CICS Transaction Server for z/OS
5.5

Shared Data Tables Guide

Note

Before using this information and the product it supports, read the information in "Notices" on page 65.

This edition applies to the IBM® CICS® Transaction Server for z/OS® Version 5 Release 5 (product number 5655-Y04) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this PDF

This PDF gives information about CICS shared data table services.

It is intended for anyone who is involved with CICS shared data tables in one or more of the following areas:

- Planning
- Application programming
- Resource definition
- Customization
- Operations
- Problem determination

For details of the terms and notation used, see Conventions and terminology used in the CICS documentation in IBM Knowledge Center.

Date of this PDF

This PDF was created on 2024-04-22 (Year-Month-Date).

Chapter 1. Introduction to shared data tables

The CICS shared data table facility is an extension of the CICS file management services.

The concept of shared data tables

Using shared data tables, all files that are defined as data tables can potentially be shared using cross-memory services. No changes are required to the file definitions for existing data tables.

The concept of shared data tables exploits the fact that it is more efficient:

- To use MVS™ cross-memory services instead of CICS function shipping to share a file of data between two or more CICS regions in the same MVS image.
- To access data from memory instead of from DASD.
- To access a file of data from memory using services integrated within CICS file management instead of using VSAM services and a local shared resource (LSR) pool.

The two versions of data tables are:

- Shared data tables
- Coupling facility data tables (CFDTs) (see CICS coupling facility data tables for details)

Benefits of shared data tables

The use of cross-memory services is one of the major benefits of shared data tables. This improves the performance of applications that currently use function shipping and makes file sharing feasible for applications that cannot accept the performance overhead of function shipping.

The other major enhancement is that nearly all read requests are supported for use with data tables. This enhancement extends the use of data tables to applications that include:

- Browse requests
- Read requests that use an imprecise key

Besides the use of cross-memory services, shared data tables offer benefits for performance and security:

- Very large reductions in path length can be achieved for remote accesses because function shipping is avoided for most read and browse requests.
- When cross-memory services are used, the requests are processed by the AOR, thus freeing the FOR to process other requests. This increases multiprocessor exploitation.
- Increased security of data is provided because the record information in shared data tables is stored outside the CICS region and is not included in CICS system dumps (either formatted or unformatted).
- For CICS-maintained data tables, all forms of non-update, keyed access (including browse requests and imprecise-key read requests) are processed by reference to the data table.
- For user-maintained data tables, all forms of non-update, keyed access (including browse requests and imprecise-key read requests) are supported.
- Any number of files referring to the same source data set that are open at the same time can retrieve data from the one CICS-maintained data table.
- An enhancement to the XDTRD user exit allows you to skip over a range of records while loading the data table.

Description of data tables

A CICS file is a representation of a data set on DASD. If you specify that the file is to use data table services, CICS copies the contents of the data set into an MVS data space when the file is opened and

uses that copy whenever possible. Because of the way that the data table services access the records, they can be used only with a VSAM key-sequenced data set (KSDS). The KSDS is called the *source data set*. The copy in memory is called the *data table*. The process of copying the records is called *loading* the data table.

When the file is read by a CICS application, the record is normally retrieved from the data table. When the file is updated by a CICS application, the effect depends on the type of data table that you have defined for the file.

CICS data table services support two types of data table:

- CICS-maintained data table (CMT)
- User-maintained data table (UMT)

CICS-maintained data table

A CICS-maintained data table is a data table whose records are automatically reflected in the source data set. When you update the file, CICS changes both the source data set and the data table.

Treating the source data set and the data table as a single entity means that:

- Changes to the file are made to both the source data set and the data table.
- If another file is defined to use the same source data set, changes that are made by that file to the source data set are also made to the data table.
- If another file is defined to use the same source data set, records can be retrieved by that file from the data table.

A CICS-maintained data table is easy to implement—you need to know little about the data table services, you do not need to change your existing application programs, and full recovery support of the file is retained.

A data set being accessed in Record Level Sharing (RLS) mode cannot be used as the source for a CICS-maintained data table. The source data set must be accessed in non-RLS mode.

User-maintained data table

A user-maintained data table is a data table whose records are not automatically reflected in the source data set. When you update the file, CICS changes only the data table.

After a user-maintained data table has been loaded, it is independent of its source data set; the source data set is not updated when the data table is updated. A user-maintained data table lets you optimize the benefits of using a data table by allowing you to eliminate activity on the source data set, for update requests as well as read requests. Therefore, a user-maintained data table is particularly suited to applications that make frequent updates to data of a transitory nature.

A small number of file operations are not supported for user-maintained data tables. Thus, you might need to make minor changes to existing application programs. Also, recovery of the file is supported after a transaction failure, but not a system failure.

A base VSAM KSDS accessed in either non-RLS or RLS mode can be used as the source data set for a user-maintained data table. You might want to make an RLS-mode data set the source of a user-maintained data table if you have other file definitions that access the data set and the data set is updated by other CICS regions.

The data table sharing environment

The environment for sharing a data table is the same as for any file accessed in non-RLS mode.

One CICS region owns the data table—this region is known as a *file-owning region* (FOR). Any other region that uses the data table is known as an *application-owning region* (AOR). In the FOR, the file is known as a *local file* and, in the AOR, the file is known as a *remote file*.

In the context of shared data tables, the FOR is also known as a *server* and the AOR is also known as a *requester*.

The same region can be both an FOR for some data tables and an AOR for others.

For information about these intercommunication concepts, see Intercommunication methods.

Shared data tables support uses cross-region sharing wherever possible to provide access to data tables that are in the same MVS image as the requesting CICS region. This means that most read accesses within the same MVS image are satisfied by cross-region sharing using shared data tables services. If cross-region sharing is not possible for the request, function shipping is used. This means that update requests from CICS regions within the same MVS image and all requests from CICS regions in different MVS images use function shipping. Application programming for a CICS-maintained data table and Application programming for a user-maintained data table tell you when commands are satisfied by either cross-region sharing or function shipping.

Note: Similarly, XCF/MRO does not provide shared data table access between CICS regions in different MVS images.

Although shared data tables support is primarily intended for sharing data within an MVS image, the support may be extended to a sysplex environment for applications that require only read access to a shared user-maintained data table or can operate with data that might not be up-to-date. The data table must be replicated across each MVS region in the sysplex, and updated periodically. See Using shared data tables support in a sysplex.

The source data set for data tables

The source data set must be a base VSAM KSDS, not an alternate index. However, updates made to the KSDS via an alternate index are reflected in a CICS-maintained data table.

The VSAM definition of the KSDS supplies the values for maximum record length and key length.

For a user-maintained data table, the updates are not reflected in either the source data set or its alternate indexes. A user-maintained data table is entirely independent of its source data set after loading has completed.

Data spaces for data tables

The data table records are stored in one or more MVS data spaces, whether the data table is to be shared by more than one region or not. A separate set of data spaces is used for each CICS region.

The initial set of data spaces, named DFHDT001 (for table entry descriptors), DFHDT002 (for index nodes), and DFHDT003 (for up to 2 GB of record data), are obtained when the first file that is defined as a data table is opened in the region. Additional data spaces from DFHDT004 upwards may be allocated as necessary for record data, up to a maximum of 100 data spaces per region. The data spaces are used by all the CICS data tables that are owned by that region, and are retained until the CICS region is shut down.

Each data space has a maximum size of 2 GB, so the maximum amount of data space storage which a CICS region could allocate (assuming sufficient operating system resources) is 200 GB. The MVS exit IEFUSI, which can be used to control the total amount of data space that a given address space may own, can reduce the maximum size to less than this amount, and even to less than 2 GB. Within this limit, CICS allocates data space storage in units of 16 MB, and then sub-allocates this storage to the data tables in increments of 32 KB for table entry descriptors or index nodes, and increments of 128 KB for record data. If a new storage increment is needed for a data table but all the existing data space storage is already allocated to tables, CICS tries to extend the data space by 16 MB. If the data space is for record data but has already reached its maximum size of 2 GB, and the maximum number of data spaces has not been reached, then, instead of extending the existing data space, CICS creates a new data space which is treated as a logical extension of the existing set of data spaces. If CICS is unable to extend the data space for table entry descriptors or index nodes, because the data space has reached the maximum size of 2

GB, or CICS is unable to allocate any more data space storage because the total data space size set by the installation's IEFUSI exit has been reached, CICS notes that the data space is now full.

If a data space is full, any shared data table requests that need additional storage fail because of insufficient storage. For a CICS-maintained data table, this means that any future reads for the affected records (including any approximate reads near to that key) must access the VSAM data set, using function shipping if the request is not issued from the file-owning region. For a user-maintained data table, this means that the record cannot be written to the table. Developing for access to data tables has information about the response returned in this situation.

CICS does not provide a facility for viewing the current total amount of allocated data space storage. However, CICS file control statistics can give an accurate indication of the storage allocated and used for each data table. In particular, the field A17DTALD contains the amount of data space storage (in KB) that is currently allocated to the table.

The data space storage that is used by the data table is freed when the file is closed in the FOR. This storage is made available for reuse in such a way that the integrity of any AOR that was using the data table is protected.

Global user exits for data tables

Three global user exits are provided to extend the normal processing done by data table services.

- XDTRD selects the records that are copied to the data table during loading when the file is opened. For a user-maintained data table, it can also be used to modify the records.
- XDTAD selects the records that are copied to the data table when new records are added to the file.
- XDTLC performs processing at the end of the loading operation.

Shared data table services and remote file access

These diagrams illustrate the differences between using function shipping and using shared data table services to access a CICS file in another region.

Using function shipping

This diagram shows the use of function shipping to access a data set owned by another CICS region.

Figure 1. Data access using function shipping

Using shared data table services

This diagram shows how a number of AOR's can use cross-memory services to execute reads or browses, using shared data table services in an FOR in order to access the data table. (Function shipping is used for update requests and for any request that needs to access the source data set, in the same way as shown in Figure 1 on page 5.)

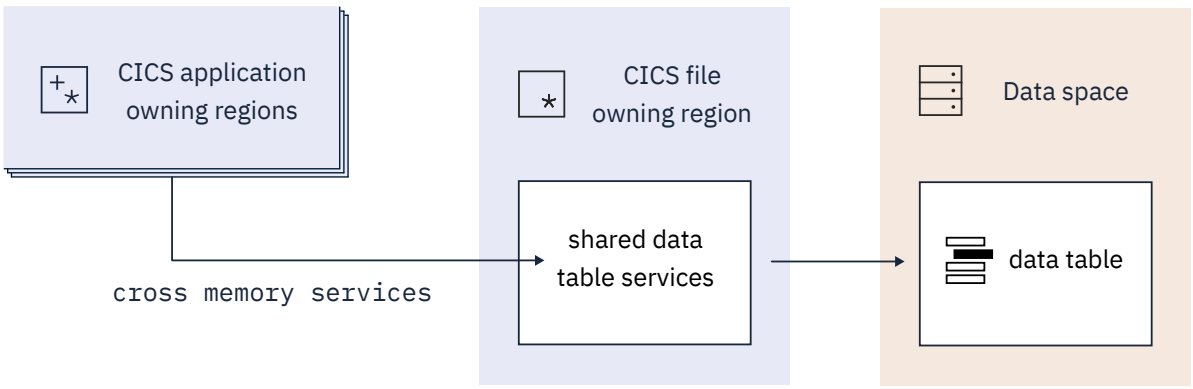

Figure 2. Data access using shared data table services

How a data table is shared

Two operations, LOGON and CONNECT, establish a data table for sharing.

LOGON

When the first file that is defined as a data table is opened in an FOR, the FOR attempts to register itself as an shared data table server. This operation is performed automatically and is known as an **SDT LOGON**. The opening of the file can be caused by the FOR or by the AOR that first accesses the file.

Regardless of whether the LOGON is successful or not, the file is opened and the data table is loaded. If the LOGON is successful, all other CICS regions in the MVS operating system are notified that the data table is available.

If the LOGON fails because of a permanent condition (such as CICS not being defined as an MVS subsystem), no further LOGON attempts are made during the CICS run.

If the LOGON fails because of a potentially transient condition, another LOGON attempt is made the next time a file defined as a data table is opened. This type of condition includes:

- Failing a security check
- Failing to obtain storage
- Failing to load a program

When a region's LOGON requests are rejected because of a security check failure, security violation messages might be issued each time a file that is defined as a data table is opened.

After an FOR logs on successfully, it remains in that state for the rest of the CICS run; no more LOGON requests are issued.

CONNECT

When an AOR with shared data table support issues a read request (or starts a browse sequence) for a remote file, CICS attempts to establish a connection to a data table for that file. This operation is performed automatically, and is known as an **SDT CONNECT**.

If the FOR is registered as a shared data table server, CICS establishes a cross-memory link from the AOR to the FOR (subject to security checks) and calls the shared data table server to ask whether there is an available data table for the file. If there is, a connection is made between the AOR and the data table.

If the CONNECT is successful, cross-memory services are used, whenever possible, to access the file while the connection exists.

If the CONNECT fails, the file request is function shipped exactly as it would have been if shared data table support were not available. The action taken for subsequent remote file requests depends on the type of failure as described below.

If the CONNECT fails because of a permanent condition (such as CICS not being defined as an MVS subsystem), no further CONNECT attempts are made during the CICS run.

If the CONNECT fails because of a potentially transient condition that is not under the control of the file owner, another CONNECT attempt is made for the next suitable request after about ten minutes have elapsed. This type of condition includes:

- Failing a security check
- Failing to obtain storage
- Failing to load a program

When a region's CONNECT requests are rejected because of a security check failure, the related security violation messages might be issued at 10-minute intervals.

If the CONNECT fails because of a potentially transient condition that is under the control of the file owner, another CONNECT attempt is made for the next suitable request following notification that at least one new file is available for shared access on the MVS system. This type of condition includes:

- File owner is not logged on as a server
- File is not associated with a data table
- File is disabled, although associated with a data table
- File is closed, although defined as a data table

After an AOR connects to a remote file successfully, it remains connected unless one of the following events occurs:

- The AOR deletes its remote file definition

  In this event, the connection is broken immediately.
- The FOR closes or disables the file

  In this event, the disconnection is scheduled at the next non-update request and is effected after all current browse sequences have terminated. See Disconnection.

If these events are later reversed, a valid connection is established in the same way as before.

Notification that a new file is available for shared access

When a data table is opened by an FOR, it becomes available for CONNECT attempts at the start of loading for a CICS-maintained data table, or at the completion of loading of a user-maintained data table. Other CICS regions are notified that a data table has become available. Notification is also made when a data table (or a file that uses a CICS-maintained data table) is enabled, having been previously disabled.

Chapter 2. Configuring shared data tables

You can configure shared data tables to share files using cross-memory services. The shared data tables facility is an extension to the CICS file management services and can improve the performance of applications that use function shipping.

Planning to use data tables

The main reason for using data tables is to take advantage of their performance benefits.

