAKR (branch and stack) assembler instruction 449
base class
  overview 476
Base classes
  in Overview of the foundation classes 476
BASE option 386
basic mapping support
  assembling and link-editing physical map sets 709
  assembling map 376
  assembly, TYPE=DSECT 376
  basic mapping support
    standard 367
  BMS support levels 367
  complex fields 378
  composite fields 378
  copy facility 440
  creating map 371
  cursor position 393
  cursor, finding the 402
  data streams 438
  data, moving to map 387
  DFHASMVS procedure 709

basic mapping support *(continued)*
DFHLNKVS procedure 709
DFHMAPS, procedure for installing maps 716
DFHMDF macro 371
DFHMDI macro 371, 373
DFHMSD macro 371, 373
DFHPSD, for defining partition sets 717
display, receiving data from 396
EOC condition 405
field, group 378
fields 370
fields, complex 378
fields, composite 378
fields, repeated 379
finding the cursor 402
full 367
group field 378
GRPNAME option 378
initializing output map 386
installing mapsets 706, 708
installing partition sets 717
installing physical map sets 709
installing symbolic description map sets 711
invalid data 394
link-edit 376
macro 371
macros, rules for writing 374
map 369
map sets 377
map, assembling 376
map, creating 371
map, initializing output 386
map, moving data to 387
map, physical 376
map, symbolic 376
mapping input data 398
maps 438–440
maps, storage 385
MAPSET resource definition 376
MDT 400
message lengths, reducing 414
minimizing path length 413
minimum 367
modified data tag (MDT) 438
moving data to map 387
multimap screens 440
OCCURS option 379
output example 368
output map, initializing 386
page building operations 440
page routing operations 440
path length, minimizing 413
performance considerations 413
physical map 376
physical maps 708
preparing maps 706
PROGRAM resource definition 376
receiving data from display 396
reducing message lengths 414
repeated fields 379
rules for writing macros 374
screen copy 357
SEND MAP command 385
storage for maps 385
basic mapping support *(continued)*
support across platforms 367
symbolic description map sets for BMS 711
symbolic map 376
symbolic maps 708
terminals supported 367
TYPE=DSECT assembly 376
types of mapsets 708
uppercase translation 400
using BMS map sets in application programs 689
using symbolic map sets in a program 711
basic mapping support (BMS)
terminal paging 562
batch compilation for COBOL programs 527
batch data interchange
definite response 273
DEFRESP option 273
destination identification 273
ISSUE WAIT command 273
NOWAIT option 273
BDAM
browsing operations 248
data sets 234
exclusive control 251
updating operations 251
BDI 346
beginInsert
in Writing records 485
BGAM 257
big COMMAREAs 109, 112, 118, 134
bind options
bind options
cursor stability 724
cursor stability 724
isolation level 724
repeatable read 724
validation 725
BIND options
in application design 724
in program preparation 724
RETAIN 724
bind process
following a program change 723
options and considerations 724
BIND time stamps 724
blank fields 438
block references 234
blocked-data set 234
BMS
routing 355
BMS commands 256
bookmarks 325, 327
BOTTOM command, CEBR transaction 654
BRACKET option 270
bracket protocol, LAST option 270
bridge (3270)
ADS descriptor 377
brightness 436
browse operation
BDAM 248
BROWSE TEMP STORAGE option, CEDF 645
browsing
DELAY 256
records 245

browsing *(continued)*
SUSPEND 256
browsing records 487
Browsing records
in File control 487
in Using CICS Services 487
BTS activities 134
buffer
in Example of starting transactions 491, 492
buffer (parameter)
in Polymorphic Behavior 509
Buffer objects
Data area extensibility 482
Data area ownership 482
IccBuf constructors 482
IccBuf methods 483
Working with IccResource subclasses 483
buffers 481, 484
build 735
BUILDCHAIN 268

C

C and C++
arguments 471
EIB, accessing 472
locale support 473
mixing with other languages 538
programming techniques 468
restrictions 468
support 467
working storage 467
XPLink 473, 474
C language considerations
LENGTH option default 93
struct, symbolic description map set 708
C++ exceptions 501
C++ Exceptions and the Foundation Classes
in Conditions, errors, and exceptions 501
caching of document templates 311
CALL DL/I interface, COBOL 516
CALL statement
assembler language 458
callable services 534
callHandleEvent
in CICS conditions 504
calling conventions 511
Calling methods on a resource object
in Overview of the foundation classes 481
in Using CICS resources 481
cancel
in Cancelling unexpired start requests 490
CANCEL command 287
Cancelling unexpired start requests
in Starting transactions asynchronously 490
in Using CICS Services 490
catch
in C++ Exceptions and the Foundation Classes 501, 502
in Exception handling (throwException) 505
CBLCARD option 582
CBLPSHPOP runtime option for Language Environment 516
CDUMP 468
CEBR transaction
BOTTOM command 654
CEBR transaction *(continued)*
browse transaction 650
CEBR initiation 650
COLUMN command 654
displays 652
FIND command 654
GET command 655
initiation 650
LINE command 655
PURGE command 655
PUT command 655
QUEUE command 655
security considerations 650
SYSID command 655
temporary storage browse 650
TERMINAL command 655
TOP command 655
transient data 656
CECI transaction
about to execute command 659
ampersand (&) 662
body 660
command execution complete 659
command input 657
command input line 657
command line 657
command syntax check 658
ENQ commands 665
function key values area 661
information area 660
introduction 656
making changes 662
message line 660
program control 664
screen layout 657
shared storage 665
status area 658
terminal sharing 665
variables 662
CECS transaction 661
CEDF transaction
abend user task 644
browse temporary storage 645
display register 645
displays 632
DPL 642
dual-screen mode 640
EDF transaction 630
function key 632
functions 629
invoke CECI 645
invoking 630
modifying execution 643
non-terminal transactions 641
options on function (PF) keys 644
overtyping displays 643
program labels 643
pseudoconversational programs 640
remote transactions 641
remote-linked programs 642
security 630
single-screen mode 639
CEEBINT, Language Environment HLL user exit 545
CEEBXITA user exit 543

CEECSTX user exit 543
CEEDOPT CSECT 541
CEEENTRY macro 453
CEEHDLR service 535
CEEROPT CSECT 541
CEEUOPT CSECT 541
CEEWUCHA sample user condition handler 517, 548
CESF, GOODNIGHT transaction 272
chaining 260
chaining of data 268
channel 9
CHANNEL
  LINK command 146
  option 108, 146
  RETURN command 146
  XCTL command 146
channels
  as large COMMAREAs 108
  basic examples 110
  benefits of 118
  CICS translator 124
  compared to BTS activities 134
  constructing 132
  creating 119
  current 120, 124
  current, example, with LINK commands 120
  current, example, with XCTL commands 122
  designing 131
  discovering which containers were passed to a program 129
  discovering which containers were returned from a link 129
  dynamic and distributed routing 136
  lifetime 129
  LINK command 152
  migrating
    dynamically-routed applications 146
    LINK command 143
    RETURN command 144
    temporary storage 145
    XCTL command 144
  on LINK and XCTL commands 90
  on RETURN commands 108
  overview 108, 109
  read only containers 133
  RETURN command 154
  scope of 127
  typical scenarios
    multiple interactive components 116
    one channel—one program 112
    one channel—several programs 114
    several channels, one component 116
  using from JCICS 134
channels as large COMMAREAs 108, 109, 112, 118, 134
char*
  in C++ Exceptions and the Foundation Classes 502, 503
character set 312
checkout, program 629
child enclaves 541
CICS
  in Platform differences 505
  testing environment 13
CICS areas, addressing 439
cics bt 735
CICS build toolkit 735
CICS bundle 31
CICS bundles 625
CICS conditions
  abendTask 505
  automatic condition handling 504
  Automatic condition handling (callHandleEvent) 504
  callHandleEvent 504
  exception handling 505
  Exception handling (throwException) 505
  in Conditions, errors, and exceptions 503
  manual condition handling 504
  Manual condition handling (noAction) 504
  noAction 504
  severe error handling 505
  Severe error handling (abendTask) 505
  throwException 505
CICS Db2 environment
  preparation 721
  testing 720
CICS file
  as document template 314
CICS for AIX
  in Platform differences 505
CICS option 582
CICS printer
  determining characteristics of 351
CICS program
  as document template
    DFHDHTL macro 314, 315
CICS resources 480
CICS-key storage 293
CICS-maintained table 236
CICS-value data areas 93
cicsbt 735–737
cicsbt resolve 732
CICSCondition
  in C++ Exceptions and the Foundation Classes 503
CICSDATAKEY option 89, 296
CICSVAR environment variable 544
class
  base 476
  resource 478
  resource identification 477
  singleton 481
  support 479
CLASS option 363
clear
  in Example of polymorphic behavior 510
  in Polymorphic Behavior 509
CLEAR
  key 432
  PARTITION AID value 432
  PARTITION key 432
CLEAR key 108
client code page 312
client region 211
CLOCK 468
cmmCICS
  in Storage management 511
cmmDefault
  in Storage management 511
cmmNonCICS
  in Storage management 511