This section contains Diagnosis, Modification, or Tuning Information.

Performance of a CICS-maintained data table

If all the data and index records of a file are completely contained in an LSR pool, defining the file as a CICS-maintained data table does not reduce DASD I/O activity. There is, however, considerable potential for reduction in CPU consumption. Also, you might be able to reduce the number of buffers in the LSR pool.

If the file is not completely contained in an LSR pool, using a CICS-maintained data table could result in reductions in both DASD I/O activity and CPU consumption.

The saving of CPU consumption for a CICS-maintained data table, compared with a VSAM KSDS resident in a local shared resource (LSR) pool, depends on the application usage.

Performance of a user-maintained data table

After the loading of a user-maintained data table, DASD I/O activity is eliminated from all data table operations, so the saving of CPU consumption compared with a VSAM KSDS resident in an LSR pool is considerable.

Storage use for shared data tables

Shared data tables provide efficient use of data in memory. This means that considerable performance benefits are achieved at the cost of some additional use of storage.

This overview of the use of storage assumes that you understand the distinction between various types of storage, such as real and virtual storage, and address space and data space storage. Most of the storage used is data space storage, which is virtual storage separate from address space virtual storage.

Shared data tables use virtual storage as follows:

- Record data is stored in data spaces DFHDT003, DFHDT004, DFHDT005, and so on, with new data spaces being allocated as required. The total record data storage at loading time is basically the total size of all records (without keys, which are stored in table-entry storage) plus a small amount of control information. Data space storage is acquired in units of 16 MB, and allocated to individual tables in increments of 128 KB. Storage is then sub-allocated in page-aligned frames that are large enough to contain the maximum record length for the table. Data table frames are loosely equivalent to VSAM control intervals, and normally hold a set of records with similar keys. Where possible, each new record is stored in the same frame as the existing record with the closest lower key.

  If many records are increased in length after loading, or new records are added randomly throughout a large part of the file, the amount of storage will be increased, possibly up to twice the original size.
- Table-entry descriptor storage is allocated from data space DFHDT001. It is allocated in increments of 32 KB.

  There is one entry descriptor for each record in the table, plus one entry descriptor for each gap in the key sequence (where one or more records have been omitted from a CICS-maintained data table). The size of each entry is the keylength + 9 bytes, rounded up to the next multiple of 8 bytes.

- Index node storage is allocated from data space DFHDT002. It is allocated in increments of 32 KB.

  The size of this area depends on the distribution and format of the key values as well as the actual number of records, as indicated in Table 1 on page 12.

Table 1. Key distribution and format

Key distribution	Key format	Bytes per record
Dense (all keys are consecutive)	binary decimal alphabetic	5.1 8.5 19
Sparse (no keys are consecutive)	decimal alphabetic	44 51
Worst possible case	-	76

- ECSA storage is used for some small control blocks that need to be accessed by all regions that share data tables.

Converting a file into a shared data table could lead to an increased use of real storage, but the use of real storage for VSAM LSR buffers might be reduced if few updates are made. Also, an application that currently achieves high performance by replicating read-only tables in each CICS region might be able to make large storage savings by sharing a single copy of each table.

Once storage has been allocated to a data table, it remains allocated to that particular table until the table is closed. For example, if a data table grows to 1 GB and then all the records are deleted from the table, the table still owns 1 GB of data space storage. No other data table can use that storage until the owning data table is closed.

Free space within a data table is tracked and reused when appropriate. For example, when table entry descriptors or index nodes are no longer needed, they are added to a free chain for reuse within the same table. Similarly, when all records in a record data frame have been deleted the empty frame goes back on a free chain. When only some of the records in a frame have been deleted, the space is reused only if a new record happens to have a key which immediately follows another existing record in the same frame (or the previous frame, if there is no space in that frame). Unlike VSAM control intervals, records within a frame are not necessarily in key sequence, because they are located indirectly by means of descriptors; and records cannot be moved to consolidate free space, because this would not allow concurrent reading.

When records are allocated keys that are continuously increasing and being deleted in approximately the same sequence, space is normally reused very efficiently, because new records normally fill up a frame before going on to the next; and old frames eventually become completely empty, allowing them to be reused. This is also the case for increasing keys within multiple separate ranges, provided that the ranges are large enough for whole frames to be freed. In this situation, the amount of storage allocated to data tables is close to the amount of storage in use.

When new data table applications are introduced, it can be helpful to monitor the storage allocated and storage in use for each data table, to ensure that sufficient operating system resources are available to support current and future usage. The readings for storage allocated show the storage owned by each data table, which will not be given up until the data table is deleted. The readings for storage in use show how much of the allocated storage is in use. The CICS sample statistics program DFH0STAT provides this information. DFH0STAT is described in The sample statistics program, DFH0STAT.

It is possible that shared data tables may run out of space for any of descriptors, index entries, or data. Running out of space can occur not only at loading time but also during normal running when records are being added or even updated. Because CICS now uses multiple data spaces for supporting shared data tables, the limits for all three types of storage are greatly increased and made independent of other considerations; for example, the entries are no longer within the CICS address space. Nevertheless, the available storage is still finite. As an example, there might be extremely large numbers of relatively small records, especially if they mostly consist of the key data, in which case either the entry descriptors or the index nodes could run out before the storage for the record data itself, depending on the key length and

other factors. If there is insufficient space for entry descriptors or index nodes, consider splitting the data tables into different CICS regions; for example, different FORs. If a single data table has run out of space on its own, the limit of space for it has been reached, in which case you must consider whether it should be split into two or more separate tables.

MVS JCL requirements when using shared data tables

Before using shared data tables, you might need to change some of your JCL statements, modify your operational procedures, or increase the value of the MAXUSER MVS initialization parameter.

This is because MVS does not allow more than one step of a job to act as a shared data table server. If a second job step attempts to act as a shared data table server, CICS issues message DFHFC0405. Also, as job steps following the server step would also be unable to use cross-memory services with MRO, it is recommended that none of the job steps following the server step are another execution of CICS.

If a job that includes a shared data tables server step ends before all requester job steps that connected to this server have ended, the server address space is terminated by MVS. If the shared data table server is running under the control of a batch initiator rather than as a started task, a new initiator must be started when this situation occurs.

MVS terminates the batch initiator with the message *IEF355A INITIATOR TERMINATED, RESTART INITIATOR* because for integrity reasons MVS would otherwise have to restrict the functions that could be used by the next job that runs under that initiator, which might cause the job to fail. MVS does not allow a shared data table rserver's ASID to be re-used until after all requester job steps that connected to the server have ended.

Selecting files for use as data tables

It is not possible to lay down any exact rules about whether a file will benefit from conversion to a shared data table. The checklist in this topic gives some general guidance.

There are many considerations, and an analysis of the potential uses of shared data tables support should be undertaken by someone who understands how the files are used by the various applications and the configuration of the CICS regions.

Additional sources of information that could help you to select the files include:

- File statistics. “Using statistics to select data tables” on page 14 describes how you can use statistics information as one of the inputs to the selection task.
- The LSR pool statistics.
- Trace entries.
- Monitoring data.

However, the most beneficial input to the selection process is a thorough understanding of the applications and the way in which they use the files.

If your installation is using data tables for the first time, the following checklist gives some general principles to help you select files for defining as data tables.

- You should consider using CICS-maintained data tables first, as these are easier to implement. If you use a CICS-maintained data table, no changes are required to the applications. If you use a user-maintained data table, some changes might be required.
- Use a CICS-maintained data table if you need to ensure the integrity of the data table across a CICS restart.
- Use a CICS-maintained data table if you require journaling of updates. If you require journaling of all access requests, the file is not suitable as a data table.
- The exec interface user exits XEIIN and XEIOUT, and the file control user exits XFCREQ and XFCREQC, are not invoked in the file-owning region if a request to access a data table is satisfied by cross-memory services. When selecting a file, you should ensure that successful operation of your application does not depend on any activity performed at these user exits.

- You should be aware of the security implications of sharing a data table, as described in "Security checking for data tables" on page 17.
- If a file is frequently accessed from another region, or if it is accessed by many other regions, or if the accesses are predominantly read requests, the benefits of making it a data table can be very large. Remember that the performance gain for a remote file is greater than for a local file.
- For a CICS-maintained data table, select files that have a reasonably high proportion of requests that only access the data table (see Developing for access to data tables). From among those, select the files with the highest usage of these requests in order to maximize the performance gains.

  Information on file usage can be found in File control statistics in DFHSTUP reports. Not all read requests can take advantage of the data table, so you should check the data table information in the CICS statistics report afterward to verify that the data table is being used effectively. See Monitoring data tables for more information.
- For a user-maintained data table, select files that have a large proportion of update activity but do not require the updates to be recovered across a CICS restart (see "Data integrity" on page 21).
- Use performance measurements to estimate the approximate CPU savings, bearing in mind any forecasts for future usage.
- Select one or two files with the best estimates. Give preference to a small file over a large file when the estimated savings are similar, because a small file will probably use less real storage.
- Monitor your real storage consumption. If your system is already real-storage constrained, using a large data table could increase your page-in rates. This in turn could adversely affect CICS system performance. Use your normal performance tools, such as RMF(Version 5), to look at real-storage usage and paging rates.
- Consider reducing the number of buffers in the LSR pool because the use of data tables could reduce the number of times that the LSR pool is used.
- You can use the user exit XDTRD to select the records included in the data table. In addition, for a user-maintained data table, you can use the user exit XDTRD to modify the records. You can thus optimize the use of virtual and real storage by storing in the data table only the data that you need.
- A very large data table might require more data space storage than your usual region limit set by the MVS IEFUSI exit. In this case, you can either increase the limit by modifying the IEFUSI exit or use a CICS XDTRD global user exit program to suppress some records. The IEFUSI exit is described in the *z/OS MVS Installation Exits* manual (SA22-7593).

Using statistics to select data tables

If your sharing is confined to a single MVS image, you should consider which files have access patterns that make the use of shared data tables beneficial.

If you need to share data between more that one MVS image, you should investigate using RLS mode to share the files.

Figure 3 on page 15, Figure 4 on page 15, and Figure 5 on page 15 show some extracts from a hypothetical set of file statistics for files accessed in non-RLS mode that are used in the following discussion to demonstrate how CICS statistics can aid the selection process.

The statistics are displayed as they would be reported by the CICS offline formatting utility. Requested file statistics are shown, but Interval or End of Day statistics would be equally suitable. The section of File "Performance Information" statistics, which reports use of VSAM strings and buffers, is not shown here.

The numbers shown in the figures are purely for the purposes of illustration, and you should not expect the statistics at your installation to resemble them. Similarly, the configuration of CICS regions and files has been chosen to highlight certain points; it is not suggested that this is a typical or desirable configuration.

Monitoring data tables discusses the statistics reported for files defined as data tables, which you can use to assess the benefits being obtained.

```
Requested Statistics Report       Collection Date-Time 12/25/99-11:51:51  Last Reset 09:00:00  Applid CICFOR    Jobname SDTGSTF1
FILES - Resource Information
  File      Data Set Name                                  Data Set   RLS      DT      Time     Time     Remote    Remote   Lsrpool
  Name          Base Data Set Name (If Applicable)          Type     File  Indicator Opened   Closed     Name     Sysid      ID
  APPLE     CIC01.CICOWN.APPLES                            K       NO              07:44:12  OPEN                           1
  BANANA    CIC01.CICOWN.BANANAS                           K       NO              09:45:08  OPEN                           1
  ORANGE    CIC01.CICOWN.CITRUS                            K       NO              10:51:10  OPEN                           2
  PEAR      CIC01.CICOWN.PEARS                             K       NO              07:30:14  OPEN                           3
Requested Statistics Report       Collection Date-Time 12/25/99-11:51:51  Last Reset 09:00:00  Applid CICFOR    Jobname SDTGSTF1
FILES - Requests Information
  File     Get      Get Upd  Browse   Update   Add      Delete   Brws Upd VSAM EXCP Requests RLS req
  Name     Requests Requests Requests Requests Requests Requests Requests   Data    Index    Timeouts
  APPLE    2317265     1020        0     1019       21        1        0     11503      310           0
  BANANA    536452     1674    20344     1674      908        0        0      2651       70           0
  ORANGE   2069454    98560    17831    98327     4543     2563        0      8511      481           0
  PEAR       45871    65493     6512    65493    30109      362        0      3773      231           0
  *TOTALS*  4969042   166747    44687   166513    35581     2926        0                              0
Requested Statistics Report       Collection Date-Time 12/25/99-11:51:51  Last Reset 09:00:00  Applid CICFOR    Jobname SDTGSTF1
FILES - Data Table Requests Information
  File     Close    Read       Recs ¬  Adds from Add      Adds rejected Adds rejected Rewrite  Delete   Highest    Storage
  Name     Type     Requests   in Table   Reads   Requests    - Exit     - Table Full  Requests Requests Table Size Alloc(K)
DFHST0223 I There are no data table statistics to report.
```

Figure 3. CICFOR requested file statistics

```
Requested Statistics Report       Collection Date-Time 12/25/99-11:51:38  Last Reset 09:00:00  Applid CICAOR1   Jobname SDTGSTA1
FILES - Resource Information
  File      Data Set Name                                  Data Set   RLS      DT      Time     Time     Remote    Remote   Lsrpool
  Name          Base Data Set Name (If Applicable)          Type     File  Indicator Opened   Closed     Name     Sysid      ID
  APPLE     REMOTE                                                                  CLOSED   CLOSED   APPLE     CIF1        N
  BANANA    REMOTE                                                                  CLOSED   CLOSED   BANANA    CIF1        N
  ORANGE    REMOTE                                                                  CLOSED   CLOSED   ORANGE    CIF1        N
  ZUCCHINI  REMOTE                                                                  CLOSED   CLOSED   COURGETT  CIA2        N
Requested Statistics Report       Collection Date-Time 12/25/99-11:51:38  Last Reset 09:00:00  Applid CICAOR1   Jobname SDTGSTA1
FILES - Requests Information
  File     Get      Get Upd  Browse   Update   Add      Delete   Brws Upd VSAM EXCP Requests RLS req
  Name     Requests Requests Requests Requests Requests Requests Requests   Data    Index    Timeouts
  APPLE    1158701      532        0      531       11        1        0         0        0           0
  BANANA    305641        0    19067        0        0        0        0         0        0           0
  ORANGE     58709    32854     4265    32621     1018     1001        0         0        0           0
  ZUCCHINI   78914        0    14765        0        0        0        0         0        0           0
  *TOTALS*  1601965    33386    38097    33152     1029     1002        0                              0
Requested Statistics Report       Collection Date-Time 12/25/99-11:51:38  Last Reset 09:00:00  Applid CICAOR1   Jobname SDTGSTA1
FILES - Data Table Requests Information
  File     Close    Read       Recs ¬  Adds from Add      Adds rejected Adds rejected Rewrite  Delete   Highest    Storage
  Name     Type     Requests   in Table   Reads   Requests    - Exit     - Table Full  Requests Requests Table Size Alloc(K)
DFHST0223 I There are no data table statistics to report.
```

Figure 4. CICAOR1 requested file statistics

```
Requested Statistics Report       Collection Date-Time 12/25/99-11:49:31  Last Reset 09:00:00  Applid CICAOR2   Jobname SDTGSTA2
FILES - Resource Information
  File      Data Set Name                                  Data Set   RLS      DT      Time     Time     Remote    Remote   Lsrpool
  Name          Base Data Set Name (If Applicable)          Type     File  Indicator Opened   Closed     Name     Sysid      ID
  COURGETT  CIC02.CICOWN.COURGETT                          K       NO              08:22:15  OPEN                           1
  LEMON     REMOTE                                                 NO               CLOSED   CLOSED   ORANGE    CIF1        N
Requested Statistics Report       Collection Date-Time 12/25/99-11:49:31  Last Reset 09:00:00  Applid CICAOR2   Jobname SDTGSTA2
FILES - Requests Information
  File     Get      Get Upd  Browse   Update   Add      Delete   Brws Upd VSAM EXCP Requests RLS req
  Name     Requests Requests Requests Requests Requests Requests Requests   Data    Index    Timeouts
  COURGETT   78914    27469    14765    27469   336472        0        0      8212      481           0
  LEMON    2010745    65706    13566    65706     3525     1562        0         0        0           0
  *TOTALS*  2089659    93175    28331    93175   339997     1562        0                              0
Requested Statistics Report       Collection Date-Time 12/25/99-11:49:31  Last Reset 09:00:00  Applid CICAOR2   Jobname SDTGSTA2
FILES - Data Table Requests Information
  File     Close    Read       Recs ¬  Adds from Add      Adds rejected Adds rejected Rewrite  Delete   Highest    Storage
  Name     Type     Requests   in Table   Reads   Requests    - Exit     - Table Full  Requests Requests Table Size Alloc(K)
DFHST0223 I There are no data table statistics to report.
```

Figure 5. CICAOR2 requested file statistics

The examples use a hypothetical configuration of three CICS regions. Most of the files used by CICS applications are owned by the file-owning region CICFOR, and the applications mostly run in the application-owning regions CICAOR1 and CICAOR2. This discussion assumes that each of the data sets shown in the statistics reports is a VSAM base KSDS (as indicated by the `Data Set Type` of K), so any of them can be defined as data tables.

This section focuses on identifying candidates for defining as CICS-maintained data tables, because the decision to define a user-maintained data table is more likely to come from consideration of particular applications than from a study of file performance in general. Because of this focus, none of the statistics shown is for files accessed in RLS mode, because an RLS-mode data set cannot be the source for a CICS-maintained data table.

The statistics also show you which file names in one region are defined to access file names in another region. The *Remote Sysid* is the name given on the connection between the two regions. In the examples, the SYSID of CICFOR is CIF2 and that of CICAOR2 is CIA2.

A file with a high read-to-update ratio

The file APPLE is used by applications that run on the application-owning region CICAOR1. It is defined in CICAOR1 as a remote file, and the file definition points to the file APPLE owned by CICFOR.

This file would benefit from being redefined in CICFOR as a CICS-maintained data table because it has a high ratio of remote reads (1158701 `Get Requests` in the time period covered by the reports) to remote updates (11 adds, 1 delete and 531 updates) as seen in Figure 4 on page 15.

See File control statistics in DFHSTUP reports for guidance on the meanings of the “FILES - Requests Information” section of a statistics report.

A file with a high proportion of remote reads

The file BANANA is updated and read on CICFOR, but is also accessed by CICAOR1.

Because all the remote accesses are reads and browses, with no updates, the applications running in CICAOR1 would probably see large benefits if BANANA was defined as a data table, and the applications on CICFOR would also benefit by reading from the local data table.

A file shared by several regions

It might appear that ORANGE is not an especially suitable data table candidate.

The statistics in Figure 4 on page 15 show that the numbers of remote retrievals from CICAOR1 (58709 `Get Requests` and 4265 `Browse Requests`) are relatively low. However, the remote file LEMON in CICAOR2 also points to ORANGE in CICFOR, so defining ORANGE in CICFOR as a shared CICS-maintained data table would probably benefit the performance of the applications in both AORs.

A good UMT candidate

The file COURGETT owned by CICAOR2 is accessed via the filename ZUCCHINI in CICAOR1.

CICAOR1 only reads or browses the file; any updating is issued by the owning region. Also, it is known that these updates are relevant only to the day's CICS run and do not need to be retained permanently (in fact, they are deleted at shutdown). The file is therefore an excellent candidate for defining as a user-maintained data table. All the updates can then be made to the data table without any VSAM I/O activity, and all the remote retrievals can be made without function shipping.

A rather poor candidate

The file PEAR would probably not benefit much from shared data tables support because it is not accessed remotely and has many update and browse requests.

Local browsing does not offer as much benefit as either local reading or any form of remote retrieval, because VSAM browsing (apart from processing of the STARTBR command) is very efficient. This analysis,

of course, does not consider the relative importance of the various file accesses; the reading might be done by critical applications, but the time taken for updates might not be important.

Other possible candidates

The preceding examples illustrate only a small sample of the possible configurations and uses of files that could benefit from shared data tables support.

You could also use shared data tables support to avoid the need to duplicate files or data tables in each region. And, in addition to looking at existing files, you could consider moving files from an AOR to an FOR. Moving files from an AOR to an FOR was not practical when shared data tables support was not available, because of the cost of file accesses using function shipping.

Security checking for data tables

Shared data tables perform security checks at LOGON or CONNECT time to provide security when cross-memory services are used. You should consider the implications of the security checks before sharing a file that is associated with a data table.

Shared data tables must ensure that:

- The FOR cannot be impersonated. This is prevented by checking at LOGON time that the FOR is allowed to log on with the specified generic *applid* of the CICS region.
- An AOR cannot gain access to data that it is not supposed to see. This is prevented by checking at CONNECT time that the AOR is allowed access to the FOR and, if file security is in force, that the AOR is allowed access to the requested file.

These security checks are performed by using the system authorization facility (SAF) to call the Resource Access Control Facility (RACF®) or an equivalent security manager.

Note: A region is still able to use data tables locally even if it does not have authority to act as a shared data table server.

Shared data table support reproduces the main characteristics of function-shipping security that operate at the region level, but the following differences should be noted:

- Shared data table support does not provide any mechanism for the FOR to perform security checks at the transaction level (the equivalent of ATTACHSEC(IDENTIFY) or ATTACHSEC(VERIFY)). Therefore, if you consider that the transaction-level checks performed by the AOR are inadequate for some files, you must ensure that those files are not associated with data tables in the FOR.
- Shared data table support does not support preset security.
- Shared data table support does not pass any installation parameter list (INSTLN) information to the security user exits.

For a description of the steps required to implement shared data table security, see RACF classes for protecting system resources.

LOGON security check

LOGON processing includes a security check to verify that the FOR is authorized to act as a server with the specified application name.

This check minimizes the risk that an application-owning region (AOR) might accept counterfeit data records from a file-owning region (FOR) that is in fact an impostor. The check is never bypassed, even when SEC=NO is specified at system initialization.

CONNECT security checks

The security checks performed at CONNECT time provide two levels of security.

Bind security
Allows an FOR that runs without CICS file security to be able to restrict shared access to selected AORs. (Running without file security minimizes run-time overheads and the number of security definitions.)

File security
Can be activated in the FOR if you need a finer granularity of security checking. Shared data table support then implements those checks that apply to the AOR as a whole.

Shared data table support provides no way of implementing those security checks that an FOR makes at the transaction level when ATTACHSEC(IDENTIFY) or ATTACHSEC(VERIFY) is used with function shipping.

Preparing to use shared data tables support

To use shared data table support, you must perform the following tasks. Some of them will already have been done for an installation that currently uses function shipping or data tables.

About this task

- Either ensure that the following modules are in an authorized system library in the LNKLST of the MVS system, or move them into a library in the LPALST concatenation.
  - DFHDTSVC and DFHDTCV, because all regions using shared data tables must use the same level of SVC code.
  - DFHMVRMS, the RESMGR exit stub, because CICS JOBLIB/STEPLIB data sets are unavailable at end-of-memory.

  The following modules are placed by the installation of CICS into the target library SDFHLINK, which is normally included in the LNKLST concatenation.
  - If SDFHLINK is in the LNKLST concatenation, you should issue the operator command `MODIFY LLA,REFRESH` and wait for the confirmatory message `CSV210I LIBRARY LOOKASIDE REFRESHED` in order to make the modules available.
  - If SDFHLINK is not in the LNKLST concatenation, you should either copy the modules into a suitable library that is included and issue an LLA refresh, or copy the modules into a library in the LPALST concatenation and re-IPL the MVS system specifying CLPA.
- If any files in any AOR are to use sharing, make sure that CICS is defined as an MVS subsystem.
- Define security authorization so that FORs can act as shared data table servers and AORs can access files owned by servers, depending on the level of security required. In a single MVS image:
  - Any number of FORs can act as shared data table servers
  - A single AOR can use any number of these FORs
  - A single FOR can serve any number of AORs
  - A region can act as an AOR for one data table and as an FOR for a different data table.
- If two FORs should have the same APPLID, at any given time only one of these FORs is used as a shared data table server. However, there is nothing to prevent one FOR acting as a shared data table server and another FOR, with the same APPLID, being used for function shipped requests. You should check that your operational procedures do not allow this because there is a risk that data table requests that use shared data table services are not directed to the same region as requests that use function shipping.
- Define those files in the FOR that are data tables as either CICS-maintained data tables or user-maintained data tables.
- Create additional remote file definitions in the AOR if required. No changes are needed to existing remote file definitions.
- For any AOR that is to share data tables, specify ISC=YES as a system initialization parameter and define MRO or ISC links to the relevant FORs. For IP interconnectivity (IPIC) connections specify the equivalent system initialization parameter TCPIP=YES and define an IPIC link to the relevant FOR.

- Before using shared data tables you might need to change some of your JCL statements, modify your operational procedures, or increase the value of the MAXUSER MVS initialization parameter. For more information, see "MVS JCL requirements when using shared data tables" on page 13.

Load modules

These load modules must be installed in your CICS region in order to use shared data tables.

Table 2. Load modules used by shared data table support

Load module	Load library	How loaded	Description
DFHDTINS	SDFHLOAD	CICS load above the 16 MB line	Initialization
DFHDTSVC	SDFHLINK	MVS LOAD above the 16 MB line from link-list	Performs all functions that need MVS authorization
DFHDTFOR	SDFHAUTH	MVS LOAD above the 16 MB line	Data table FOR module
DFHDTAM	SDFHAUTH	MVS LOAD into subpool 252 storage above the 16 MB line	Data table access manager. It includes code that is executed in cross-memory mode from an AOR
DFHDTAOR	SDFHAUTH	MVS LOAD above the 16 MB line	Data table AOR module
DFHDTCV	SDFHLINK	MVS LOAD into ECSA from link-list	Connection validation (AOR)
DFHDTXS	SDFHAUTH	MVS LOAD into ECSA	Connection security checking (FOR)
DFHMVRMS	SDFHLINK	MVS LOAD above the 16 MB line from link-list	Resource manager EOT/EOM interface code

Storage occupancy

The total size of the modules that occupy storage above the 16MB line is about 41KB. For modules that are in ECSA storage, about 1.5KB are required for each logged-on FOR, and about 0.5KB for each AOR.

The modules are all eligible for inclusion in the link pack area (LPA), but only DFHDTFOR, DFHDTAM, DFHDTAOR, and possibly DFHDTCV are used sufficiently frequently to be worth considering.

Resource definition for data tables

You define a data table in the same way as a CICS file, except that you also need to specify the type of data table to be used, and the maximum number of records that can be held in the data table.

The VSAM KSDS definition supplies the maximum record length and the key length.

You can define a file as a data table by using the CEDA DEFINE FILE command, described in "The DEFINE FILE command defines data tables" on page 21.

Also, to change or check the data table attributes of an existing file you can use:

- EXEC CICS SET FILE and INQUIRE FILE commands (see "EXEC CICS commands for data tables" on page 24)
- CEMT SET FILE and INQUIRE FILE commands (see "CEMT commands for data tables" on page 25)

Resource definition for CICS-maintained data tables

Either fixed-or variable-length record format can be specified for a CICS-maintained data table.

The maximum record length that is supported by shared data table support is 32KB. This length exceeds that supported by CICS file management, which thus imposes the actual limit. See Lengths of areas passed to CICS commands. The maximum number of records that is supported is 16,777,215.

Only the base VSAM cluster can have a CICS-maintained data table based on it. Read requests through alternate index paths do not use the data table, but changes to the source data set through alternate index paths are reflected in the data table.

Note that the source data set for a CICS-maintained data table cannot be open in RLS access mode. Thus the file definition must specify RLSACCESS(NO), so any other files should be associated with the same base data set.

After a file that is defined as a CICS-maintained data table has been opened, any other non-UMT file (whether defined as a CMT or not) that names the same source data set in its definition automatically uses the same data table. If any of these other files are defined as CMTs, message DFHFC0937 is issued to the console when they are opened. This is not an error situation; the files are opened and use the existing data table whenever possible.

VSAM SHAREOPTION

If the source data set is allocated with DISP=SHR, there is a risk that it could be updated by a region other than the FOR. If this happened, the data table would no longer match the source data set. To minimize this risk, the VSAM cross-region SHAREOPTION should be set to 1 or 2.

- 1 means that either one region can have update access to the data set or many regions can have read-only access.
- 2 means that one region can have update access to the data set and, at the same time, many regions can have read-only access.

Regardless of the setting of DISP, a warning message is issued if the cross-region SHAREOPTION is 3 or 4, or if it is 2 but the CICS-maintained data table has read-only access (which means another region might be able to update the data set).

Data integrity

A file that uses a CICS-maintained data table can be defined as a recoverable resource. The source data set is recovered in the normal way after a system or transaction failure.

- After a system failure, the data table is reloaded from the recovered source data set when the file is reopened.
- After a transaction failure, changes that are made to the source data set by dynamic transaction backout are also made to the data table.

Automatic journaling is supported (in the same way as for any other file) for file operations that access the source data set. File operations that do not access the source data set are not journaled.

Resource definition for user-maintained data tables

Variable-length record format must be specified for a user-maintained data table.

The maximum record length that is supported by shared data table support is 32KB. This length exceeds that supported by CICS file management, which imposes the actual limit. See Lengths of areas passed to CICS commands. The maximum number of records supported is 16 777 215.

The source data set for a user-maintained data table can be open in RLS access mode. You might want to make an RLS-mode data set the source of a user-maintained data table if you have other file definitions that access the data set and the data set is updated by other CICS regions.

You can load multiple user-maintained data tables from the same source data set by using a separate command or macro to define each data table and making all the definitions refer to that data set.