CMT 236
CNOTCOMPL option 268
COBOL
  31-bit addressing 513
  ADDRESS special register 518
  addressing CICS data areas 518
  batch compilation 527
  blank lines 525
  CALL DL/I interface 516
  calling subprograms 519, 522
  CBLPSHPOP runtime option 516
  compiler options 513
  example of DFHNCTR call 343
  global variables 525
  mixing with other languages 538
  nested programs 528
  programming restrictions
    VS COBOL II 517
  programming techniques 513, 518, 522
  reference modification 525
  REPLACE statement 525
  reserved word table 513
  restrictions 98, 513, 522, 525
  run unit 520
  support 513
  translation 525
  WITH DEBUGGING MODE 513
  working storage 513
COBOL2 option 582
COBOL3 option 582
code pages for CICS documents 312
CODEREG operand 455
COLUMN
  command, CEBR transaction 654
COM assembler instruction 449
command language translator
  APOST option 582
  CBLCARD option 582
  CICS option 582
  COBOL2 option 582
  COBOL3 option 582
  CPSM option 582
  DBCS option 582
  DEBUG option 582
  DLI option 582
  EDF option 582
  EPILOG option 582
  EXCI option 582
  FLAG option 582
  GDS option 582
  GRAPHIC option 582
  LEASM option 582
  LENGTH option 582
  line numbers 581
  LINECOUNT option 582
  LINKAGE option 582
  MAIN option 582
  MARGINS option 582
  NATLANG option 582
  NOCBLCARD option 582
  NOCPSM option 582
  NODEBUG option 582
  NOEDF option 582
  NOEPILOG option 582
command language translator *(continued)*
  NOFEPI option 582
  NOLENGTH option 582
  NOLINKAGE option 582
  NOMAIN option 582
  NONUM option 582
  NOOPSEQUENCE option 582
  NOOPTIONS option 582
  NOPROLOG option 582
  NOSEQ option 582
  NOSEQUENCE option 582
  NOSOURCE option 581, 582
  NOSPIE option 582
  NOVBREF option 582
  NOXREF option 582
  NUM option 582
  OPMARGINS option 582
  OPSEQUENCE option 582
  options 590, 592
  OPTIONS option 582
  PROLOG option 582
  QUOTE option 582
  SEQ option 582
  SEQUENCE option 582
  SOURCE option 581, 582
  SP option 582
  SPACE option 582
  SPIE option 582
  SYSEIB option 582
  VBREF option 581, 582
  XOPTS keyword 590
  XREF option 582
command line 735, 750
command-level interpreter
  ENTER key 661
  invoking 661
  security considerations 656
command-line 735, 750
command, SYNCPOINT 188
COMMAREA
  LINK command 146
  migrating to channels and containers
    dynamically-routed applications 146
    LINK command 143
    RETURN command 144
    XCTL command 144
  option 107, 146, 149, 150
COMMAREA (communication area) 9, 559
COMMAREAs > 32K 108, 118, 134
common work area (CWA)
  protecting 104
communication area (COMMAREA) 9
communication area for a document template 316
communication between transactions
  use of resources 559
communication with terminals
  external design considerations 553
Communications Server 258
compilation 577
compilers supported
  Language Environment 533
compiling programs 499
Compiling Programs
  in Compiling, executing, and debugging 499

complete key 485
complex fields 378
components
  multiple, interactive 116
  one channel—several programs 114
  several channels, one component 116
composite fields 378
condition
  in Manual condition handling (noAction) 504
  in Resource classes 479
condition handling 563
condition, exception 189
Conditions, errors, and exceptions
  Automatic condition handling (callHandleEvent) 504
  Exception handling (throwException) 505
  Manual condition handling (noAction) 504
  Method level 506
  Object level 506
  Parameter level 506
  Severe error handling (abendTask) 505
configuring
  agile service delivery 623
  application 623
CONNECT command sample 747
CONNECT PROCESS command 220
CONSISTENT option
  READ command 243
constructing a channel 132
container 9
containers
  basic examples 110
  context, BTS or channel 136
  designing a channel 131
  discovering which containers were passed to a program 129
  discovering which were returned from a link 129
  lifetime 129
  migrating
    dynamically-routed applications 146
    LINK command 143
    RETURN command 144
    temporary storage 145
    XCTL command 144
  overview 108, 109
  read only 133
  using from JCICS 134
contention for terminal 258
context
  of containers, BTS or channel 136
control
  exclusive of BDAM 251
  of VSAM blocks 230
conversation partner 258
conversational processing 558
conversational programming 274
CONVERSE command 260, 270, 274
copy facility
  BMS 440
COPY statements 594
copybook translation 594
counter name
  named counters 331
coupling facility data tables 237
coupling facility list structure
coupling facility list structure *(continued)*
  current value 331
CPI
  references 8
CPI Communications interface module, DFHCPLC 594
CPI Communications stub 223
CPI-C 209, 223
CPSM option 582
CQRY transaction 74
creating a channel 119
creating a resource object 480
Creating a resource object
  in Overview of the foundation classes 480
  in Using CICS resources 480
  Singleton classes 481
Creating an object
  in C++ Objects 475
creating object 475
CSECT, adding to map assembly 716
CSNAP 468
CSPG transaction 355, 357, 407, 408
CSPP transaction 74
CSYSGRP 31
CTDLI 468
CTEST 468
CTLCHAR option 348
CTRACE 468
current channel
  example, with LINK commands 120
  example, with XCTL commands 122
  overview 120, 124
cursor
  in Finding out information about a terminal 496
CURSOR option
  ACCUM option 389
  SEND MAP command
    ACCUM option 389
cursor position 393
cursor positioning, symbolic 393
cursor-detectable field 436
cursor, finding the 402
cut
  in IccBuf constructors 483
CVDA 93
CWA 104
CWAKEY parameter 104

D

data
  chaining 268
  definition 97
  in Accessing start data 490
  in Finding out information about a terminal 496
  initialization 97
  passing to other program 148
  records 93
  storing in transaction 89
data area extensibility 482
Data area extensibility
  in Buffer objects 482
  in IccBuf class 482
data area ownership 482
Data area ownership

Data area ownership *(continued)*
in Buffer objects 482
in IccBuf class 482
data conversion
and channels
a SOAP example 141
data interchange block 581
data sets
access from CICS application programs 242
batch data interchange 272
BDAM 234
blocked 234
empty 224
sequential 102
user 93
data storing in transaction 89
data streams
compressing 440
inbound 438
RA order 440
repeat-to-address orders (SBA) 440
SBA order 440
set buffer address order 440
data tables
coupling facility 237
shared 236
data, moving to map 387
data, reading from a display 398
dataAreaOwner
in Data area ownership 482
dataAreaType
in Data area extensibility 482
databases and files
application requirements questions 551
exclusive control 566
locking 566
dataItems
in Example of polymorphic behavior 510
DATAONLY option 389, 390, 439
DATAREG operand 455
date field of EIB 12
date services 498
dateSeparator (parameter)
in Example of time and date services 498
dayOfMonth
in Example of time and date services 498
dayOfWeek
in Example of time and date services 498
daysSince1900
in Example of time and date services 498
DBCS option 582
DBRMs, production of 722
DCLGEN operations 725
DDname list, in translator dynamic invocation 590
DDS 383
deadlock
prevention 230
deadlock, transaction
avoiding 560
deadlocks 242
DEBKEY option 248
deblocking argument 234
DEBREC option 234, 248
debug 629
DEBUG option 582
debugging 629
debugging programs 501
Debugging Programs
in Compiling, executing, and debugging 501
default
action for condition 189
deferred journal output 187
definite response protocol
terminal control 269
definition of CICS
for recovery 553
DEFRESP option
terminal control 269
delay
in Support Classes 480
DELAY command 287, 288
delete
in Deleting an object 476
in Storage management 511
delete operator 475
DELETE statement, COBOL 513
deleteLockedRecord
in Deleting locked records 486
DELETEQ TD command 304
DELETEQ TS command 306
deleteRecord
in Deleting normal records 486
deleteRecord method 486
Deleting an object
in C++ Objects 476
deleting items 495
Deleting items
in Temporary storage 495
in Using CICS Services 495
Deleting locked records
in Deleting records 486
in File control 486
Deleting normal records
in Deleting records 486
in File control 486
deleting queues 493
Deleting queues
in Transient Data 493
in Using CICS Services 493
deleting records 251, 486
Deleting records
Deleting locked records 486
Deleting normal records 486
in File control 486
in Using CICS Services 486
deploy 750
DEPLOY APPLICATION command 742
DEPLOY APPLICATION command sample 747
DEPLOY BUNDLE command 738
DEPLOY BUNDLE command sample 747
deploying 673
deployment 673
DEQ command 289
DEQUEUE command 355
design considerations of applications
exclusive control of resources 100
designator character 436
designing a channel 131

DESTCOUNT option 424
DESTID option 273
DESTIDLENG option 273
destination identification 273
detectable field 436
device dependent support 383
device-dependent maps 382
DFH3QSS 13
DFHAID 468
DFHAPXPO 541
DFHASMVS procedure 709, 711, 717
DFHBMSCA 389, 468
DFHBMSCA definitions 402
DFHBMSUP 378
DFHCOMMAREA 149, 513
DFHCPLC 223
DFHCPLC, CPI Communications interface module 594
DFHCPLRR, SAA Resource Recovery interface module 594
DFHDHTL macro 314, 315
DFHDHTXD 316
DFHDHTXH 316
DFHDHTXL 316
DFHDHTXO 316
dfhdploy 736, 737, 747, 750
DFHDPLOY definition utility
  commands
    DEPLOY APPLICATION 742
    DEPLOY BUNDLE 738
    PERFORM 746
    SET 741, 744, 745
    SET CICSPLEX 737
    UNDEPLOY APPLICATION 743
    UNDEPLOY BUNDLE 739
DFHDPLOY examples 747
DFHDPLOY utility 750
DFHDYPDS 143
DFHEAG, interface module for assembler 620
DFHEAG0, interface module for assembler 620
DFHEAI, interface module for assembler 595
DFHEAI0, interface module for assembler 595
DFHEAP1$, translator for assembler 581
DFHECP1$, translator for COBOL 581
DFHEDF group 719
DFHEDP1$, translator for C 581
DFHEGTAL procedure 690, 692
DFHEIBLK 513
DFHEIEND macro 588, 589
DFHEIENT macro
  CODEREG 455
  DATAREG 455
  defaults 455
  EIBREG 455
DFHEIPLR symbolic register 455
DFHEIRET macro 458, 584, 587
DFHEISTG macro 588, 589
DFHEITAL procedure 690, 692
DFHEIVAR 513
DFHELII 595
DFHELII, interface module for Language Environment conforming compilers 595
DFHEPP1$, translator for PL/I 581
DFHEXEC 595
DFHEXTAL procedure 690, 692
DFHFCT macro 236
DFHLNKVS procedure 709, 717
DFHMAPS procedure 689, 713
DFHMAPT
  procedure for installing HTML templates 716
DFHMDF macro
  display characteristics 387
  DSATTS option 387
  MAPATTS option 387
DFHMDI macro 371, 373
DFHMIRS program 214
DFHMSD macro
  BASE option 386
  STORAGE option 386
DFHMSD, macro for assembling map sets 708
DFHMSRCA 435, 468
DFHNC001
  default named counter pool 331
DFHNCO macro
  named counter options table 331
DFHNCOPT
  named counter options table 331
DFHNCTR
  example COBOL call with null pointers 343
DFHPDI macro 429
DFHPEP program 204
DFHPLT macro 554
DFHPSD macro 429, 717
DFHRESP translator function 190, 581
DFHURLDS 421
DFHVALUE 581
DFHXLT macro 554
DFHYITDL procedure 690, 701
DFHYITEL procedure 690, 701
DFHYITPL procedure 690, 699
DFHYITVL procedure 690, 697
DFHYXTDL procedure 690, 701
DFHYXTEL procedure 690, 701
DFHYXTPL procedure 690, 699
DFHYXTVL procedure 690, 697
DFHZITCL procedure 690, 697
DFHZITPL procedure 690, 699
DIB 581
direct terminal 425
DISCONNECT command sample 747
discovering which containers were passed to a program 129
discovering which containers were returned from a link 129
display
  register, EDF 645
  screens 108
display characteristics 387
DISPLAY statement, COBOL 513
display, reading from 398
distributed application design 77
distributed program link
  client region 211
  COMMAREA option 214
  DPL API subset 218
  exception conditions 220
  independent syncpoints 214
  options 212
  programming considerations 218
  REMOTENAME option 214
  REMOTESYSTEM option 214
  server program 214

distributed program link *(continued)*
server region 211, 214
SYSID option 214
TRANSID option 214
DL/I
implicit enqueuing upon 571
references 8
scheduling
program isolation scheduling 571
syncpoints 189
DLI option 582
DLLs 540
DOCTEMPLATE resource definition
EXITPGM attribute 316
FILE attribute 314
HFSFILE attribute 313
MEMBERNAME attribute 313
PROGRAM attribute 314
TDQUEUE attribute 314
TSQUEUE attribute 314
DOCTOKEN 320, 329
DOCUMENT CREATE command
DELIMITER option 321, 323
DOCSIZE option 328
DOCTOKEN 320
LISTLENGTH option 321
SYMBOLLIST option 307, 308, 320, 321, 323
DOCUMENT DELETE command
DOCTOKEN 329
DOCUMENT INSERT command 307, 325, 327, 328
DOCUMENT RETRIEVE command
CLNTCODEPAGE option 312, 328
HOSTCODEPAGE option 312
DOCUMENT SET command
DELIMITER option 321, 323
LISTLENGTH option 321
SYMBOL option 307, 308, 321, 323
SYMBOLLIST option 307, 308, 321, 323
document templates
and CICS Web support 307, 312, 313
caching
specifying in exit programs 316
CICS file 314
CICS program 314, 315
communication area 316
DFHDHTL macro 314, 315
embedded commands
#echo 318
#include 318
#set 308, 318
exit program 316
HFS file 313
partitioned data set (PDS) 313
placing in documents 320, 325, 327
setting up 313
symbol lists 308
symbols 307, 308, 318
temporary storage queue 314
transient data queue 314
documenting recovery and restart programs 555
documents
adding data 325
and CICS Web support 307
bookmarks 325, 327
documents *(continued)*
code page conversion 312
creating 320
deleting 329
replacing data 327
retrieving 328
reusing 328
doSomething
in Using an object 475
DPL 146, 189, 209, 211, 642
DSA 97
DSATTS option 387
DSNTIAR 723
DTP 209, 222
DUPKEY condition 246
dynamic
program 99
storage area 97
transaction backout program 204
transaction routing 150
dynamic creation 475
dynamic deletion 475
dynamic invocation of translator 582
Dynamic LIBRARY 676
Dynamic Link Libraries 540
dynamic program link
affinity 182
dynamic routing with channels 136
dynamic storage, extensions 460
dynamic transaction backout
decision to use 563
dynamically-routed applications
migrating to channels and containers 146

E

ECBLIST 289
EDF 581, 584, 629
EDF option 582
EIB
description 12
EIBCALEN field 149
EIBCOMPL field 260
EIBFN field 150
terminal control feedback 267
EIBREG operand 455
empty
in Deleting items 495
in Deleting queues 493
in Temporary storage 494
in Transient Data 493
empty data sets 224
end-of-data indicator character 271
ENDBR command 245
endInsert
in Writing records 485
endl
in Example of terminal control 497
ENQ command 101, 289
ENQBUSY condition 101
ENQUEUE command 355
enqueuing
implicit enqueuing on DL/I databases 571
implicit enqueuing on nonrecoverable files 567

enqueuing *(continued)*
implicit enqueuing on recoverable files 569
implicit enqueuing on temporary storage queues 571
implicit enqueuing on transient data destinations 570
in a VSAM file 100
VSAM internal 100
entry point 626
entry-sequenced data set (ESDS) 224
EOC condition 268, 405
EODI character 271
EODS condition 268
EPILOG option 582
EQUAL option 243
erase
in Example of terminal control 497
in Sending data to a terminal 496
ERASE option 348, 389, 406
ERASEAUP option 389, 406, 415
ERDSA 686
error 735, 750
escape sequence 308, 323
ESDS
in File control 484
upgrading to extended addressing 229
ESDS (entry-sequenced data set) 224
ESDS file 484
events
monitoring point 207
Example of file control
in File control 487
in Using CICS Services 487
Example of managing transient data
in Transient Data 493
in Using CICS Services 493
Example of polymorphic behavior
in Miscellaneous 510
in Polymorphic Behavior 510
Example of starting transactions
in Starting transactions asynchronously 490
in Using CICS Services 490
Example of Temporary Storage
in Temporary storage 495
in Using CICS Services 495
Example of terminal control
in Terminal control 497
in Using CICS Services 497
Example of time and date services
in Time and date services 498
in Using CICS Services 498
examples
channels, basic 110
CICS client program that contructs a channel 133
CICS server program that uses a channel 133
containers, basic 110
multiple interactive components 116
one channel—one program 112
one channel—several programs 114
several channels, one component 116
simple client program compared to a BTS activity 134
exception condition
description 189
exception conditions
HANDLE CONDITION command 193, 195
IGNORE CONDITION command 197
Exception handling (throwException)
in CICS conditions 505
in Conditions, errors, and exceptions 505
exceptions 501
EXCI
CALL 223
communications 223
option 582
EXCI - external call interface 209
exclusive control of records
BDAM 251
VSAM 230
VSAM RLS 231
exclusive resources 100
EXEC interface block 581
EXEC interface modules 594, 595, 620, 703
EXEC interface stubs 595, 620
EXECKEY 90, 104
EXECKEY parameter 293
Executing Programs
in Compiling, executing, and debugging 500
execution diagnostic facility 581, 584, 629
exit program
as document template
communication area 316
expiration time
specifying 288
EXPLAIN 727
extended read-only DSA (ERDSA) 686
extended relative byte address (XRBA) 228
extended relative byte addressing (XRBA)
upgrading 229
external call interface (EXCI) 209
External CICS interface (EXCI) 223
extrapartition queues 305
extrapartition transient data 93, 102
EYUVALUE 581

F

familyConformanceError
in C++ Exceptions and the Foundation Classes 503
FEPI
references 8
FEPI - Front End Programming Interface 209
FETCH 468
FETCHABLE option 550
field
blank 438
group 378
fields
BMS 370
complex 378
composite 378
repeated 379
file (parameter)
in Example of file control 487
file control
BDAM data sets 234
browsing records 487
deleting records 486
example 487
overview 224
rewriting records 486

file control *(continued)*
updating records 486
File control
Browsing records 487
Deleting locked records 486
Deleting normal records 486
Deleting records 486
Example of file control 487
in Using CICS Services 484
Reading ESDS records 485
Reading KSDS records 485
Reading records 485
Reading RRDS records 485
Updating records 486
Writing ESDS records 486
Writing KSDS records 486
Writing records 485
Writing RRDS records 486
file control recovery control program
exits 573
FIND command, CEBR transaction 654
Finding out information about a terminal
in Terminal control 496
in Using CICS Services 496
finding the cursor 402
flag byte, route list 421
FLAG option 582
flip-flop mode 258
FLOAT compiler option 547
floating maps 410
flush
in Example of terminal control 497
FMH
inbound 270
option 270
outbound 270
FMHPARM option 406
for
in Example of file control 488
FOR option 288
Form
in Polymorphic Behavior 509
format (parameter)
in Example of time and date services 498
formatted screen, reading from a 399
FORMATTIME command 287
FORMFEED option 349, 406
Foundation Class Abend codes
in Conditions, errors, and exceptions 501
FREE command 270
FREEKB option 389
freeKeyboard
in Sending data to a terminal 496
FREEMAIN command 292
FROM option 389
Front End Programming Interface (FEPI) 209
FRSET option 389
fsAllowPlatformVariance
in Platform differences 506
fsEnforce
in Platform differences 506
function (PF) keys, CEBR transaction 653
function key 632, 661
function management header
function management header *(continued)*
description 269
function shipping 210
Function shipping 209
function-ship 250
functions, EDF 629