Although a data table must be loaded from a VSAM KSDS, an application can then copy records to a user-maintained data table from any data source that is accessible from the CICS address space. This

could be an IMS or Db2® file. The KSDS that is used as the source data set for the data table can be empty; it is needed only to define the maximum record length and the key length and position.

Data integrity

A user-maintained data table can be defined as a recoverable resource. Changes to the data table are not recorded in the system log, but they are held internally in CICS memory. Thus the data table can be recovered after a transaction failure (by dynamic backout) but not after a system failure.

This is because the CICS Shared Data Table facility manages its own recovery and does not use the services of the log manager or the recovery manager. The exception is when changes are made to a recoverable data table as part of a distributed unit of work. In this case, as with other recoverable resources, a record of the link is written to the system log as part of the two-phase commit process. However, the changes themselves are not recorded in the system log.

After a system failure, the data table is reloaded from the source data set when the file is reopened. Remember that, at the time of failure, the contents of the source data set and data table would not have been the same unless you had ensured that:

- no change is made to either, or
- any change is made to both.

Automatic journaling is supported only for requests that access the source data set during loading. The records that are accessed by the loading process are journaled before user exit XDTRD, and the records that are accessed due to application requests are journaled after user exit XDTRD.

The DEFINE FILE command defines data tables

You use the **DEFINE FILE** command to define a file as either a CICS-maintained data table or a user-maintained data table.

Full details of FILE definitions are given in FILE resources. Only the attributes that relate to data tables are described in this topic.

TABLE({NO|CICS|USER|CF})
Specify **TABLE(CICS)** to define the file as a CICS-maintained data table.

Specify **TABLE(USER)** to define the file as a user-maintained data table.

If you do not specify the TABLE parameter, or specify **TABLE(NO)**, or **TABLE(CF)**, the file is not defined as a CICS shared data table.

MAXNUMRECS(NOLIMIT|number)
Specifies the maximum number of records that can be contained in the data table, in the range 1 through 99999999. The default is that there is no limit on the maximum number of records.

FILE(name)
Specifies the name of the file.

For a CICS-maintained data table, this name is used to refer to both the data table and the source data set, which are treated as a single entity by CICS.

For a user-maintained data table, this name is used to refer to only the data table.

DSNAME(name)
Specifies the name of the VSAM KSDS to be used as the source data set. This must be a base data set, not a path, or an alternate index data set. If there is a path or alternate index associated with the source data set, any updates for a CICS-maintained data table, made via the file, are reflected in both the source data set and its alternate indexes. For a user-maintained data table, the updates are not reflected in either the source data set or its alternate indexes. After loading has completed, a user-maintained data table is entirely independent of its source data set.

LSRPOOLID(number|1)
This attribute is obsolete, but is supported to provide compatibility with earlier releases of CICS.

LSRPOOLNUM(number|1|NONE)
Specifies the number of the VSAM local shared resource (LSR) pool that is to be used by the data table. You must specify an LSRPOOL number, in the range 1 through 255. The default value is 1, unless a value has been specified for the NSRGROUP attribute, in which case the default value for LSRPOOLNUM is NONE.

OPENTIME({FIRSTREF|STARTUP})
Specifies when the file is to be opened, either on first reference or immediately after startup, by the automatically initiated transaction CSFU. **OPENTIME(FIRSTREF)** is assumed by default.

Remember that the data table is loaded when the file is opened, so if you are using the user exit XDTRD, make sure that the user exit is activated before the file is opened (see Activating user exits for data tables).

RECORDFORMAT({V|F})
Specifies the format of the records in the file—either **RECORDFORMAT(V)** for variable-length records or **RECORDFORMAT(F)** for fixed-length records.

RECORDFORMAT(V) is assumed by default. A user-maintained data table must have variable-length records.

ADD(NO|YES), BROWSE(NO|YES), DELETE(NO|YES), READ(YES|NO), and UPDATE(NO|YES)
Specifies the file operations that can be requested for the data table.

RECOVERY({NONE|BACKOUTONLY|ALL})
Specifies the type of recovery support that is required for the data table. The default is **RECOVERY(NONE)**.

For a user-maintained data table, only dynamic transaction backout is supported by CICS, so **RECOVERY(BACKOUTONLY)** and **RECOVERY(ALL)** have the same meaning.

For a CICS-maintained data table, the RECOVERY parameter applies to the source data set; it must be consistent with any other file definition for the same data set.

The recovery attributes of a user-maintained data table are independent of any recovery attributes that its source data set might have.

When you define a user-maintained data table, you specify its recovery attributes on the file definition by specifying either **RECOVERY(NONE)** if it is to be unrecoverable, or **RECOVERY(BACKOUTONLY|ALL)** if it is to be recoverable after a transaction failure.

The source data set for the user-maintained data table can be unrecoverable, recoverable for backout only (after both transaction and system failures), or forward recoverable, regardless of what you have specified for the user-maintained data table.

The source data set can acquire its recovery attributes in one of two ways:

- By having the recovery attributes for the data set defined in the ICF catalog (this is possible in CICS Transaction Server for z/OS, Version 5 Release 5 for both RLS and non-RLS mode files).
- By using another file name to access the data set as an ordinary CICS file, with the recovery attributes specified in the file definition (this is only possible in CICS Transaction Server for z/OS, Version 5 Release 5 for non-RLS mode files).

Example of a CICS-maintained data table definition

This example shows the definition of a CICS-maintained data table. Only the relevant parameters are shown.

```
 File         ==> APPLE
 Group        ==> FRUIT
 DEscription  ==>
VSAM PARAMETERS
 DSNAme       ==> CIC01.CICOWN.APPLES
 Password       :                  PASSWORD NOT SPECIFIED
 RLSACCESS    ==> NO               YES|NO
 LSRPOOLId    ==> 1                1-8 | None
 LSRPOOLNum   ==> 002              1-255 | None
 READINTEG    ==> UNCOMMITTED      UNCOMMITTED|CONSISTENT|REPEATABLE
 DSNSharing   ==> Allreqs          Allreqs | Modifyreqs
 STRings      ==> 005              1 - 255
 Nsrgroup     ==>
REMOTE ATTRIBUTES
 REMOTESystem ==>
 REMOTEName   ==>
REMOTE AND CFDATATABLE PARAMETERS
 RECORDSize   ==> 00080            1-32767
 Keylength    ==> 006              1-255 (1-16 For CF Datatable)
INITIAL STATUS
 STAtus       ==> Enabled          Enabled | Disabled | Unenabled
 Opentime     ==> Startup          Firstref | Startup
 DIsposition  ==> Share            Share | Old
BUFFERS
 DAtabuffers  ==> 00002            2 - 32767
 Indexbuffers ==> 00001            1 - 32767
DATATABLE PARAMETERS
 TABLE        ==> CICS             No | Cics | User | CF
 Maxnumrecs   ==> 1000000          Nolimit  | 1-99999999
CFDATATABLE PARAMETERS
 Cfdtpool     ==>
 TABLEName    ==>
 UPDATEModel  ==> Locking          Contention | Locking
 LOad         ==> No               No | Yes
DATA FORMAT
 RECORDFormat ==> F                V | F
OPERATIONS
 Add          ==> Yes              No | Yes
 BRowse       ==> No               No | Yes
 DELete       ==> Yes              No | Yes
 REAd         ==> Yes              Yes | No
 Update       ==> Yes              No | Yes
AUTO JOURNALING
 JOurnal      ==> No               No | 1 - 99
 JNLRead      ==> None             None | Updateonly | Readonly | All
 JNLSYNCRead  ==> No               No | Yes
 JNLUpdate    ==> No               No | Yes
 JNLAdd       ==> None             None | Before | AFter |ALl
 JNLSYNCWrite ==> Yes              Yes | No
RECOVERY PARAMETERS
 RECOVery     ==> All              None | Backoutonly | All
 Fwdrecovlog  ==> 10               No | 1-99
 BAckuptype   ==> STAtic           STAtic | DYNamic
SECURITY
 RESsecnum      : 00               0-24 | Public
```

Example of a user-maintained data table definition

This example shows the definition of a user-maintained data table. Only the relevant parameters are shown.

```
 File         ==> COURGETT
 Group        ==> VEGS
 DEscription  ==>
VSAM PARAMETERS
 DSNAme       ==> CIC02.CICOWN.COURGETT
 Password       :                  PASSWORD NOT SPECIFIED
 RLSACCESS    ==> NO               YES|NO
 LSRPOOLId    ==> 1                1-8 | None
 LSRPOOLNum   ==> 002              1-255 | None
 READINTEG    ==> UNCOMMITTED      UNCOMMITTED|CONSISTENT|REPEATABLE
 DSNSharing   ==> Allreqs          Allreqs | Modifyreqs
 STRings      ==> 005              1 - 255
 Nsrgroup     ==>
REMOTE ATTRIBUTES
 REMOTESystem ==>
 REMOTEName   ==>
REMOTE AND CFDATATABLE PARAMETERS
 RECORDSize   ==> 00080            1-32767
 Keylength    ==> 006              1-255 (1-16 For CF Datatable)
INITIAL STATUS
 STAtus       ==> Enabled          Enabled | Disabled | Unenabled
 Opentime     ==> Firstref         Firstref | Startup
 DIsposition  ==> Share            Share | Old
BUFFERS
 DAtabuffers  ==> 00002            2 - 32767
 Indexbuffers ==> 00001            1 - 32767
DATATABLE PARAMETERS
 TABLE        ==> User             No | CIcs | User | CF
 Maxnumrecs   ==> 2000000          Nolimit | 1-99999999
CFDATATABLE PARAMETERS
 Cfdtpool     ==>
 TABLEName    ==>
 UPDATEModel  ==> Locking          Contention | Locking
 LOad         ==> No               No | Yes
DATA FORMAT
 RECORDFormat ==> V                V | F
OPERATIONS
 Add          ==> Yes              No | Yes
 BRowse       ==> Yes              No | Yes
 DELete       ==> No               No | Yes
 REAd         ==> Yes              Yes | No
 Update       ==> Yes              No | Yes
AUTO JOURNALING
 JOurnal      ==> No               No | 1 - 99
 JNLRead      ==> None             None | Updateonly | Readonly | All
 JNLSYNCRead  ==> No               No | Yes
 JNLUpdate    ==> No               No | Yes
 JNLAdd       ==> None             None | Before | AFter |ALl
 JNLSYNCWrite ==> Yes              Yes | No
RECOVERY PARAMETERS
 RECOVery     ==> Backoutonly      None | Backoutonly | All
 Fwdrecovlog  ==> No               No | 1-99
 BAckuptype   ==> STAtic           STAtic | DYNamic
SECURITY
 RESsecnum      : 00               0-24 | Public
```

EXEC CICS commands for data tables

You can use the **EXEC CICS SET FILE** command to change the definition of an existing file and the **EXEC CICS INQUIRE FILE** command to check the definition of an existing file.

For programming information, including details of how to use these commands and the parameters described here, see SET FILE. The parameters that are relevant to data tables are described below.

This section contains General-use Programming Interface and Associated Guidance Information.

SET FILE

The following parameters are relevant to data tables; you can use them only when the file is closed and disabled.

You can specify a data table attribute of a file in a CICS-value data area (cvda):

TABLE(cvda)
Specify a CVDA value of **CICSTABLE** to define the file as a CICS-maintained data table.

Specify a CVDA value of **USERTABLE** to define the file as a user-maintained data table.

Specify a CVDA value of **NOTTABLE** to indicate that the file is not a data table.

Note: You can also specify CFTABLE to indicate a coupling facility data table.

MAXNUMRECS(value)
Specifies the maximum number of records that can be contained in the data table, in the range 1 through 99999999. The value of zero means no limit.

INQUIRE FILE

The following parameters are relevant to data tables.

You can request that each data table attribute of a file is returned in a CICS-value data area (cvda) by specifying:

TABLE(cvda)
If the value **CICSTABLE** is returned, the file has been defined as a CICS-maintained data table.

If the value **USERTABLE** is returned, the file has been defined as a user-maintained data table.

If the value **CFTABLE** is returned, the file has been defined as a coupling facility data table.

If the value **NOTTABLE** is returned, the file is not currently defined as a data table.

If the value **NOTAPPLIC** is returned, the option is not applicable because the file is a remote file.

MAXNUMRECS(cvda)
The value returned indicates the maximum number of records that can be contained in the data table. The value of zero means no limit.

CEMT commands for data tables

You can use the CEMT SET FILE command to change the definition of an existing file, and the CEMT INQUIRE FILE command to check the definition of an existing file.

Full details of how to use these commands, including the parameters described here, are given in INQUIRE FILE. The parameters that are relevant to data tables are described below.

SET FILE

The following parameters are relevant to data tables; you can use them only when the file is closed and disabled.

{CICSTABLE|USERTABLE|CFTABLE|NOTTABLE}
specify **CICSTABLE** to define the file as a CICS-maintained data table

specify **USERTABLE** to define the file as a user-maintained data table

Note: You can also specify CFTABLE to indicate a coupling facility data table.

specify **NOTTABLE** to indicate that the file is not a data table

MAXNUMRECS(value)
Specify the maximum number of records that can be contained in the data table, in the range 1 through 99999999. The value of zero means no limit.

INQUIRE FILE

The following parameters are relevant to data tables.

Data table
If the value **CICSTABLE** is returned, the file has been defined as a CICS-maintained data table.

If the value **USERTABLE** is returned, the file has been defined as a user-maintained data table.

If the value **CFTABLE** is returned, the file has been defined as a coupling facility data table.

If the value **NOTTABLE** is returned, the file is not currently defined as a data table.

MAXNUMRECS(value)
The value returned indicates the maximum number of records that can be contained in the data table. The value of zero means that there is no maximum limit.

Chapter 3. Developing for access to data tables

You access a data table with the same EXEC CICS file control commands that you use with any normal CICS file. These commands can be used fully with a CICS-maintained data table and, with some restrictions, a user-maintained data table.

For general information about using these commands, see Using the distributed program link function. For programming information, see CICS command summary.

Application programming for a CICS-maintained data table

CICS handles a CICS-maintained data table and its source data set as a single entity. After the data table has been loaded, CICS automatically keeps the contents of the data table and the source data set consistent; any changes that an application makes to the file are reflected in both. In almost all situations, the use of a data table is transparent to the application programmer.

All file control commands and options can be used for a CICS-maintained data table. Some commands are performed by access only to the data table (using cross-memory services for shared files), some by access only to the source data set (using function shipping for shared files), and some by access to both.