G

GDDM 391
GDS option 582
generic delete 251
generic key 243, 485
GENERIC option 228, 243
get
in Example of polymorphic behavior 510
in Polymorphic Behavior 509
GET command, CEBR transaction 655
GETMAIN command
CICSDATAKEY option 89, 296
INITIMG option 98, 292
NOSUSPEND option 292
SHARED option 89, 93, 292
TASKDATAKEY option 89
USERDATAKEY option 89, 296
GETMAIN64 command
SHARED option 93
global user exits
XFCBFAIL 574
XFCBOUT 575
XFCBOVER 574
XFCLDEL 574
global variables in COBOL 525
GOODNIGHT transaction, CESF 272
GRANT command 727
GRAPHIC option 582
group field 378
GRPNAME option 378
GTEQ option 228, 243

H

half-duplex mode 258
HANDLE ABEND 198
HANDLE ABEND command 193, 204, 563, 564
HANDLE ABEND LABEL restriction with assembler language 449
HANDLE AID command 401
HANDLE CONDITION command 193, 198, 563
HANDLE CONDITION ERROR command 196
handleEvent
in Automatic condition handling (callHandleEvent) 504
Header files
in Installed contents 499
HFS file
as document template 313
HOLD option 146
HONEOM option 348
horizontal tabs 350
host code page 312
HTML templates
installing 716

I

IBM Screen Definition Facility II (SDF II) 706
IBM-supplied classes
  example for user-defined resources 209
IBMWRLKC linkage editor input 548
Icc
  in Method level 506
  in Overview of the foundation classes 476
Icc::initializeEnvironment
  in Storage management 511
IccAbendData
  in Singleton classes 481
IccAbsTime
  in Base classes 477
  in Support Classes 480
  in Time and date services 498
IccAbsTime,
  in Support Classes 480
IccBase
  in Base classes 476
  in Resource classes 478
  in Resource identification classes 477
  in Storage management 511
  in Support Classes 480
IccBase class
  overview 476
IccBuf
  in Buffer objects 481, 482
  in C++ Exceptions and the Foundation Classes 503
  in Data area extensibility 482
  in Data area ownership 482
  in Example of file control 487, 488
  in Example of managing transient data 494
  in Example of polymorphic behavior 510
  in Example of starting transactions 491, 492
  in Example of Temporary Storage 495, 496
  in Example of terminal control 497
  in IccBuf class 482
  in IccBuf constructors 482, 483
  in IccBuf methods 483
  in Reading data 493
  in Reading items 494
  in Scope of data in IccBuf reference returned from 'read' methods 512
  in Support Classes 480
  in Working with IccResource subclasses 483, 484
IccBuf class
  constructors 482
  data area extensibility 482
  Data area extensibility 482
  data area ownership 482
  Data area ownership 482
  IccBuf constructors 482
  IccBuf methods 483
  in Buffer objects 482
  methods 483
  Working with IccResource subclasses 483
IccBuf constructors
  in Buffer objects 482
  in IccBuf class 482
IccBuf methods
  in Buffer objects 483
  in IccBuf class 483
IccBuf reference 512
IccClock
  in Example of time and date services 498
  in Time and date services 498
IccCondition
  in C++ Exceptions and the Foundation Classes 503
IccConsole
  in Buffer objects 482
  in Object level 506
  in Singleton classes 481
IccConsole class
  overview 481
IccControl
  in Base classes 476
  in Example of starting transactions 491, 492
  in Method level 506
  in Singleton classes 481
  in Support Classes 480
IccControl class
  overview 476, 481
IccDataQueue
  in Buffer objects 482
  in Example of managing transient data 493, 494
  in Example of polymorphic behavior 510
  in Resource classes 478
  in Temporary storage 494
  in Transient Data 493
  in Working with IccResource subclasses 484
  in Writing data 493
IccDataQueueId
  in Example of managing transient data 493
  in Transient Data 493
IccEvent
  in Support Classes 480
IccException
  in C++ Exceptions and the Foundation Classes 503
  in Method level 506
  in Object level 506
  in Parameter level 507
  in Support Classes 480
IccException class
  CICSCondition type 503
  familyConformanceError type 503
  internalError type 503
  invalidArgument type 503
  invalidMethodCall type 503
  objectCreationError type 503
IccFile
  in Browsing records 487
  in Buffer objects 482
  in C++ Exceptions and the Foundation Classes 503
  in Deleting locked records 486
  in Deleting normal records 486
  in Example of file control 487
  in File control 484
  in Reading ESDS records 485
  in Reading KSDS records 485
  in Reading records 485
  in Reading RRDS records 485
  in Resource identification classes 477
  in Singleton classes 481
  in Updating records 486
  in Writing ESDS records 486
  in Writing KSDS records 486

IccFile *(continued)*
in Writing records 485
in Writing RRDS records 486
IccFile class
deleteLockedRecord 486
deleteRecord method 486
isReadable method 485
keyLength method 485
keyPosition method 485
readRecord method 485
recordFormat method 485
recordIndex method 485
recordLength method 485
registerRecordIndex 485
registerRecordIndex method 485
rewriteRecord method 486
writeRecord method 485
IccFile::readRecord
in Scope of data in IccBuf reference returned from 'read' methods 512
IccFileId
in Base classes 477
in File control 484
in Resource identification classes 477
IccFileId class
overview 477, 484
reading records 484
IccFileIterator
in Browsing records 487
in Buffer objects 482
in Example of file control 487, 488
in File control 484
IccFileIterator class
overview 484
readNextRecord method 487
readPreviousRecord 487
IccJournal
in Buffer objects 482
in Object level 506
IccKey
in Browsing records 487
in Deleting normal records 486
in File control 484
in Reading KSDS records 485
in Reading records 485
in Writing KSDS records 486
in Writing records 485
IccKey class
reading records 484
IccMessage
in Support Classes 480
IccProgram
in Buffer objects 482
in Program control 488, 489
in Resource classes 478
IccProgram class
program control 488
IccProgramId
in Resource identification classes 477
IccRBA
in Browsing records 487
in File control 484
in Reading ESDS records 485
in Reading records 485
IccRBA *(continued)*
in Writing ESDS records 486
in Writing records 485
in Writing RRDS records 486
IccRBA class
reading records 484
IccRecordIndex
in C++ Exceptions and the Foundation Classes 503
IccRequestId
in Example of starting transactions 491, 492
in Parameter passing conventions 512
IccResource
in Base classes 476
in Example of polymorphic behavior 510
in Polymorphic Behavior 509
in Resource classes 479
in Scope of data in IccBuf reference returned from 'read' methods 512
IccResource class
overview 476
working with subclasses 483
IccResourceId
in Base classes 476, 477
in C++ Exceptions and the Foundation Classes 503
in Resource identification classes 477
IccResourceId class
overview 476, 477
IccRRN
in Browsing records 487
in Deleting normal records 486
in File control 484
in Reading records 485
in Reading RRDS records 485
in Writing records 485
IccRRN class
reading records 484
IccSession
in Buffer objects 482
IccStartRequestQ
in Accessing start data 490
in Buffer objects 482
in Example of starting transactions 491, 492
in Parameter passing conventions 511
in Singleton classes 481
in Starting transactions asynchronously 490
IccStartRequestQ class
overview 481
IccSysId
in Program control 489
IccSystem
in Singleton classes 481
IccSystem class
overview 481
IccTask
in C++ Exceptions and the Foundation Classes 503
in Example of starting transactions 492
in Parameter level 506
in Singleton classes 481
in Support Classes 480
IccTask class
overview 481
IccTask::commitUOW
in Scope of data in IccBuf reference returned from 'read' methods 512

IccTempstore
in Working with IccResource subclasses 483
IccTempStore
in Automatic condition handling (callHandleEvent) 504
in Buffer objects 482
in C++ Exceptions and the Foundation Classes 503
in Deleting items 495
in Example of polymorphic behavior 510
in Example of Temporary Storage 495
in Reading items 494
in Resource classes 478
in Temporary storage 494
in Transient Data 493
in Updating items 495
in Working with IccResource subclasses 483, 484
in Writing items 495
IccTempStore::readItem
in Scope of data in IccBuf reference returned from 'read' methods 512
IccTempStore::readNextItem
in Scope of data in IccBuf reference returned from 'read' methods 512
IccTempStoreId
in Base classes 477
in Example of Temporary Storage 495
in Temporary storage 494
IccTermId
in Base classes 476
in C++ Exceptions and the Foundation Classes 503
in Example of starting transactions 491
in Example of terminal control 497
in Terminal control 496
IccTermId class
overview 476
IccTerminal
in Buffer objects 482
in Example of terminal control 497
in Finding out information about a terminal 496
in Receiving data from a terminal 496
in Resource classes 478
in Singleton classes 481
in Terminal control 496
IccTerminal::receive
in Scope of data in IccBuf reference returned from 'read' methods 512
IccTerminalData
in Example of terminal control 497
in Finding out information about a terminal 496
in Terminal control 496
IccTime
in Base classes 477
in Parameter passing conventions 512
in Support Classes 480
IccTime class
overview 477
IccTimeInterval
in Base classes 477
in Example of starting transactions 491, 492
in Support Classes 480
IccTimeOfDay
in Base classes 477
in Support Classes 480
IccTransId
in Base classes 476
IccTransId *(continued)*
in Example of starting transactions 491
in Parameter passing conventions 512
IccTransId class
overview 476
IccUserControl
in C++ Exceptions and the Foundation Classes 502
in Example of file control 487
in Example of managing transient data 493
in Example of polymorphic behavior 510
in Example of starting transactions 491
in Example of Temporary Storage 495
in Example of terminal control 497
in Example of time and date services 498
in Program control 489
in Singleton classes 481
ICTL (input format control) assembler instruction 449
ICVR parameter 719
Id
in Resource identification classes 477
identification
BDAM record 234
VSAM record 228
IGNORE CONDITION command 193, 197
IGREQID condition 406
IGYCCICS 513
IGZWRLKA 517
IMMEDIATE option 150, 258, 270
IMS.RESLIB (IMS library) 703
INBFMH condition 268
inbound
data streams 438
index, alternate 224
indirect queues 305
initialization (PLT) programs
defining 554
initialization and termination exit
for transaction backout 574
initializeEnvironment
in Method level 506
in Storage management 511
initializing output map 386
INITIMG option 98, 292
input data
chaining of 268
input map, symbolic 398
INPUTMSG option 146, 150
INQUIRE command 11
INQUIRE TERMINAL command 267, 351
INRTN option 431
insert
in Example of Temporary Storage 496
in IccBuf constructors 483
installing application programs
assembler language 692
C 699
COBOL 695
PL/I 699
using your own job stream 703
installing assembly application programs
sample job stream for 692
Installing HTML templates 716
instance
in Singleton classes 481

integrated CICS translator 577
integrated translators 12, 577
inter-transaction affinity
  affinity life times 179
  affinity transaction groups 179
  programming techniques 158
  relations and lifetimes
    BAPPL relation 180
    global relation 181
    terminal relation 184
    userid relation 185
  safe programming techniques
    the COMMAREA 159
    the TCTUA 160
    using BTS containers 162
    using DEQ with ENQMODEL 162
    using ENQ with ENQMODEL 162
  suspect programming techniques
    DELAY and CANCEL REQID commands 176
    global user exits 157
    INQUIRE and SET commands 157
    POST command 178
    RETRIEVE WAIT and START commands 173
    START and CANCEL REQID commands 175
    transient data 172
  temporary storage 170
  temporary storage data-sharing 170
  unsafe programming techniques
    the CWA 163
    using DEQ 168
    using ENQ 168
    using LOAD PROGRAM HOLD 165
    using shared storage 164
    using task lifetime storage 166
    using WAIT EVENT 167
inter-transaction communication 558
interactive debugging
  CECI transaction 656
  CECS transaction 661
  CEDF transaction 629
intercommunication 209
interface modules
  CPI Communications 594
  EXEC 594
  programs using EXEC CICS or EXEC DLI commands 703
  SAA Resource Recovery 594
  using 691
interface stubs, EXEC 595, 620
interleaving conversation with message routing 425
internalError
  in C++ Exceptions and the Foundation Classes 503
interregion communication 223
interrupting 259
intertransaction communication
  use of COMMAREA 559
  use of resources 559
interval control
  cancel interval control command 287
  DELAY command 288
  delay processing of a task 287
  expiration time 288
  POST command 288
  specifying request identifier 287
  START command 288
interval control *(continued)*
  starting a task 287
interval control START requests 561
INTERVAL option 288
intrapartition queues 304
intrapartition transient data
  implicit enqueuing upon 570
intrapartition transient data queues 9
invalidArgument
  in C++ Exceptions and the Foundation Classes 503
invalidMethodCall
  in C++ Exceptions and the Foundation Classes 503
INVITE option 258
INVOKE APPLICATION 626
invoking EDF 630
INVPARTN condition 432
INVPARTNSET condition 432
INVREQ condition 406
IOERR condition processing 566
IRC 223
isAddable
  in Writing ESDS records 486
  in Writing KSDS records 486
  in Writing RRDS records 486
iscics 13
ISCINVREQ 210
isolation level 724
ISR2
  in Example of starting transactions 491
isReadable
  in Reading ESDS records 485
  in Reading KSDS records 485
  in Reading RRDS records 485
isReadable method 485
ISSUE ABORT command 273
ISSUE ADD command 272
ISSUE COPY command 259, 357
ISSUE DISCONNECT command 259
ISSUE END command 273
ISSUE ERASE command 259, 273
ISSUE NOTE command 273
ISSUE PRINT command 357
ISSUE QUERY command 272
ISSUE RECEIVE command 272
ISSUE REPLACE command 273
ISSUE SEND command 272
ISSUE WAIT command 273
ITMP
  in Example of starting transactions 491

J

JCICS
  and channels 134
JCICSX 134
JES 11, 358
JES (job entry subsystem)
  exits 362
  input 362
  RESP and RESP2 options 360
  retrieve data from JES spool 358
  spooler commands 360
Job Entry Subsystem component of MVS 358
journal

journal *(continued)*
records 93, 186
journal control
output synchronization 187
journal identifier 188
journal type identifier 188
journaling 102, 186
JOURNALNAME 188
JOURNALNUM 188
JTYPEID 188
JUSFIRST option 415
JUSLAST option 415
JUSTIFY option 415

K

key
alternate (secondary) 224
complete 485
generic 243, 485
hardware print 357
key (parameter)
in Example of file control 487
key-sequenced data set (KSDS) 224
keyLength
in Reading KSDS records 485
in Writing KSDS records 486
keyLength method 485
keyPosition
in Reading KSDS records 485
in writing KSDS records 486
keyPosition method 485
keys
physical 234
KSDS
in File control 484
KSDS (key-sequenced data set) 224
KSDS file 484

L

L8 and L9 mode TCB
restrictions 87
Language Environment
abend handling
PL/I 548
AID handling 535
assembler language 453
callable services 534
CEEBINT 545
CICSVAR environment variable 544
compilers supported 533
condition handlers for Language Environment 535
condition handlers, user-written 535
condition handling 535
DLLs 540
dump destination 534
Dynamic Link Libraries 540
HANDLE AID command 535
HANDLE CONDITION command 535
HLL user exit 545
levels of support 533
message destination 534
Language Environment *(continued)*
mixing languages 538
PL/I 548
runtime options
and CICS LINK 541
CBLPSHPOP 516
CEEBXITA user exit 543
CEECSTX user exit 543
in child enclaves 541
storage 537
support, levels of 533
VS COBOL II 517
large COMMAREAs 108, 109, 112, 118, 134
LAST option
bracket protocol 270
LDC 433
LDCMNEM option 433
LDCNUM option 433
LEASM option 582
LENGERR condition 260
LENGTH option 93, 260, 582
levels, application program logical 147
LIBRARY 676
library lookaside (LLA) 685
light pen-detectable field 436
line
in Finding out information about a terminal 496
LINE command
CEBR transaction 655
line length on printing 348
line traffic reduction 439
LINECOUNT option 582
LINK
migrating to channels and containers 143
LINK command
CHANNEL option 146
COMMAREA option 146, 149, 150
IMMEDIATE option 150
INPUTMSG option 146, 150
TRANSID option 150
link pack area (LPA) 686
LINK PROGRAM 223
link to program, expecting return 147
link-edit 577, 581
link-edit of map 376
LINKAGE option 582
LIST option 419
LLA (library lookaside) 685
LOAD command
HOLD option 146
load libraries
support for secondary extents 691
local copy key 357
locale support in C and C++ 473
locality of reference 98
LOCKED affinity relation 182
locking
implicit locking on nonrecoverable files 567
implicit locking on recoverable files 569
in application programs 566
logging 103
logical device component 433
logical levels, application program 147, 520
logical messages, rules for 406

logical record presentation 269
logical unit of work (LUW)
  syncpoints used 188
logical units (LUs)
  facilities for 268
lookaside transaction 429
LPA 686
LU type 4
  batch data interchange 273
  device 259
  logical record presentation 269
LUs (logical units)
  facilities for 268

M

magnetic slot reader, 10/63 435
main
  in C++ Exceptions and the Foundation Classes 502
  in Example of file control 487
  in Example of managing transient data 493
  in Example of polymorphic behavior 510
  in Example of starting transactions 491
  in Example of Temporary Storage 495
  in Example of terminal control 497
  in Example of time and date services 498
  in Header files 500
  in Program control 489
  in Storage management 511
MAIN option 582
main storage 97
main temporary storage 91
management bundle 31
management part 58
Manual condition handling (noAction)
  in CICS conditions 504
  in Conditions, errors, and exceptions 504
map
  BMS 369
  creating 371
  initializing output 386
  link-edit 376
  moving data to 387
  sets 377
  symbolic input 398
  symbolic output 397
map sets
  Adding a CSECT 716
  Using symbolic description map sets in a program 689
MAPATTS option 387
MAPCOLUMN option 413
MAPFAIL condition 400, 405
MAPHEIGHT option 413
MAPLINE option 413
MAPONLY option 389, 390, 439
MAPPED option 417
mapping input data 398
maps
  BMS 438–440
  device-dependent 382
  floating 410
  sets 99
MAPSET option 389
MAPSET resource definition 376
mapsets
  loading above the 16MB line 706
MAPWIDTH option 413
MARGINS option 582
MASSINSERT option 252, 255
MDT 400, 438
MERGE statement, COBOL 513
message routing 418
message title 423
messages, undeliverable 423
Method level
  in Conditions, errors, and exceptions 506
  in Platform differences 506
MGMTPART 58
migrating START data to use channels 145
migration
  of START data to use channels 145
mirror transactions
  availability of 6
Miscellaneous
  Example of polymorphic behavior 510
mixed addressing mode transaction 152
mixing languages 538
modernising COMMAREAs 109
modified data tag 400, 438
modifying execution, EDF 643
modular program 99
MONITOR command 207
MONITOR POINT command 207
monitoring application performance 207
monthOfYear
  in Example of time and date services 498
moving data to map 387
MSGINTEG option 274
MSR 435
MSR option 406, 435
multi-version 626
multi-versioned applications 626
multimap screens 440
multipage outputs 440
multiple base registers 455
multithread testing 718
multithreading 78
MVS subspace 302
MVS transaction 152
MVS/ESA
  in Storage management 511
MXT parameter 100
MyTempStore
  in Automatic condition handling (callHandleEvent) 505