The following commands usually access only the data table:

- READ commands without the UPDATE or RBA options
- STARTBR, RESETBR, READNEXT, and READPREV commands without the RBA option
- ENDBR command (unless the browse sequence has accessed the source data set)

The following commands access only the source data set:

- READ commands with the UPDATE or RBA options
- STARTBR, RESETBR, READNEXT, and READPREV commands with the RBA option
- ENDBR command for a browse sequence that has accessed the source data set

The following commands might access both the data table and the source data set:

- READ and browse commands (that would usually access only the data table) that find a gap in the key sequence of records in the data table. This gap might indicate that one or more records are missing from the data table because:
  - records have been suppressed by a user exit
  - the maximum number of records has been reached
  - insufficient virtual storage is available for the data table
  - some abnormal event has occurred
- READ, READNEXT, and READPREV commands for records that are currently being processed by a WRITE, REWRITE, or DELETE command. These commands need to first access the data table to determine that this situation exists.
- WRITE, REWRITE, and DELETE commands. These commands are always executed in the FOR, where they first update the source data set. If this is successful, a corresponding change to the data table is attempted using local shared data table services in the FOR. In the case of a WRITE command, the addition of the record to the data table might be rejected by the XDTAD user exit or might fail because the data table is full, or insufficient virtual storage is available.

Generic reads for a CICS-maintained data table

For applications that carry out generic reads, using the GENERIC option on the READ command, there is a difference in behavior for a CICS-maintained data table compared to a VSAM file. You might need to modify these applications when you convert a VSAM file to a CICS-maintained data table.

For a generic read of a VSAM file, if CICS returns a NOTFND condition because the record is not found in the table, the INTO() and RIDFLD() areas from the READ command are left unchanged. However, for a generic read of a CICS-maintained data table, if CICS returns a NOTFND condition, CICS clears the INTO() and RIDFLD() areas to ensure that an incorrect record is not returned.

This behavior optimizes performance for CICS-maintained data tables, but it means that applications can no longer depend on the original values in the INTO() and RIDFLD() areas being returned. If you have any applications that carry out generic reads, modify them as necessary to take appropriate action if a NOTFND condition is returned and the INTO() and RIDFLD() areas are cleared.

Using a CICS-maintained data table during loading

It is possible to use a CICS-maintained data table while it is being loaded. If the required record has already been loaded, processing the request is handled in the normal way.

If the record has not yet been loaded, the following is done:

- For a READ command, the record is read from the source data set and returned to the application program. It is added to the data table when the normal loading sequence reaches it.
- For a WRITE command, the record is added to the source data set and the data table (if not suppressed by the user exit XDTAD).
- For a REWRITE or DELETE command, the change is applied to the source data set. This change is then reflected in the data table by the normal loading process.

Application programming for a user-maintained data table

CICS handles a user-maintained data table and its source data set as separate entities. When loading is complete, all file control commands that access the filename are performed only on the data table.

If a request cannot be satisfied from a user-maintained data table, CICS does not access the source data set (as it would for a CICS-maintained data table). CICS returns an exception condition response instead.

You can use the user exits in data table services to put only the records that you need to access in the data table; there is no possibility of the source data set being accessed for those that you do not load. You can also use the user exit XDTRD to modify each record (by selecting, for example, only a subset of its fields) when it is loaded.

Records that were in the source data set when the data table was opened might be absent from the data table because they were not copied during loading. This could be because of suppression by the user exit XDTRD, or an abnormal event such as the data table becoming full.

Some application programming requests are not supported for a user-maintained data table. For example, read requests that use the UPDATE option with an imprecise key are not supported. Also, some exception conditions are unique to user-maintained data tables. You might need to change existing applications to comply with the restrictions on which commands and options can be used, or to handle the exception conditions that CICS returns.

The following commands are not supported; they return the INVREQ condition and a value of 44 in the EIBRESP2 field:

- Commands with the RBA option
- WRITE commands with the MASSINSERT option

The following commands are supported (using cross-memory services for remote access):

- READ commands without the RBA option or the UPDATE option. If the record does not exist in the data table, the NOTFND condition is returned.
- STARTBR, RESETBR, READNEXT, and READPREV commands without the RBA option.
- ENDBR commands.

The following commands are supported (using function shipping for remote requests):

- WRITE commands without the RBA or MASSINSERT options. The record is added to the data table (if it is not suppressed by the XDTAD user exit).

  The NOSPACE condition is returned in the following situations:

  – There is not enough data space storage to add the record to the data table.
  – The data table already contains the maximum number of records that is specified in the file definition.

  CICS issues message DFHFC0432 if a write is attempted but insufficient space is available.

  The SUPPRESSED condition is returned if the user exit XDTAD suppresses the addition of the record to the data table.
- REWRITE commands without the RBA option. The record is updated in the data table. The NOSPACE condition is returned if there is insufficient virtual storage for the updated record. CICS issues message DFHFC0432 if insufficient space is available.
- DELETE commands without the RBA option. The record is deleted from the data table. The NOTFND condition is returned if the record does not exist in the data table. The NOSPACE condition is returned if the data table is recoverable and there is insufficient virtual storage for the information that CICS writes about the deleted record.

Using a user-maintained data table during loading

A user-maintained data table can be accessed only by the FOR during loading. All remote requests are function shipped to the FOR, which processes them in the same way as for a local request.

While a user-maintained data table is being loaded, you can use only non-update read requests with precise keys. If the record has already been loaded, processing the request is handled in the normal way. If the record has not yet been loaded, the record is read from the source data set and submitted to the user exit XDTRD (if activated):

- If it is not suppressed by XDTRD, the record is added to the data table and returned to the application program.
- If it is suppressed by XDTRD, the NOTFND condition is returned.

The LOADING condition is returned for other requests that would have been valid had loading been complete.

Use of cross-memory services for shared data tables

Cross-memory services are used to satisfy an application programming command when all the conditions listed here have been met.

- CICS must retrieve the SYSID of the target system from the file's resource definition in the AOR. This condition is met when the application programming command either specifies no explicit SYSID, or specifies a SYSID the same as the AOR itself and the SYSID given in the file resource definition is the same as the FOR.

  Within a single browse sequence, an application should not change between specifying an explicit SYSID and not specifying one, as this is likely to lead to unpredictable results.
- The serving system has logged on; that is, it has registered itself as a shared data table owner.
- The requesting system has connected to the server for the files specified on the application programming command.
- The file supports the requested function.

Note: Function shipping of a request might result in “daisy chaining”; that is, the request passes through one or more intermediate CICS nodes between the region issuing the request (an AOR) and the region owning the resource (the FOR). In such cases, use of shared data tables cross-memory services is limited to the final link (from the last intermediate system to the FOR).

Connection

Commands cannot use cross-memory services until the connection is made between the AOR and the remote data table.

Also, if a browse sequence starts before the connection is made, all subsequent requests in the sequence use function shipping services. This is likely to occur if the connection cannot be established at the STARTBR command because the data table is not open, and the command causes the data table to be implicitly opened. The connection is then made on the next new request to the data table, but the original browse sequence continues to use function shipping services.

Disconnection

When a connection has been made, it remains in force until either the AOR deletes its remote file definition or the FOR closes or disables the file.

The effects of close or disable are as follows:

- If the FOR closes the file (with or without the FORCE option), disconnection is scheduled at the next non-update request that is issued for the file (that is, the next request to attempt to use cross-memory services to access the data table).

  The disconnection takes place as soon as all outstanding browse sequences (if any) against the file have terminated. Each browse sequence terminates either at the next browse request (and the transaction is abended with code AFCH unless the request is an ENDBR command) or when the transaction terminates.

  After the disconnection is scheduled, all requests (except any outstanding browse requests, as described above) are function shipped until a connection is re-established.
- If the FOR disables the file without the FORCE option, disconnection is scheduled at the next non-update READ or STARTBR command issued for the file, unless the FOR re-enables the file before then.

  If scheduled, disconnection takes place as soon as all outstanding browse sequences (if any) against the file have ended. Such browse sequences continue normally; they are unaffected by the disabling unless a browse of the source data set is started in the FOR in order to satisfy a request in the browse sequence.
- If the FOR disables the file with the FORCE option, the effect is the same as when a file is closed, except that if the FOR re-enables the file before the AOR issues the next non-update request for the file, the disabling is not observed by the AOR and disconnection is not scheduled.

Differences between function shipping and cross-memory services

There are a number of differences between the way requests are handled, depending on whether function shipping or cross-memory services are used to access the data table.

Closing a data table

When function shipping is used for a browse sequence of a remote file, the file cannot be closed (except by using the FORCE option) until after the browse sequence ends.

When cross-memory services are used, it is possible for the file to be closed during the browse sequence. In this case, the transaction is ended with abend code AFCH at the next request for that file. If your applications or operational procedures rely on the quiescing of browse activity either when closing a file or at the normal shutdown of an FOR, you should review them before using a shared data table for the file.

Disabling a data table

When function shipping is used for a browse sequence of a remote file, the browse sequence, once started, can continue normally even if the file is then disabled (unless the FORCE option is used).

When cross-memory services are used, the effect is the same unless, during the browse sequence, it is necessary to function ship a STARTBR command to the FOR. This can happen if, for example,

a gap in a CICS-maintained data table makes it necessary to browse the VSAM source data set to retrieve records. The function-shipped STARTBR command fails if the file is then disabled by a request that was issued by the FOR after the browse sequence started in the AOR. In this case the browse sequence is unable to continue normally, so the transaction in the AOR is abended with code AFCH.

If the FORCE option is used with the disable request, all function-shipped browse requests are always terminated. If the file is re-enabled, it is possible for browse requests that use cross-memory services to continue unaffected. (For information about FORCE, see "Disconnection" on page 30).

User exits

For function-shipped requests, the exec interface user exits XEIIN and XEIOUT, and the file control user exits XFCREQ and XFCREQC, are invoked in both the AOR and FOR.

For cross-memory requests, these user exits are invoked only in the AOR.

Security checking

For function-shipped requests, security checking in the FOR is invoked for the first request that refers to a given file in each unit of work. Thus transaction-level security checks can be performed in the FOR.

For cross-memory requests, security checking is invoked only at CONNECT time. Thus transaction-level security checks cannot be performed in the FOR.

Read request failure

If a read request using function shipping fails, the input area is unchanged.

If a read request using cross-memory services fails, there is a chance that the input area will be altered although no record was retrieved. You should not therefore rely on the input area being unchanged, although you can be sure that the key will not have been changed.

EXEC interface block

You might notice that read requests using cross-memory services return a value in the EIBRESP2 field. However, function-shipped requests do not, so your applications should not be dependent on this field being set by read requests.

Key length

For function-shipped requests, you must specify the correct key length in either the remote-file definition in the AOR or explicitly on the file request (to match the key length in the VSAM definition in the FOR). If you do not, the INVREQ condition is returned for any request that accesses the file. This applies to any file, not just the one that is defined as a data table.

For cross-memory requests, the key length in the AOR is not used; requests can complete successfully even if the key length is not specified in the AOR, or if the key length specified in the AOR does not match that in the FOR. However, your applications should not depend on this because some of the requests might be function shipped.

Differences between shared data tables services and VSAM

Because shared data table services replace VSAM for many data table requests, there are differences in the way that certain requests are implemented.

Read while updating (different transactions)

In the case of a READ command for a data table record following a READ UPDATE issued for that record by *another* transaction and preceding the associated update request, when shared data table services are used the READ command is processed immediately.

When VSAM is used, the READ command waits until the update request is complete.

Read while updating (same transaction)

In the case of a READ command for a data table record following a READ UPDATE issued for that record by *the same* transaction and preceding the associated update request, when shared data table services are used the READ command is processed immediately.

When VSAM is used, the transaction incurs a deadlock abend AFCG.

Delete during browse

When shared data table services are used for a STARTBR or RESETBR command for a data table record, it is possible for the record to be deleted before the associated READNEXT or READPREV command is issued. When VSAM is used, the record cannot be deleted before the associated READNEXT or READPREV command is issued.

Thus, when shared data table services are used, if a STARTBR or RESETBR command is issued with a key other than the special ‘last record' key, X'FF....', and the record selected is deleted before the READNEXT command, the READNEXT command reads the succeeding record.

If there is no succeeding record, the ENDFILE condition is returned. If the EQUAL option was used on the STARTBR or RESETBR, the key of the record that is read might not match the key specified.

If a STARTBR or RESETBR command is issued with the special ‘last record' key, and the selected record is deleted before the READPREV command, the READPREV command reads the preceding record, or returns the ENDFILE condition if there is none.

Write during browse

When shared data table services are used, if a browse reads to the end of a file, raising the ENDFILE condition, and a new record is then inserted beyond the end of the file, a subsequent READNEXT is able to read the new record.

When VSAM is used, the subsequent READNEXT may not be able to find the new record, but instead reports the ENDFILE condition again.

Delete while updating (same transaction)

When shared data table services are used for a DELETE command that specifies a RIDFLD for a data table record after a READ UPDATE has been issued for that record by the same transaction and before the associated update request, the DELETE command is processed successfully and the associated update request receives a NOTFND condition.

Chapter 4. Customizing data tables using user exits

Three global user exit points are included in data table services. You can supply one or more assembler language programs to be executed at each of these points in order to extend or modify the function provided by CICS.

Note: *This section contains Product-sensitive Programming Interface and Associated Guidance Information.*

CICS supplies sample user exit programs in the SDFHSAMP library for the XDTRD, XDTAD, and XDTLC global user exits. These programs, which are reproduced in this documentation, describe with samples of coding and data definition sequences the conventions used in user exit programs that are used with shared data tables. These samples are intended only as general guidance and do not define a programming interface.

Programming information about global user exits and how to use them is given in Global user exit programs.

Note: The EXEC interface user exits XEIIN and XEIOUT, and the file control user exits XFCREQ and XFCREQC, are not started in the file-owning region if a request to access a data table is satisfied by cross-memory services.

Communicating between CICS and shared data table exit programs

A parameter list is used to pass information between CICS and the data table exit programs.

In the CICSTS55.CICS.SDFHMAC library, CICS supplies a copybook named DFHXDTDS that contains a DSECT to define this parameter list. Include a COPY DFHXDTDS statement in each of your exit programs. The DSECT is shown in Figure 6 on page 34.

The field names used in this DSECT are referenced in the user exit descriptions that follow the figure.

```
*************************************************************************
*                                                                       *
*   Data Table Parameter List for User Exits XDTRD, XDTAD and XDTLC.    *
*                                                                       *
*   Some of the parameters are only used by one or two of the exits.    *
*   This is indicated in the comments for those parameters.             *
*   The comments also indicate whether the field is used for input      *
*   (In), output (Out), or both (In/Out).                               *
*                                                                       *
*   This definition can be used by exit programs running on CICS        *
*   regions which are at a level to support coupling facility data      *
*   tables (CFDT), providing the UEPDTCFT flag is used to test          *
*   whether the exit has been invoked from within coupling facility     *
*   data tables support, and that parameters which are specific to      *
*   CFDT support are only used when it is set.  CFDT support will       *
*   only be available to exit programs running on CICS regions at       *
*   the CICS Transaction Server version 1 release 3 level or higher.    *
*                                                                       *
*   This definition can be used by exit programs running on CICS        *
*   regions which are at a level to support shared data tables (SDT),   *
*   or which have SDT support installed, providing the UEPDTSDT flag    *
*   is used to test whether the exit has been invoked from within       *
*   shared data tables support, and that the parameters which are       *
*   specific to SDT support are only used when it is set.  SDT          *
*   support will only be available to CICS regions running at the       *
*   CICS/ESA version 4 release 1 level or higher (or running on         *
*   CICS/ESA version 3 release 3 if the Shared Data Tables feature      *
*   is installed).                                                      *
*                                                                       *
*   Careful use of these flags, and of the parameters which relate      *
*   to the various kinds of data tables support, should allow the       *
*   same user exit program to be used for more than one kind of data    *
*   table.                                                              *
*                                                                       *
*************************************************************************
DT_UE_PLIST_DSECT  DSECT ,
DT_UE_PLIST        DS  0XL84           Data Table User Exits          X
                                       Parameter List
UEPDTNAM             DS  CL8           Data table name (In)
UEPDTFLG             DS  0CL1          Flags (In):
```

Figure 6. Data table user exit parameter list

```
*-------------------------------------------------------------------*
*           The UEPDTSDT and UEPDTCFT flags indicate whether the    *
*           exit has been invoked for shared data tables or for     *
*           coupling facility data tables support.                  *
*                                                                   *
*           The UEPDTCMT and UEPDTUMT flags are available only to   *
*           exits which have been invoked by shared data tables     *
*           support.  They distinguish the two kinds of shared      *
*           data table.  Please note that on releases earlier than  *
*           CICS Transaction Server 1.3, the UEPDTUMT flag is NOT   *
*           available; on these releases, a user-maintained data    *
*           table is implied by the UEPDTCMT flag being turned off. *
*                                                                   *
*           If the exit has been invoked by CFDT support, then the  *
*           data table can only be a coupling facility data table,  *
*           so there are no extra flags to identify the kind of     *
*           data table when the exit has been invoked by coupling   *
*           facility data tables support.                           *
*                                                                   *
*           The UEPDTOPT flag is available to exits which have      *
*           been invoked by either shared data tables support or    *
*           coupling facility data tables support.                  *
*-------------------------------------------------------------------*
                      DS  BL1
UEPDTSDT              EQU X'80'       Exit invoked by SDT support
UEPDTCMT              EQU X'40'       Table is CICS-maintained
UEPDTOPT              EQU X'20'       Exit invoked by table loader,  X
                                      so optimization of the load by X
                                      skipping may be requested      X
                                      (flag is for XDTRD only)
UEPDTCFT              EQU X'10'       Exit invoked by CFDT support   X

UEPDTUMT              EQU X'08'       Table is user-maintained
*                     EQU X'07'       Reserved
```

Figure 7. Data table user exit parameter list continued

```
*-------------------------------------------------------------------*
*          The following fields are available to exits which        *
*          have been invoked by all flavors of data tables support. *
*          Not all fields are available at all of the exit points.  *
*-------------------------------------------------------------------*
UEPDTORC            DS  AL1           Data table load return code -  X
                                      XDTLC only, values below (In)
                    DS  BL2           Reserved
UEPDTRA             DS  A             Data record address - XDTRD    X
                                      and XDTAD only (In)
UEPDTRBL            DS  F             Data buffer length - XDTRD and X
                                      XDTAD only (In)
UEPDTRL             DS  F             Data table record length -     X
                                      XDTRD and XDTAD only, XDTRD    X
                                      can return new length in here  X
                                      if it amends record (only      X
                                      allowed for UMT or CFDT)       X
                                      (In/Out)
UEPDTKA             DS  A             Key address - XDTRD and XDTAD  X
                                      only (In)
UEPDTKL             DS  F             Key length - XDTRD and XDTAD   X
                                      only (In)
```

Figure 8. Data table user exit parameter list continued

```
*--------------------------------------------------------------------*
*          The following fields are available to exits which          *
*          have been invoked either by shared data tables support     *
*          or by coupling facility data tables support.               *
*          Not all fields are available at all of the exit points.    *
*--------------------------------------------------------------------*
UEPDTDSL             DS  F             Length of data set name (In)
UEPDTDSN             DS  CL44          Source data set name (In)
UEPDTSKA             DS  A             Address of skip-key area: exit X
                                       should return a key of length  X
                                       UEPDTKL in this area if it has X
                                       requested optimisation of load X
                                       by skipping - XDTRD only (In)
*--------------------------------------------------------------------*
*   Values for UEPDTORC (supplied to XDTLC exit only)                *
*--------------------------------------------------------------------*
UEPDTLCS           EQU 0               load completed successfully
UEPDTLFL           EQU 128             load failed
```

Figure 9. Data table user exit parameter list continued

The user exits should set a return code in register 15. The return code values are supplied by the DFHUEXIT macro. The valid values for each user exit are given in the following descriptions.

You can check UEPDTFLG to find out which version of data tables support invoked the exit program. For shared data tables, this flag byte also indicates which type of data table is being used and whether the exit program is being invoked during loading.

The exit program should use either the filename (field UEPDTNAM) or the name of the source data set (see fields UEPDTDSN and UEPDTDSL) to determine whether any action is to be taken for this file.

You can enable several exit programs at the same exit point, each of which, for example, takes action for a particular file or data set.

XDTRD user exit

The XDTRD user exit is invoked just before CICS attempts to add a record that has been retrieved from the source data set to the data table. You can choose whether to load the record into the data table or not. For a user-maintained data table, you can also modify the record.

XDTRD is normally invoked when the loading process retrieves a record during the sequential copying of the source data set. However, it can also be invoked when an application retrieves a record that is not in the data table and one of the following conditions applies:

- For a user-maintained data table, loading is still in progress.
- For a CICS-maintained data table, loading terminated before the end of the source data set was reached (because, for example, the data table was full).

The record retrieved from the source data set is passed as a parameter to the user exit program—see fields UEPDTRA and UEPDTRL. This program can choose (depending, for example, on the key value—see fields UEPDTKA and UEPDTKL) whether to include the record in the data table or not.

Alternatively, the exit program can request that all subsequent records up to a specified key are skipped —see field UEPDTSKA; these records are not passed to the exit program. This facility is available only during loading. You can specify the key as a complete key, or you can specify just the leading characters by padding the skip-key area with binary zeros.

The action required is indicated by setting the return code. Depending on the return code value, the following action is taken by CICS:

Table 3. Return codes for XDTRD user exit. A value of UERCPURG should be returned if the exit program has received a PURGED response to a call that it has issued.

Return code	Action
UERCDTAC	Include the record in the data table. This is the default if the exit is not activated.

Table 3. Return codes for XDTRD user exit. A value of UERCPURG should be returned if the exit program has received a PURGED response to a call that it has issued. *(continued)*	
Return code	**Action**
UERCDTRJ	Do not include the record in the data table.
UERCDTOP	Skip over this record and the following records until a key is found that is equal to or greater than the key specified in the skip-key area.

For a user-maintained data table, the program can also modify the data in the record to reduce the storage needed for the data table. Application programs that use the data table must be aware of any changes made to the record format by the exit program. If the record length is changed, the exit program must set the new length in the parameter list—see field UEPDTRL. The new length must not exceed the data buffer length—see field UEPDTRBL.

Sample XDTRD exit program: DFH$DTRD

DFH$DTRD is a sample XDTRD global user exit program. It demonstrates the use of the XDTRD user exit for shared data tables. The sample program is provided in the SDFHSAMP library.

XDTAD user exit

XDTAD is invoked for each record that is added to the source data set after initial loading. You can choose whether to add the record to the data table or not. This user exit cannot modify the records because, as the records are written by the application, it is assumed that they are already in the format used in the data table.

The XDTAD user exit is invoked when a write request is issued to a data table.

- For a user-maintained data table, the user exit is invoked once—before the record is added to the data table.
- For a CICS-maintained data table, the user exit is invoked twice—before the record is added to the source data set and then again before the record is added to the data table.

Note: For coupling facility data tables, the exit can be invoked on an open TCB; therefore, ensure that the exit is threadsafe and enabled to CICS as threadsafe to avoid excessive TCB switching.

The record written by the application is passed as a parameter to the user exit program—see fields UEPDTRA and UEPDTRL. This program can choose (depending on the key value, for example—see fields UEPDTKA and UEPDTKL) whether to include the record in the data table or not. This decision is indicated by setting the return code.

Depending on the return code value, the following action is taken by CICS:

Table 4. Return codes for XDTAD user exit. A value of UERCPURG should be returned if the exit program has received a PURGED response to a call that it has issued.	
Return code	**Action**
UERCDTAC	Add the record to the data table. This is the default if the exit is not activated.
UERCDTRJ	Do not add the record to the data table.

The XDTAD exit must not modify the data in the record. If you used XDTRD to truncate the data records when the user-maintained data table was loaded, you must code your application so that it only tries to write records of the correct format for the data table.

Sample XDTAD exit program: DFH$DTAD

DFH$DTAD is a sample XDTAD global user exit program. It demonstrates the use of the XDTAD user exit for shared data tables. The sample program is provided in the SDFHSAMP library.

XDTLC user exit

The XDTLC user exit is invoked when the loading of the data table is complete, whether successful or not. The user exit is not invoked if the data table is closed for any reason before loading is complete.

The exit program is informed if the loading did not complete successfully; see field UEPDTORC. This could occur, for example, if the maximum number of records was reached or there was insufficient virtual storage. In this case, the exit program can request that the file is closed immediately, by setting the return code.

Depending on the return code value, the following action is taken by CICS:

Table 5. Return codes for XDTLC user exit. A value of UERCPURG should be returned if the exit program has received a PURGED response to a call that it has issued.

Return code	Action
UERCDTOK	No action; the file remains open. This is the default if the exit is not activated.
UERCDTCL	Close the file.

Sample XDTLC exit program: DFH$DTLC

DFH$DTLC is a sample XDTLC global user exit program. It demonstrates the use of the XDTLC user exit for shared data tables. The sample program is provided in the SDFHSAMP library.

Activating user exits for data tables

To activate the data table user exits, complete these steps.

Procedure

1. Decide which user exits you want to use.

   For a description of each user exit, see Chapter 4, “Customizing data tables using user exits,” on page 33.
2. Write the user exit programs.

   Examples are included in Chapter 4, “Customizing data tables using user exits,” on page 33.
3. Define the user exit programs to CICS, using the CEDA DEFINE PROGRAM command, as described in PROGRAM resources.
4. Activate the user exits using the **EXEC CICS ENABLE** command.

   If required, you can later deactivate the user exits using the **EXEC CICS DISABLE** command.

Unless you control the opening of a data table explicitly, with a CEMT or EXEC CICS command, you should probably activate the user exits during CICS startup. Otherwise the loading of the data table might begin before the user exits are activated. To activate the user exits during startup:

5. Write one or more program list table post-initialization (PLTPI) programs that include the **EXEC CICS ENABLE** commands to activate the user exits.

   For programming information about PLTPI programs, see Writing initialization and shutdown programs .
6. Define a program list table (PLT) with an entry for each of those PLTPI programs, as described in Program list table (PLT).
7. Specify the **PLTPI=suffix** parameter for system initialization, as described in PLTPI system initialization parameter. Use the suffix of the PLT that was defined in the previous step. This causes the PLTPI programs to be executed in the second stage of initialization, before any files are opened.

What to do next

You can use PLT shutdown (PLTSD) programs in a similar way to disable the user exits during CICS shutdown.

Chapter 5. Administering data tables

Information about the operational aspects of data tables.

Opening a data table

A data table must be opened before it can be used by an application.

You open a data table in the same way as you would any CICS file, by one of the following methods:

- Automatically, by the CICS-supplied transaction CSFU, at the end of CICS startup, if the data table is defined with OPENTIME(STARTUP).
- Explicitly by a CEMT or EXEC CICS request issued by the user.
- Implicitly, on first reference to the data table, if the data table is defined with OPENTIME(FIRSTREF). The first remote access to a closed data table implicitly opens it.

All the rules and options for opening a CICS file also apply to a file that is defined as a data table. In addition, the loading of the data table is initiated.

For a large data table, loading could take a significant time. Developing for access to data tables discusses the application programming commands that can be used with a user-maintained data table, and the way that performance gains that can be achieved with a CICS-maintained data table are limited until loading is completed.

The following steps are done during the opening of the file:

1. The access method control block (ACB) for the VSAM source data set is opened under a separate MVS task control block (TCB). This step is the same as for any CICS file.
2. For the first data table used by a region, CICS:
   - creates MVS storage pools for use by shared data table support
   - creates an MVS data space for use by this region's data tables
   - attempts a LOGON operation as a server
3. A special CICS transaction, CFTL, is attached to load the data table into the data space.
4. The transaction that issued the request to open the data table can now continue processing.
5. CICS issues a message DFHFC0940 to indicate that loading has started. The message is sent to the CSFL transient data queue.
6. The transaction that loads the data table reads the source data set sequentially. Under the optional control of the user exit XDTRD, the transaction copies the records into data space storage.
7. CICS issues a message to indicate the result of the loading. The message number is:
   - if loading is successful: DFHFC0941
   - if loading fails: DFHFC0942, DFHFC0943, DFHFC0945, DFHFC0946, DFHFC0947, or DFHFC0948

   The message is sent to the CSFL transient data queue. Also, if loading fails, the message is sent to the console.
8. When loading is complete (whether successful or not), the user exit XDTLC is invoked if it is active. If the loading was not completed successfully, the exit program can request that the data table is closed.
9. For a user-maintained data table, the ACB for the source data set is closed when loading is complete. The data set is deallocated if it was originally dynamically allocated and becomes available to other jobs, providing there are no other ACBs still open for it.

Note: During an emergency restart, any file that requires backout action is reopened. However, if the file is defined as a data table, loading is not initiated at that time; instead it is initiated by the CSFU transaction at the end of the emergency restart. This gives an opportunity for any user exits that control the copying of records to the data table during loading to be activated at any stage of PLTPI processing.

Closing a data table

A data table is closed in the same way as for any CICS file.

Use one of the following methods:

- Explicitly, by a CEMT or EXEC CICS request issued by the user.
- Implicitly, when CICS is shutdown normally.

All the rules and options for closing a CICS file also apply to a data table. In particular:

- The rules about the quiescing of current users of the file apply (except that the file can be closed even when a transaction that is running in an AOR is in the middle of a browse sequence).
- For a user-maintained data table, if the data table is defined as recoverable, all units of work that have changed the data table must complete before the data table can be closed.

The data space storage that is used for the data table records is freed as part of the close operation. If a file is reopened after it has been closed, the processing is the same as if the file had not been previously opened.

Using shared data tables support in a sysplex

Read this information if you are currently using user-maintained data tables in a single-MVS environment, but are planning to move to a sysplex environment. It might also be helpful if you already have a sysplex, because it can show you how to exploit shared data tables support in that environment.

Overview of shared data tables support in a sysplex

A shared data table can exploit shared data tables support only within a single MVS image. However, you can extend the use of shared data tables to a sysplex environment for an application that requires only read access to a shared user-maintained data table, or for one that does not require that changes are seen immediately.