N

named counters
  CICS API 332
  counter name 331
  coupling facility list structure 331
  current value 331
  DFHNC001 331
  DFHNCO macro 331
  maximum value 331
  minimum value 331
  named counter fields 331
  options table 331

named counters *(continued)*
overview 330
pools 331
national characters
uppercase translation 621
native runtime libraries 533
NATLANG option 582
nested programs in COBOL 528
new
in Storage management 511
new operator 475
NLEOM option 347–349, 406
noAction
in CICS conditions 504
NOAUTOPAGE option 408
NOCBLCARD option 582
NOCPSM option 582
NODE option 361
NODEBUG option 582
NOEDF option 582
NOEDIT option 417
NOEPILOG option 582
NOEPILOG translator option 455
NOFEPI option 582
NOFLUSH option 410, 415
NOHANDLE option 190, 195
NOJBUFSP condition 101
NOLENGTH option 582
NOLINKAGE option 582
NOMAIN option 582
non-CICS printer 346
Non-CICS printer 355
non-terminal transactions
EDF 641
NONUM option 582
NOOPSEQUENCE option 582
NOOPTIONS option 582
NOPROLOG option 582
NOPROLOG translator option 455
NOQUEUE option 101
NOSEQ option 582
NOSEQUENCE option 582, 590
NOSOURCE option 582
NOSPACE condition 196
NOSPIE option 582
NOSUSPEND option
GETMAIN command 292
READ command 243
READNEXT command 251
READPREV command 251
WRITE command 252
NOTRUNCATE option 260
NOVBREF option 582
NOWAIT option 273
NOXREF option 582
null parameters, example of DFHNCTR CALLs with 343
null values, use of 440
NUM option 582
number
in Writing RRDS records 486
NUMREC option 251

O

obj (parameter)
in Using an object 475
object
creating 475
deleting 476
using 475
Object level
in Conditions, errors, and exceptions 506
in Platform differences 506
objectCreationError
in C++ Exceptions and the Foundation Classes 503
OCCURS option 379
on-units 566
OO COBOL
support 513
OPCLASS option 419
OPEN statement, COBOL 513
open transaction environment (OTE) 80
Open Transaction Environment (OTE)
CONCURRENCY(REQUIRED) programs 85
OPENAPI programs
restrictions 87
OPENAPI
restrictions 87
operating system waits 101
operator«
in Working with IccResource subclasses 484
operator=
in Example of file control 488
in Working with IccResource subclasses 483, 484
OPID option 419
OPIDENT value 419
OPMARGINS option 582
OPSEQUENCE option 582
OPSYN (equate operation code) assembler instruction 449
options
HANDLE CONDITION command 195
on function keys, EDF 644
OPTIONS option 582
OPTIONS(MAIN) specification 547
OTE, open transaction environment 80
outboard controller 272
outboard formatting 437
output data, chaining of 268
output map, initializing 386
output map, symbolic 397
OVERFLOW condition 410
overlays 99
overtyping EDF displays 643
overview
dynamic routing with channels 136
overview of Foundation Classes 476
Overview of the foundation classes
Calling methods on a resource object 481
Creating a resource object 480

P

PA key 351
packaging applications 625
page break 410
page building operations 440

page fault 97
page overflow 424
page routing operations 440
PAGENUM option 424
PAGESIZE value 409
paging
  reducing effects 97
PAGING option 348, 389, 394
Parameter level
  in Conditions, errors, and exceptions 506
  in Platform differences 506
parameter passing 511
Parameter passing conventions
  in Miscellaneous 511
parameters
  null 343
partition sets
  installing 717
  loading above the 16MB line 706
partition, active 431
partitioned data set (PDS)
  as document template 313
partitions 427
partitions, defining 429
PARTITIONSET option 430
PARTN option 431
partner, conversation 258
partners, range of 222
PARTNFAIL condition 432
PARTNPAGE option 432
passing control, anticipating return (LINK) 147
passing data to other program 148
pen-detectable field 436
PERFORM command 11, 746
PF (program function) key 652
physical keys 234
physical map sets
  installing 709
PL/I
  fetched procedures 550
  FLOAT compiler option 547
  Language Environment requirements 548
  mixing with other languages 538
  OPTIONS(MAIN) specification 547
  programming techniques 547, 548, 550
  programs and error handling
    on-units 566
  restrictions 547
PLATDIR 31
platform
  application binding 49
  application status 58
platform bundle 31
platform differences
  method level 506
  object level 506
  parameter level 506
Platform differences
  in Conditions, errors, and exceptions 505
  Method level 506
  Object level 506
  Parameter level 506
PLATHOME 31
PLT (program list table)
PLT (program list table) *(continued)*
  definition of 554
PLT program 296
plus 32 K COMMAREAs 108
plus 32K COMMAREAs 118, 134
policy 31
polymorphic behavior 507
Polymorphic Behavior
  Example of polymorphic behavior 510
  in Miscellaneous 507
POP HANDLE command 193, 198, 204
POST command 287, 288
postinitialization (PLT) programs
  (initialization programs)
    defining 554
pre-translated code 703
preprinted form 306
presentation space 428
PRGDLY option 423
principal facility 73, 74
print
  in Polymorphic Behavior 509
print control bit 348
print formatting 272
print key 357
print key, Hardware 357
printed output, requests for 346
printer
  3270
    options 348
  CICS
    determining characteristics of 351
  non-CICS 346
  Non-CICS 355
  SCS 349
PRINTERCOMP option 350
printing
  CICS API considerations 356
  line length on 348
  START command 353
  transient data 354
  with BMS routing 355
program
  as document template 314
  size 96
  testing 629
program control
  example 488
  introduction 488
  linking to another program 147
  passing data to another program 148
  program logical levels 147
Program control
  in Using CICS Services 488
program design
  conversational 274
program error program (PEP)
  design considerations 564
program isolation scheduling 571
program labels in EDF 643
PROGRAM option 204
PROGRAM resource definition 376
program storage 90
programId

programId *(continued)*
in Method level 506
programming models 73
programming restrictions
assembler language 449
C and C++ 468
COBOL 513, 522, 525
PL/I 547
VS COBOL II 517
programming techniques
assembler language 449, 453, 458
assembly language 458
C and C++ 468
COBOL 513, 518, 522
general 96
PL/I 547, 548, 550
PROLOG option 582
propagation 53
PROTECT option 274
pseudoconversational processing 558
PURGE command, CEBR transaction 655
purge delay 423
PURGE MESSAGE command 407, 418
PUSH HANDLE command 193, 198, 204
put
in Example of polymorphic behavior 510
in Polymorphic Behavior 509
PUT command, CEBR transaction 655

Q

QBUSY condition 101
QUERY SECURITY command
NOLOG option 208
query transaction 74
QUEUE command, CEBR transaction 655
queueName
in Accessing start data 490
queues
extrapartition 305
intrapartition 304
transient data 303
QUOTE option 582
QZERO condition 305

R

RACF 208
range of partners 222
RBA 484
RBA (relative byte address) 224, 228
RDF 255
READ command
CONSISTENT option 243
NOSUSPEND option 243
REPEATABLE option 243
UNCOMMITTED option 243
read only containers 133
READ statement, COBOL 513
read-ahead queueing 259
read-only DSA (RDSA) 686
reading data 493
Reading data
in Transient Data 493
in Using CICS Services 493
reading data from a display 398
Reading ESDS records
in File control 485
in Reading records 485
reading from a formatted screen 399
reading items 494
Reading items
in Temporary storage 494
in Using CICS Services 494
Reading KSDS records
in File control 485
in Reading records 485
reading records 242
Reading records
in File control 485
in Using CICS Services 485
Reading ESDS records 485
Reading KSDS records 485
Reading RRDS records 485
Reading RRDS records
in File control 485
in Reading records 485
readItem
in Example of Temporary Storage 496
in Reading data 493
in Reading items 494
in Scope of data in IccBuf reference returned from 'read' methods 512
in Temporary storage 494
in Transient Data 493
in Working with IccResource subclasses 483, 484
READNEXT command
NOSUSPEND option 251
readNextItem
in Scope of data in IccBuf reference returned from 'read' methods 512
in Temporary storage 494
readNextRecord
in Browsing records 487
readNextRecord method 487
READPREV command
NOSUSPEND option 251
readPreviousRecord
in Browsing records 487
READQ TD command 101, 304
READQ TS command
ITEM option 306
readRecord
in C++ Exceptions and the Foundation Classes 503
in Deleting locked records 486
in Reading records 485
in Updating records 486
readRecord method 485
receive
in Receiving data from a terminal 496
RECEIVE command
MAPFAIL condition 405
RECEIVE MAP command
ASIS option 400
RECEIVE PARTN command 431
receive3270data