Note that a shared data table can be shared using function shipping across MVS images.

You can replicate a user-maintained data table across the sysplex, with one data table per MVS. You must have one shared data table server region in each MVS image, each owning a user-maintained data table that can be accessed using shared data table by any of the other CICS regions within that MVS. These other regions require remote file definitions that refer to the user-maintained data table in their server region. Each user-maintained data table (UMT) must have the same source data set, and this data set must be readable by all of the shared data table server regions. If the access is read-only, with the data never being updated, this will in effect provide a shared user-maintained data table in a sysplex.

If, as is probably more likely, the underlying data changes from time to time, but it is not necessary to reflect such changes immediately in the UMTs, you may periodically perform some processing to refresh the contents of the UMTs so that they are updated to match the underlying data without the need to close and then reload the UMTs. Changes are applied to the source data set, rather than to the user-maintained data table, using CICS applications that refer to the data set by a non-data table file definition, or using batch programs. An example COBOL application program, which is described in “Source code for the example program to refresh a replicated user-maintained data table” on page 46, illustrates how you can refresh the UMTs to reflect the current contents of the source data set. The program would run on each MVS image and update the UMT in that image. Such a program could be run at regular times during the day, or at user request. It would be most efficient to run it in the shared data table server regions, to avoid function shipping updates to the UMT.

If it is critical that the CICS regions in all the MVS images in the sysplex are synchronized in their view of the data, the transactions that read the data must be stopped while the refresh programs run, and restarted only after the programs have completed on all MVS systems.

This technique is appropriate only for user-maintained data tables because:

- Where read-only access is required, a user-maintained data table is the usual choice.
- It would not be possible with a CICS-maintained data table to apply updates to the source data set while leaving the tables unaffected.
- Any update made to the source data set would be reflected only in the table on the system on which the update was made.

How to refresh replicated user-maintained data tables

The following steps describe how to set up an environment to refresh replicated UMTs. In practice, you may already have some of this in place. For example, you may already have files defined as data tables. The steps described here assume that you already have a sysplex environment.

1. Select a file that is appropriate.

   As an illustration, consider an application that checks credit card numbers against a list of stolen credit cards, and requires rapid access to this list. The list is updated periodically with new batches of numbers of stolen cards. The application accesses records in a VSAM KSDS data set named PRODN.SOURCEDS using a filename UMTNAME. The application runs in a sysplex consisting of two MVS images. CICS regions CICS1A, CICS1B, and CICS1C run in the first image, and CICS2A, CICS2B and CICS2C run in the second.
2. Set up file definitions:
   - In each MVS in the sysplex, select a CICS region to be the shared data table server for this file. Within this region, define the filename by which your applications read the data as a user-maintained data table, with the data set name as that containing the source data.

     In this illustration, CICS1A and CICS2A are set up as the server regions, and files are defined to them called UMTNAME. The file definitions specify DSNAME as PRODN.SOURCEDS, TABLE as USER, and allowed operations of YES for READ, BROWSE, ADD, DELETE and UPDATE (because this file definition is used both for reading the data and for updating the UMT when it is refreshed).
   - For all the other regions in the sysplex, define the filename by which the applications read the data as a remote file with the REMOTESYSTEM as the shared data table server region in the same MVS, and the REMOTENAME as the name of the UMT in that region.

     So, in this illustration, files called UMTNAME would be defined in CICS1B and CICS1C with the REMOTESYSTEM as the sysid for CICS1A and the REMOTENAME as UMTNAME, this time with READ and BROWSE as the only allowed operations, because there is no need for the UMT to be updated through these remote definitions. Similar file definitions are set up in CICS2B and CICS2C, but for these CICS2A is the remote system.
   - In each shared data table server region, set up a file definition that can be used to read the source data set when the UMTs are refreshed.

     In this illustration, files named SOURCEDS are defined to CICS1A and CICS2A, with the DSNAME as PRODN.SOURCEDS, TABLE as NO, and allowing only READ and BROWSE operations.
   - In one region in the sysplex (which has access to the source data set), define a file that is used to apply updates to the source. The file definition could be the same as that used by the refresh program to read the source data set, but in this case it would need to allow both reading and updating operations. You might, if you prefer, decide to update the data set using a batch program, in which case this CICS file definition would not be needed.

     This illustration uses the same file definition as is used in refreshing the UMTs. In this case, one of the regions would need to define SOURCEDS as allowing all file operations.

3. Set up the source data set so that it can be accessed by all applications that need to read or update it.

   If you have DFSMS/MVSversion 1 release 3, you can access the data set from any CICS region for reading or updating by specifying RLSACCESS(Yes) in the file definitions. Note that, if you use RLS access mode, unless the data set is non-recoverable, you cannot apply the updates to it from a batch program (because only CICS can open a recoverable data set for update in RLS mode).

   If you are at an earlier release of DFSMS/MVS, you can set up the data set SHAREOPTIONS so that it can be updated by the program that applies updates to the source, and read by all others. Alternatively, you can set up the data set so that it can be updated only when it is not being read, and ensure that its opening is serialized. For the shareoptions to operate throughout the sysplex, you must use GRS (Global Resource Serialization).

   In this illustration, if RLS is not available, define PRODN.SOURCEDS either with:

   - SHR(2), so that it can be updated by the region that runs the program that applies changes to the data set and at the same time read by all the refresh programs,

   or

   - SHR(1), and normally have it open to the program that applies changes; then, when it is to be refreshed, close that access to it, and, on each server region in turn, open it, run the refresh program, and close disable it to allow the next region to open it.
4. Modify the example program so that it names your files for the UMT and the source data set, and so that the data definitions match the layout of your records. Define the program and transaction in your server regions.

   The file names in the illustration are the same as those in the program (UMTNAME and SOURCEDS). Define the program and a transaction to run it in CICS1A and CICS2A.
5. You should now be ready to start using the replicated UMTs.
6. Prime the source data set with its initial contents.
7. Open the UMTs in the shared data table server regions, to cause the contents of the source data set to be loaded into each one.
8. Start the applications in all regions in the sysplex. They will all be able to access the data using data table sharing.

   The applications running in MVS 1 will access the data through the UMT in CICS1A, and those running in MVS 2 will access it through the UMT in CICS2A.
9. When new data arrives, update the source data set.

   In this illustration, the data is updated by file SOURCEDS.
10. When you want the applications to access the new data, run the transactions in each server region that will read the source data set and the UMT, and refresh the latter to be in step with the former. Providing your applications are not invalidated if the data seen on one MVS is slightly different from that seen on another, you do not have to stop them running while you do the refresh.

Example program for refreshing a user-maintained data table

To help you write your own program, here is an example of a COBOL program that demonstrates how to refresh a UMT while it is still open, to match the source data set.

If updates are applied frequently to the source data set, and could be applied while the refresh program is running, this could mean that the source data set is never exactly reflected by the UMT, because the record being processed or records already processed could be changed. This means that the program has to be tolerant to the possibility of the records changing. The program is also written to allow for the possibility that the UMT itself is updated by other programs, although you are not recommended to operate in this way (that is, the only program that updates the UMT should be the refresh program).

How the example program operates

First, the environment is initialized. A check is made that the UMT file is local and is already open. If the UMT file is remote, the program issues a message and ends. If the UMT file is not open, the program opens it and ends (because opening the UMT will load the latest data from the source data set without the need to perform any more processing). A check is also made that the source file is local; if it is remote, the program issues a message and ends. The file that directly accesses the UMT's source data set is opened. Start browse operations are then performed on both files to allow the program to step through them both sequentially.

If the environment is set up without error, the update of the UMT starts. This involves the retrieval and comparison of pairs of records, one from the UMT and one from the base data set.

The records retrieved are compared:

- If the records are equal, the flags are set to read the next record from the UMT and the data set.
- If the UMT has a greater key than the data set, there is a record in the data set that must be added to the UMT.
- If the data set has a greater key than the UMT, there is an extra record in the UMT that must be removed.
- If the keys are equal, but the records are different, the UMT should be updated with the record in the data set.

If a record must be added to the UMT, a write operation is performed.

- If the write operation succeeds, the program goes on to process the next pair of records.
- If the write operation fails because of a record that has been inserted by another transaction between the read and the write operation performed by the program, an attempt is made to delete the record and write it again.
- If the second attempt fails, the program processes the next pair of records.
- When the next pair of records is processed, the current UMT record is compared with the next record in the data set to check for further UMT record omissions.

If a record must be deleted from the UMT, a delete operation is performed.

- If the delete operation succeeds, the program goes on to process the next pair of records.
- If the delete operation fails because the record has already been deleted between the read and delete operations, the program continues to process the next pair of records.
- When the next pair of records is processed, the current data set record is compared with the next record in the UMT to check for further records that should not be in the UMT.

If a record must be updated in the UMT, a read for update operation is performed, to get a lock on the record.

- If this is a success, the updated record is rewritten to the UMT, and the program goes on to process the next pair of records.
- If the operation fails because another transaction has deleted the record, a write operation is performed to put it back in.
- If the write operation fails, the program continues to process the next pair of records.
- When the next pair of records is processed, new records are read from both the UMT and the data set.

When the end of both files has been reached, and there are no more records left to process, the program performs end browses on both the data set and the UMT and returns. Note that the example does not close the file that directly accesses the data set. If the data set cannot operate for update in a shared environment, the file that accesses it should be set to CLOSED DISABLED to allow it to be updated.

The program traps any unexpected errors and issues an error message on the screen. Only the first operation on the UMT is checked (either the delete, write or read/rewrite operations). If that fails with a return code that could be caused by a record being changed after it was originally read, one final attempt

is made to correct the record, but this attempt is not checked. This is to prevent the program entering a loop state.

There are further comments in the code.

Setting up and executing the example program

Edit the program, according to the comments in the example, to match the format of the records being updated.

'UMTNAME' and 'SOURCEDS' should be renamed to match your file definitions.

Translate, compile and link the program using a COBOL compiler.

Define the program to CICS, and define a transaction to the program. Define the file (UMTNAME) to point to the UMT, and give it a source data set from which to load when first opened. Define the other file (SOURCEDS) to point directly to the source data set the UMT is defined to load from.

Each sysplex should have one CICS region where the UMT that is to be refreshed resides. In these regions, the definitions needed to run the refresh transaction must be installed. In all other regions in the sysplex, the UMT should be defined as a remote file, pointing to the UMT in the UMT-owning region. It is not necessary to run the refresh transaction on the regions that have the UMT defined as remote.

The update strategy used will depend on the way the source data set is set up. If the source data set is set up as RLS, all UMTs can be refreshed at the same time. Any updates to the source data set could also be applied. If the data set has the SHAREOPTIONS set so that it can be read by multiple systems at any one time, then, as with RLS, a simultaneous refresh can also occur. Otherwise, when the source data set is updated, the file that is used to read the source data set for refreshing would need to be closed and disabled on each system for the duration of the update. If all the UMTs are refreshed serially, the source data set could be opened and closed to each UMT-owning region in turn when needed for update.

Source code for the example program to refresh a replicated user-maintained data table

This source code is not provided in the CICS samples library, only in this documentation.

Example program to refresh a replicated UMT: CBL XOPTS(SP)

Program name
UMTUPDT COBOL

Descriptive name
CICS application to dynamically update a UMT with the current contents of a data set

Overview
This program demonstrates how to update a user maintained table (UMT) to match the data in the source data set it was loaded from when opened, while it remains in use by one (or more) CICS systems. It can be used to update a UMT that is replicated in different sysplexes so that they all match the source data set. It should be run on the FOR.

Requirements
This program should be translated, compiled and linked as a CICS COBOL program, and defined to CICS. A transaction name should be defined to this program. A UMT file, currently called UMTNAME, is used to access the UMT, and a source data set file, currently called SOURCEDS, is used to directly access the data set the UMT is loaded from. These definitions must be installed only in the region in which the UMT resides (the FOR). Any regions in the same sysplex that use the UMT remotely do not need to run any update process.

Description
The program will first initialize the two files that are needed, and start browsing them from the beginning. Opening the UMT will cause it to be loaded if it isn't open. If it is not open and the UMT is loaded, the operation of the program is effectively redundant and the update code will not be run. The program will also check for a remote system name. If one is present for either file, then the

program will not run. This is to prevent function shipping occurring which would obviously degrade performance.

The program will continuously read a pair of records from the two files and compare them, adding, deleting or updating any records in the UMT that don't match the source data set.

The keys of the pair of records are compared. If the key to the UMT and the key to the source data set are equal, and the records match, then no update is required. If both keys are equal, but the records are different, then the record in the source data set is used to update the UMT. If the key in the UMT is greater than the key in the source data set, then the record(s) in the source data set are written to the UMT until the keys become equal or the UMT key becomes less than the source data set key. If the UMT key is less than the source data set key, then the record(s) in the UMT are removed until the keys become equal, or the UMT key is greater than the source data set. This continues until the end of both files is reached, or an unexpected error occurs.

Any errors that are unexpected are reported to the screen, and operation of the program stops. Some errors are trapped, and a further attempt will be made to update the UMT. If this attempt fails, no further action is taken for those records, and the program will continue to process the next pair.

Modifying the program

This program may not work as is. The record structure it uses assumes that a 4 character key is used to access a 40 character record. The following changes will need to be made to allow this program to work with different record types.

The key that accesses the UMT and source data set should be changed. The variables that store the key are UMT-KEY and DS-KEY.

The length of the records are held in UMT-LEN and DS-LEN.

The UMT and source data set record variables should be changed. The variables that store these are UMT-REC (which contains UMT-REC-KEY and UMT-REC-TEXT), and DS-REC (which contains DS-REC-KEY and DS-REC-TEXT). Additional fields can obviously be added as needed.

The filename of the UMT is set as UMTNAME. This can be changed to match any UMT already defined. The source data set file is set as SOURCEDS, and can also be changed.

Source code

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. UMTUPDT.

       ENVIRONMENT DIVISION.
           EJECT.

       DATA DIVISION.

       WORKING-STORAGE SECTION.

      * Declare the UMT and DS record variables
       77 UMT-KEY          PIC X(4)  VALUE '0000'.
       77 UMT-LEN          PIC 9(2)  VALUE 40.
       01  UMT-REC.
         03  UMT-REC-KEY   PIC X(4)  VALUE SPACES.
         03  UMT-REC-TEXT  PIC X(36) VALUE SPACES.

       77 DS-KEY           PIC X(4)  VALUE '0000'.
       77 DS-LEN           PIC 9(2)  VALUE 40.
       01  DS-REC.
         03  DS-REC-KEY    PIC X(4)  VALUE SPACES.
         03  DS-REC-TEXT   PIC X(36) VALUE SPACES.