receive3270data *(continued)*
in Receiving data from a terminal 496
receiving data from a terminal 496
Receiving data from a terminal
in Terminal control 496
in Using CICS Services 496
record
identification 228, 234
locking 230
locking (RLS) 231
record description field 255
record locking 231
record-level sharing (RLS)
accessing files in RLS mode 226
recordFormat
in Reading ESDS records 485
in Reading RRDS records 485
in Writing ESDS records 486
in Writing RRDS records 486
recordFormat method 485
recordIndex
in Reading ESDS records 485
in Reading KSDS records 485
in Reading RRDS records 485
in Writing ESDS records 486
in Writing KSDS records 486
in Writing RRDS records 486
recordIndex method 485
recordLength
in Reading ESDS records 485
in Reading KSDS records 485
in Reading RRDS records 485
in Writing ESDS records 486
in Writing KSDS records 486
in Writing RRDS records 486
recordLength method 485
records
adding 252
adding to BDAM data set 254
browsing 242
deleting 251
journal 186
length of 93
reading 242
updating 249
writing 249, 252
recoverability
CICS required definitions 554
recovery
of resources 100
problem avoidance 94
sequential terminal support 270
syncpoint 188
reduction of line traffic 439
reentrancy 78
reference modification 525
reference set 99
REGIONTYPE 31
registerData
in Example of starting transactions 491
in Starting transactions 490
registerRecordIndex
in Reading ESDS records 485
in Reading KSDS records 485
registerRecordIndex *(continued)*
in Reading RRDS records 485
in Writing ESDS records 486
in Writing KSDS records 486
in Writing records 485
in Writing RRDS records 486
registerRecordIndex method 485
regression testing 718
relative byte address 484
relative byte address (RBA) 224, 228
relative byte addressing (RBA)
upgrading 229
relative record data set (RRDS) 224
relative record number 484
relative record number (RRN) 224, 228
RELEASE 468
RELEASE command
HOLD option 146
RELEASE option 408
RELTYPE keyword 234
remote transactions, EDF 641
remote-linked programs
DPL 642
EDF 642
REMOTENAME option 214
REMOTESYSTEM option 214
remoteTermId
in Example of starting transactions 491
RENT attribute 686
REPEATABLE option
READ command 243
repeatable read 724
repeated fields 379
replace
in IccBuf constructors 483
REPLACE statement 525
req
in Example of starting transactions 492
req1
in Example of starting transactions 491
req2
in Example of starting transactions 491
REQID option 246, 287, 406, 423
request/response unit (RU) 268
REQUIRED 85
reset
in Browsing records 487
RESETBR command 245
residence mode (RMODE)
options for CICS applications 684
resource class 478
Resource classes
in Overview of the foundation classes 478
resource identification class 477
Resource identification classes
in Overview of the foundation classes 477
resource object
creating 480
resource recovery
SAA compatibility 557
resources
controlling sequence of access to 290
exclusive control of 100
recoverable 100

RESP and RESP2 options
for interface to JES 360
RESP option
deactivating NOHANDLE 195
RESP value 189
RESP2 option 190
RESP2 value 189, 190
restart 552
restrictions
31-bit addressing 449
64–bit registers 449
assembler language 449
C and C++ 468
COBOL 513, 522, 525
PL/I 547
RETAIN option 724
RETPAGE condition 395, 415
RETRIEVE command 287, 290
retrieveData
in Accessing start data 490
RETURN
migrating to channels and containers 144
return code 735, 750
RETURN command
CHANNEL option 108, 146
COMMAREA option 107, 146
IMMEDIATE option 258, 270
INPUTMSG option 146, 150
returnTermId
in Accessing start data 490
returnTransId
in Accessing start data 490
reverse interrupt 259
REWRITE command 249
REWRITE statement, COBOL 513
rewriteItem
in Example of Temporary Storage 496
in Temporary storage 494
in Updating items 495
in Writing items 494
rewriteRecord
in Updating records 486
rewriteRecord method 486
rewriting records 486
RIDFLD option 234, 245
RMODE (residence mode)
options for CICS applications 684
RMODE compiler option 513
ROLLBACK
considerations for use 563
ROUTE command
LIST option 419
page overflow 424
TITLE option 423
route list
LIST option 421
segment chain entry format 421
standard entry format 421
ROUTEDMSGS option 419
routing terminals 425
routing, Transaction 209
RPTOPTS 541
RRDS (relative record data set) 224
RRDS file
RRDS file *(continued)*
in File control 484
RRN 484
RRN (relative record number) 224, 228
RTEFAIL condition 421
RTESOME condition 421
RU (request/response unit) 268
rules for logical messages 406
run
in Base classes 476
in C++ Exceptions and the Foundation Classes 502
in Example of file control 487, 488
in Example of managing transient data 493, 494
in Example of polymorphic behavior 510
in Example of starting transactions 491
in Example of Temporary Storage 495, 496
in Example of terminal control 497
in Example of time and date services 498, 499
in Program control 489
run unit 520
runaway tasks 101
runtime libraries
Language Environment 533
native 533
runtime options for Language Environment
STORAGE 513
RUWAPOOL 537
RVI 259

S

SAA Resource Recovery 188
SAA resource recovery interface 557
SAA Resource Recovery interface module, DFHCPLRR 594
SAM 257
scenarios
multiple interactive components 116
one channel—one program 112
one channel—several programs 114
several channels, one component 116
scope of a channel
example, with XCTL commands 127
scope of data 512
Scope of data in IccBuf reference returned from 'read' methods
in Miscellaneous 512
scope of references 512
screen copy, BMS 357
screen field, 3270 400
screen, reading from a formatted 399
SCS
printer 349
SCS input 351
SDF II 377
SDF II (IBM Screen Definition Facility II) 706
secondary extents, CICS load libraries 691
security
EDF 630
security considerations
for restart 553
selection field 436
send
in Example of terminal control 497
SEND command

SEND command *(continued)*
CNOTCOMPL option 268
CTLCHAR option 348
FMH option 270
INVITE option 258
LAST option 270
MSR option 435
SEND CONTROL command 346, 390
SEND MAP command
ACCUM option 406
ALARM option 389
CURSOR option 389, 393
DATAONLY option 389
ERASE option 389
ERASEAUP option 389, 415
FREEKB option 389
FROM option 389
LAST option 389
MAPONLY option 389
MAPSET option 389
NOFLUSH option 410, 415
PAGING option 389, 394
SET
command 394
SET option 389
TERMINAL option 389, 394
WAIT option 389
SEND PAGE command
AUTOPAGE option 408
NOAUTOPAGE option 408
RELEASE option 408
SEND PARTNSET command 430
SEND TEXT command
MAPPED option 417
NOEDIT option 417
sending data to a terminal 496
Sending data to a terminal
in Terminal control 496
in Using CICS Services 496
sendLine
in Example of file control 488
in Example of terminal control 497
SEQ option 582
sequence of access to resources, controlling 290
SEQUENCE option 582
sequential reading of files 487
sequential terminal support 270, 718
server
program 214
region 211, 214
Server Side Include commands 318
SESSBUSY condition 101
SET
command 11
SET CICSPLEX command 737
SET command 741, 744, 745
SET DOCTEMPLATE NEWCOPY command 311
SET option 389, 394
set...
in Sending data to a terminal 496
setColor
in Example of terminal control 497
setData
in Starting transactions 490
setHighlight
in Example of terminal control 497
setKind
in Example of file control 487
SETLOCALE 468
setQueueName
in Example of starting transactions 491
in Starting transactions 490
setReturnTermId
in Example of starting transactions 491
in Starting transactions 490
setReturnTransId
in Example of starting transactions 491
in Starting transactions 490
setRouteOption
in Example of starting transactions 491, 492
in Program control 489
Severe error handling (abendTask)
in CICS conditions 505
in Conditions, errors, and exceptions 505
shared control of records
VSAM RLS 231
shared data tables 236
SHARED option
GETMAIN command 292
SHARED option 89
shared storage 93
sharing data across transactions 104
SIGNAL condition 259
simultaneous browse 246
single-screen mode, EDF 639
single-thread testing 718
single-threading 78
singleton class 481
Singleton classes
in Creating a resource object 481
in Using CICS resources 481
size, program 96
SORT statement, COBOL 513
SOURCE option 582
SP option 582
SPACE option 582
space, presentation 428
SPIE option 582
SPOLBUSY condition 362
spool
commands 11
file 358
SPOOLCLOSE command 358
SPOOLOPEN
examples 365
SPOOLOPEN command
NODE option 361
TOKEN option 361
USERID option 361
SPOOLREAD command 358
SPOOLWRITE command 358
SQL
references 8
SQL return code
-818 724
SQLCA formatting routine 723
STAE option 566
standby procedures 553

start
in Example of starting transactions 492
in Parameter passing conventions 511
in Starting transactions 490
START command 287, 290, 353
START data, migrating to use channels 145
START statement, COBOL 513
START TRANSID command 566
STARTBR command 245
Starting transactions
in Starting transactions asynchronously 490
in Using CICS Services 490
starting transactions asynchronously 490
Starting transactions asynchronously
Accessing start data 490
Cancelling unexpired start requests 490
Example of starting transactions 490
in Using CICS Services 490
Starting transactions 490
startRequestQ
in Example of starting transactions 491, 492
startType
in Example of starting transactions 492
Static LIBRARY (DFHRPL) 676
STATIC operand 455
static storage 99
STATREG operand 455
status flag byte, route list 421
STOP statement, COBOL 513
storage
channels
lifetime 129
CICS-key 293
containers
lifetime 129
main 97
program 90
shareable 292
static 99
user 89
user-key 293
storage area, dynamic 97
storage control 292
Storage management
in Miscellaneous 511
STORAGE option 386
storage protection 292
STORAGE runtime option, Language Environment 513
storage, dynamic 460
storm drain effect 202
struct, C/370 symbolic description map set 708
stubs, EXEC interface 595, 620
subroutines 97
subspace 302
support classes 479
Support Classes
in Overview of the foundation classes 479
SUSPEND command 289
suspend data set 306
SVC99 468
symbol
defining value 321, 323
placing in document template 318
symbol list 308, 321, 323
symbol reference 308, 318
symbol substitution 308
symbol table 308
symbolic
input map 398
output map 397
symbolic cursor positioning 393
symbolic description map sets
using in a program 689
symbolic register DFHEIPLR 455
symbols in document templates 307
synchronize action
journal output 187
SYNCONRETURN option 214, 220
syncpoint
rollback 563
SYNCPOINT command
ROLLBACK option 204
syncpointing 188, 189
syncpointing, DPL 214
SYSEIB option 582
SYSID command, CEBR transaction 655
SYSID option 214
SYSIDERR 210
SYSIN 580
SYSPARM, operand for assembling map sets 708
SYSPRINT 581
SYSPUNCH 581
SYSSTATE macro 455
SYSTEM 468
system failures
designing for restart 564
system information, access to 11
system log stream
basic definition 554
system recovery table (SRT)
definition of 553

T

tables
for recovery 553
tabs, horizontal 350
tabs, vertical 350
task control
sequence of access to resources 290
task-related user exit 296
TASKDATAKEY option 89, 295
TASKDATALOC option 89, 590
TCTUA 106, 295
TCTUAKEY 107, 295
TCTUALOC 107
techniques, programming 96
temporary data 306
temporary storage
auxiliary 91
browse transaction, CEBR 650
data 306
deleting items 495
example 495
implicit enqueuing upon 571
introduction 494
main 91
migrating to channels and containers 145

temporary storage *(continued)*
reading items 494
updating items 495
Writing items 494
Temporary storage
Deleting items 495
Example of Temporary Storage 495
in Using CICS Services 494
Reading items 494
Updating items 495
Writing items 494
temporary storage queue
as document template 314
inter-transaction affinity 170
TERM option 383
TERMID value 419
terminal
contention for 258
finding out about 496
option 389
receiving data from 496
sending data to 496
sharing 665
support, sequential 270
wait 259
TERMINAL
option 394
TERMINAL command, CEBR transaction 655
terminal control
bracket protocol, LAST option 270
break protocol 259
chaining of input data 268
chaining of output data 268
definite response 269
example 497
facilities for logical units 268
finding out information 496
FMH, inbound 270
FMH, outbound 270
function management header (FMH) 269
interrupting 259
introduction 496
logical record presentation 269
map input data 398
print formatting 272
protocol, break 259
read-ahead queueing 259
receiving data 496
sending data 496
table user area (TCTUA) 106
Terminal control
commands 257
conversation partner 258
Example of terminal control 497
Finding out information about a terminal 496
flip-flop mode 258
half-duplex mode 258
in Using CICS Services 496
partner, conversation 258
Receiving data from a terminal 496
Sending data to a terminal 496
Terminal control commands 256
TERMINAL option 394
terminal paging through BMS 562
terminals, extended data stream 709
termination and initialization exit
for transaction backout 574
Test
in C++ Exceptions and the Foundation Classes 502
TEST compiler option 513
testing applications
multithread testing 718
preparing application table entries 719
preparing system table entries 719
preparing the system 719
regression testing 718
sequential terminal support 718
single-thread testing 718
using sequential devices 270, 718
testing recovery and restart programs 555
threadsafe programs
ADDRESS CWA 82
EXTRACT EXIT 82
GETMAIN SHARED 82
throw
in C++ Exceptions and the Foundation Classes 501
in Exception handling (throwException) 505
throwException
in CICS conditions 504
ti
in Example of starting transactions 491, 492
Time and date services
Example of time and date services 498
in Using CICS Services 498
time field of EIB 12
time services 498
time stamps 724
TIOATDL value 418
TITLE option 423
title, message 423
TOKEN option 250, 361
TOP command, CEBR transaction 655
tracing
activating trace output 501
TRANGRP 179
TRANISO 301
transaction
affinity 146, 210, 287, 289, 292, 299
deadlock 240
routing 209, 210
routing, dynamic 150
transaction affinity
inter-transaction affinity 157
transaction-system affinity 157
transaction backout during emergency restart
XRCINIT (initialization and termination) exit 574
transaction failure
facilities to be invoked 562
transaction groups
affinity 179
transaction identifier
CEBR 650
CECI 661
CEDF transaction 629
transaction isolation 292
transaction list table (XLT)
definition of 554
transaction restart

transaction restart *(continued)*
decision to use after DTB 564
transaction work area 89
transaction-system affinity 157
TRANSID operand
use of 561
TRANSID option 150, 214
transient data
deleting queues 493
example 493
extrapartition 93, 102
intrapartition 92
introduction 493
queue 92, 172
reading data 493
Writing data 493
Transient Data
Deleting queues 493
Example of managing transient data 493
in Using CICS Services 493
Reading data 493
Writing data 493
transient data control
automatic transaction initiation (ATI) 305
queues 304, 305
transient data queue
as document template 314
transient data queue, intrapartition 9
transient data queues
for large amounts of data 562
transient data trigger level 561
transient data, intrapartition
implicit enqueuing upon 570
translation
COBOL 525
translator
dynamic invocation of 582
integrated with Language Environment-conforming compilers 577
translator data sets 579, 590
trigger field 435
troubleshooting 735, 750
TRUNC compiler option 513
try
in C++ Exceptions and the Foundation Classes 501, 502
in Exception handling (throwException) 505
tryNumber
in C++ Exceptions and the Foundation Classes 502
tuning
CICS applications 727
TWA 89
TWASIZE option 89
type
in C++ Exceptions and the Foundation Classes 503
TYPE=DSECT assembly 376

U

ucd 751
UMT 236
UNCOMMITTED option
READ command 243
undeliverable messages 423
UNDEPLOY APPLICATION command 743
UNDEPLOY APPLICATION command sample 747
UNDEPLOY BUNDLE command 739
UNDEPLOY BUNDLE command sample 747
unescaping 308, 323
unit of work
short units of work preferred 557
UNIX
in Storage management 511
UNTIL option 288
UOW
affinity 182
update operation, BDAM 251
UPDATE option 249
updating items 495
Updating items
in Temporary storage 495
in Using CICS Services 495
updating records 249, 486
Updating records
in File control 486
in Using CICS Services 486
upgrade set 224
uppercase translation
of national characters 621
uppercase translation in BMS 400
urban code 751
urbancode 751
user
data sets 93
storage 89
user exits
emergency restart 573
transaction backout 573
user log record recovery program
exits 573
user-key storage 293
user-maintained table 236
user-replaceable module 296
USERDATAKEY option 89, 296
USERID option 361, 363
Using an object
in C++ Objects 475
using channels from JCICS 134
using CICS resources 480
Using CICS resources
Calling methods on a resource object 481
Creating a resource object 480
in Overview of the foundation classes 480
Singleton classes 481
Using CICS Services
Accessing start data 490
Browsing records 487
Cancelling unexpired start requests 490
Deleting items 495
Deleting queues 493
Deleting records 486
Example of file control 487
Example of managing transient data 493
Example of starting transactions 490
Example of Temporary Storage 495
Example of terminal control 497
Example of time and date services 498
Finding out information about a terminal 496
Reading data 493

Using CICS Services *(continued)*
Reading items 494
Reading records 485
Receiving data from a terminal 496
Sending data to a terminal 496
Starting transactions 490
Updating items 495
Updating records 486
Writing data 493
Writing items 494
Writing records 485
Using dynamic LIBRARY resources 676

V

VALIDATE 725
validation 725
validity of reference 98
variables, CECI/CECS 662
VBREF option 582
VERSION keyword 726
vertical tabs 350
viewport 428
virtual lookaside facility (VLF) 685
virtual storage 97
virtual storage environment 96
VLF (virtual lookaside facility) 685
VOLUME option 273
VOLUMELENG option 273
VS COBOL II
Language Environment 517
Language Environment callable services 534
programming 517
support 513
WORKING-STORAGE limits 513
VSAM
data sets 255
enqueues 100
MASSINSERT option 255
processor overhead 255
VSAM exclusive control 569

W

WAIT EVENT command 167, 287
WAIT EXTERNAL command 167, 289
WAIT JOURNALNUM command
synchronize with journal output 186
WAIT option 187, 259, 389
WAIT TERMINAL command 259
wait, terminal 259
WAITCICS command 167, 289
waitForAID
in Example of terminal control 497
waits, operating system 101
WITH DEBUGGING MODE 513
working set 98
Working with IccResource subclasses
in Buffer objects 483
in IccBuf class 483
WRITE command
NOSUSPEND option 252
WRITE JOURNALNAME command 101, 187
WRITE JOURNALNUM command
create a journal record 186
WRITE statement, COBOL 513
writeItem
in C++ Exceptions and the Foundation Classes 503
in Calling methods on a resource object 481
in Temporary storage 494
in Transient Data 493
in Working with IccResource subclasses 483, 484
in Writing data 493
in Writing items 495
WRITEQ TD command 304
WRITEQ TS command 306
writeRecord
in Example of file control 488
in Writing KSDS records 486
in Writing records 485
in Writing RRDS records 486
writeRecord method
IccFile class 485
Writing data
in Transient Data 493
in Using CICS Services 493
Writing ESDS records
in File control 486
in Writing records 486
Writing items
in Temporary storage 494
in Using CICS Services 494
Writing KSDS records
in File control 486
in Writing records 486
writing records 249, 252
Writing records
in File control 485
in Using CICS Services 485
Writing ESDS records 486
Writing KSDS records 486
Writing RRDS records 486
Writing RRDS records
in File control 486
in Writing records 486
WRKAREA parameter 104

X

X8 and X9 TCBs 474
XCTL
migrating to channels and containers 144
XCTL command
CHANNEL option 146
COMMAREA option 146, 149
INPUTMSG option 146, 150
XLT (transaction list table)
definition of 554
XOPTS keyword 590
XPCFTCH 474
XPCREQ global user exit 211, 214
XPCTA 474
XPLink
global user exits 474
non-XPLink objects
474
TCBs 474

XRBA (extended relative byte address) 228
XRCINIT global user exit 574
XREF option 582
XTC OUT exit 440

Z

z/OS communications Server 257