      * Declare other work variables
      * Screen output strings
       01  MESSAGE-OUTPUT  PIC X(26) VALUE 'UMT SUCCESSFULLY REFRESHED'.
       01  REMOTE-OUTPUT   PIC X(25) VALUE 'FILE RESOURCE NOT LOCAL'.
       01  ERROR-OUTPUT.
         03  ERROR-OPNAME  PIC X(8)  VALUE SPACES.
         03  FILLER        PIC X(15) VALUE ' RETURNED RESP '.
         03  ERROR-RESP    PIC X(8)  VALUE SPACES.
         03  FILLER        PIC X(7)  VALUE ' RESP2 '.
         03  ERROR-RESP2   PIC X(8)  VALUE SPACES.
         03  FILLER        PIC X(10) VALUE ' FOR FILE '.
```

```
   03  ERROR-FILE    PIC X(8)  VALUE SPACES.

* End of file flags
 77  UMT-EOF         PIC 9(1)  VALUE 0.
 77  DS-EOF          PIC 9(1)  VALUE 0.

* Record retrieval flags
 77  GET-NEXT-UMT    PIC 9(1)  VALUE 1.
 77  GET-NEXT-DS     PIC 9(1)  VALUE 1.

* File inquire variables
 77  REM-SYS-NAME    PIC X(4)  VALUE SPACES.
 77  OPEN-STAT       PIC S9(8) BINARY.

* Program operation flags
 77  PROCESS-FILES   PIC 9(1)  VALUE 1.
 77  REM-FILE        PIC 9(1)  VALUE 0.
 77  UMT-STARTBR     PIC 9(1)  VALUE 0.
 77  DS-STARTBR      PIC 9(1)  VALUE 0.
```

```
* EXEC CICS response variables
 77  RESPONSE        PIC S9(8) BINARY.
 77  RESPONSE2       PIC S9(8) BINARY.


 COPY DFHAID.
 COPY DFHBMSCA.


 LINKAGE SECTION.
     EJECT.

 PROCEDURE DIVISION USING DFHEIBLK.

*****************************************************************
* Main processing starts here.                                  *
*****************************************************************
 MAIN-PROCESSING SECTION.

* Check the UMT and data set for processing
     PERFORM FILE-CHECK.

* If the file check completed okay, process the UMT
     IF (PROCESS-FILES = 1)

* Ready the UMT and DS for access
        PERFORM INITIALIZE

* Call the update routine until the end of both files reached
        PERFORM UPDATE-UMT UNTIL (DS-EOF = 1 AND UMT-EOF = 1)

     END-IF.

* Exit the program cleanly
     PERFORM TRAN-FINISH.

 MAIN-PROCESSING-EXIT.
     GOBACK.
     EJECT


*****************************************************************
* Procedures start here.                                        *
*****************************************************************
```

```
*****************************************************************
* Check the files open status and that they aren't remote       *
*****************************************************************
 FILE-CHECK SECTION.

* Inquire on the UMT to get remote and open status information
     MOVE SPACES TO REM-SYS-NAME.
     EXEC CICS INQUIRE FILE('UMTNAME')
          OPENSTATUS(OPEN-STAT)
          REMOTESYSTEM(REM-SYS-NAME)
          RESP(RESPONSE)
          RESP2(RESPONSE2)
     END-EXEC.
* Output an error if inquire on the UMT failed
```

```
           IF (RESPONSE NOT = DFHRESP(NORMAL))
              MOVE 'INQUIRE ' TO ERROR-OPNAME
              MOVE 'UMTNAME ' TO ERROR-FILE
              PERFORM PROCESS-ERROR
           END-IF.
      * System name is not blank if the file is defined as remote
      * We don't want to do any processing if the file is remote
           IF (REM-SYS-NAME NOT = SPACES)
              MOVE 0 TO PROCESS-FILES
              MOVE 1 TO REM-FILE
           ELSE
      * If the UMT is not open, then opening it will update it
              IF (OPEN-STAT NOT = DFHVALUE(OPEN))
                 EXEC CICS SET FILE('UMTNAME')
                      OPEN
                      RESP(RESPONSE)
                      RESP2(RESPONSE2)
                 END-EXEC
      * Check open of UMT was successful
                 IF (RESPONSE NOT = DFHRESP(NORMAL))
                    MOVE 'OPEN    ' TO ERROR-OPNAME
                    MOVE 'UMTNAME ' TO ERROR-FILE
                    PERFORM PROCESS-ERROR
                 ELSE
      * Don't want to do any processing, as open will update UMT
                    MOVE 0 TO PROCESS-FILES
                 END-IF
              END-IF
           END-IF.

      * Inquire on the source data set to get remote and open status
           MOVE SPACES TO REM-SYS-NAME.
           EXEC CICS INQUIRE FILE('SOURCEDS')
                REMOTESYSTEM(REM-SYS-NAME)
                OPENSTATUS(OPEN-STAT)
                RESP(RESPONSE)
                RESP2(RESPONSE2)
           END-EXEC.
```

```
      * Output an error if inquire on the data set failed
           IF (RESPONSE NOT = DFHRESP(NORMAL))
              MOVE 'INQUIRE ' TO ERROR-OPNAME
              MOVE 'SOURCEDS' TO ERROR-FILE
              PERFORM PROCESS-ERROR
           END-IF.
      * Don't do any processing if it's a remote file
           IF (REM-SYS-NAME NOT = SPACES)
              MOVE 0 TO PROCESS-FILES
              MOVE 1 TO REM-FILE
           ELSE
      * Open the source data set
              IF (OPEN-STAT = DFHVALUE(CLOSED))
                 EXEC CICS SET FILE('SOURCEDS')
                      OPEN
                      RESP(RESPONSE)
                      RESP2(RESPONSE2)
                 END-EXEC
      * Check open of data set was successful
                 IF (RESPONSE NOT = DFHRESP(NORMAL))
                    MOVE 'OPEN    ' TO ERROR-OPNAME
                    MOVE 'SOURCEDS' TO ERROR-FILE
                    PERFORM PROCESS-ERROR
                 END-IF
              END-IF
           END-IF.

       FILE-CHECK-EXIT.
           EXIT.
           EJECT



      ****************************************************************
      * Initialize the files ready for sequential reading            *
      ****************************************************************
       INITIALIZE SECTION.

      * Start browsing the UMT from the first record
           EXEC CICS STARTBR FILE('UMTNAME')
                RIDFLD(UMT-KEY)
                GTEQ
```

```
               RESP(RESPONSE)
               RESP2(RESPONSE2)
          END-EXEC.
     * If UMT is empty (NOTFND) then treat as end of UMT and fill
          IF (RESPONSE = DFHRESP(NOTFND))
             MOVE 1 TO UMT-EOF
          ELSE
```

```
     * Output an error if the start browse for the UMT failed
             IF (RESPONSE NOT = DFHRESP(NORMAL))
                MOVE 'STARTBR ' TO ERROR-OPNAME
                MOVE 'UMTNAME ' TO ERROR-FILE
                PERFORM PROCESS-ERROR
             END-IF
          END-IF.
     * Set UMT start browse flag
          MOVE 1 TO UMT-STARTBR.

     * Start browsing the data set from the first record
          EXEC CICS STARTBR FILE('SOURCEDS')
               RIDFLD(DS-KEY)
               GTEQ
               RESP(RESPONSE)
               RESP2(RESPONSE2)
          END-EXEC.
     * If data set is empty then treat as end of data set an empty UMT
          IF (RESPONSE = DFHRESP(NOTFND))
             MOVE 1 TO DS-EOF
          ELSE
     * Output an error if the start browse for the data set failed
             IF (RESPONSE NOT = DFHRESP(NORMAL))
                MOVE 'STARTBR ' TO ERROR-OPNAME
                MOVE 'SOURCEDS' TO ERROR-FILE
                PERFORM PROCESS-ERROR
             END-IF
          END-IF.
     * Set data set start browse flag
          MOVE 1 TO DS-STARTBR.

      INITIALIZE-EXIT.
          EXIT.
          EJECT



     ****************************************************************
     * Update the UMT according to the record/key states            *
     ****************************************************************
      UPDATE-UMT SECTION.

     * Get the next records from the UMT and data set
          PERFORM READ-FILES.

     * If both records are the same, move to the next record
          IF UMT-REC = DS-REC
             MOVE 1 TO GET-NEXT-UMT
             MOVE 1 TO GET-NEXT-DS
          ELSE
```

```
     * If UMT is behind data set then extra record in UMT so delete it.
     * Also delete records from UMT if EOF DS reached before EOF UMT
             IF (UMT-EOF = 0 AND (UMT-KEY < DS-KEY OR DS-EOF = 1))
                PERFORM UMT-DELETE
             END-IF

     * If UMT ahead of data set then extra record in DS so add to UMT
     * Also add records to the UMT if the EOF reached before EOF DS
             IF (DS-EOF = 0 AND (UMT-KEY > DS-KEY OR UMT-EOF = 1))
                PERFORM UMT-WRITE
             END-IF

     * If both keys equal but record different, update UMT
             IF ((DS-EOF = 0 AND UMT-EOF = 0) AND UMT-KEY = DS-KEY)
                PERFORM UMT-UPDATE
             END-IF

          END-IF.
```

```
 UPDATE-UMT-EXIT.
     EXIT.
     EJECT


*****************************************************************
* Read the next record from both files                          *
*****************************************************************
 READ-FILES SECTION.

* If the flags are set to read the next UMT record, do so
     IF (GET-NEXT-UMT = 1 AND UMT-EOF = 0)
        MOVE SPACES TO UMT-REC
        EXEC CICS READNEXT FILE('UMTNAME')
             RIDFLD(UMT-KEY)
             INTO(UMT-REC)
             RESP(RESPONSE)
             RESP2(RESPONSE2)
        END-EXEC
* Set the EOF flag if the end of the UMT has been reached
        IF (RESPONSE = DFHRESP(ENDFILE))
              MOVE 1 TO UMT-EOF
        ELSE
* Output an error if the return code from the READ is unexpected
           IF (RESPONSE NOT = DFHRESP(DUPKEY) AND
              RESPONSE NOT = DFHRESP(NORMAL))
                 MOVE 'READNEXT' TO ERROR-OPNAME
                 MOVE 'UMTNAME ' TO ERROR-FILE
                 PERFORM PROCESS-ERROR
           END-IF
        END-IF
     END-IF.
```

```
* If the flags are set to read the next data set record, do so
     IF (GET-NEXT-DS = 1 AND DS-EOF = 0)
        MOVE SPACES TO DS-REC
        EXEC CICS READNEXT FILE('SOURCEDS')
             RIDFLD(DS-KEY)
             INTO(DS-REC)
             RESP(RESPONSE)
             RESP2(RESPONSE2)
        END-EXEC
* Set the EOF flag if the end of the data set has been reached
        IF (RESPONSE = DFHRESP(ENDFILE))
              MOVE 1 TO DS-EOF
        ELSE
* Output an error if the return code from the READ is unexpected
           IF (RESPONSE NOT = DFHRESP(DUPKEY) AND
              RESPONSE NOT = DFHRESP(NORMAL))
                 MOVE 'READNEXT' TO ERROR-OPNAME
                 MOVE 'SOURCEDS' TO ERROR-FILE
                 PERFORM PROCESS-ERROR
           END-IF
        END-IF
     END-IF.

 READ-FILES-EXIT.
     EXIT.
     EJECT

*****************************************************************
* Attempt to delete a record from the UMT                       *
*****************************************************************
 UMT-DELETE SECTION.

* Delete the last read record in the UMT
     EXEC CICS DELETE FILE('UMTNAME')
          RIDFLD(UMT-KEY)
          RESP(RESPONSE)
          RESP2(RESPONSE2)
     END-EXEC.
* Allow NORMAL and NOTFND return codes in case record has been
* deleted since it was first read, otherwise output an error
     IF (RESPONSE = DFHRESP(NORMAL) OR
       RESPONSE = DFHRESP(NOTFND))
* Set flags to get next UMT record, but keep same data set record
        MOVE 1 TO GET-NEXT-UMT
        MOVE 0 TO GET-NEXT-DS
     ELSE
        MOVE 'DELETE  ' TO ERROR-OPNAME
```

```
           MOVE 'UMTNAME ' TO ERROR-FILE
           PERFORM PROCESS-ERROR
        END-IF.

 UMT-DELETE-EXIT.
        EXIT.
        EJECT
```

```
      ****************************************************************
      * Attempt to write a record to the UMT                         *
      ****************************************************************
       UMT-WRITE SECTION.

      * Attempt to write the missing record using the data set key
           EXEC CICS WRITE FILE('UMTNAME')
                RIDFLD(DS-KEY)
                FROM(DS-REC)
                RESP(RESPONSE)
                RESP2(RESPONSE2)
           END-EXEC.
      * If the UMT has had a record written to this position since the
      * read then delete it and try one last time.
      * If write still unsuccessful, move to the next pair of records
           IF RESPONSE = DFHRESP(DUPREC)
              EXEC CICS DELETE FILE('UMTNAME')
                   RIDFLD(DS-KEY)
                   RESP(RESPONSE)
                   RESP2(RESPONSE2)
              END-EXEC
              EXEC CICS WRITE FILE('UMTNAME')
                   RIDFLD(DS-KEY)
                   FROM(DS-REC)
                   RESP(RESPONSE)
                   RESP2(RESPONSE2)
              END-EXEC
           ELSE
      * Output an error if return code from first write was bad
      * (but allow suppression return code by user exit)
              IF (RESPONSE NOT = DFHRESP(NORMAL) AND
                 RESPONSE NOT = DFHRESP(SUPPRESSED))
                    MOVE 'UMTNAME ' TO ERROR-FILE
                    MOVE 'WRITE   ' TO ERROR-OPNAME
                    PERFORM PROCESS-ERROR
              END-IF
           END-IF.

      * Set flags to keep same UMT record, and get next data set record
           MOVE 0 TO GET-NEXT-UMT.
           MOVE 1 TO GET-NEXT-DS.

       UMT-WRITE-EXIT.
           EXIT.
           EJECT
```

```
      ****************************************************************
      * Attempt to update a record in the UMT to match the DS        *
      ****************************************************************
       UMT-UPDATE SECTION.

      * Attempt to get a lock on the record using read for update
           EXEC CICS READ FILE('UMTNAME')
                RIDFLD(UMT-KEY)
                INTO(UMT-REC)
                UPDATE
                RESP(RESPONSE)
                RESP2(RESPONSE2)
           END-EXEC.
      * If record has been deleted since original read, write it.
      * If write is unsuccessful, move to next pair of records
           IF RESPONSE = DFHRESP(NOTFND)
              EXEC CICS WRITE FILE('UMTNAME')
                   RIDFLD(UMT-KEY)
                   FROM(DS-REC)
                   RESP(RESPONSE)
                   RESP2(RESPONSE2)
              END-EXEC
           ELSE
      * If read for update was successful, write data set record to UMT
```

```
           IF RESPONSE = DFHRESP(NORMAL)
              EXEC CICS REWRITE FILE('UMTNAME')
                   FROM(DS-REC)
                   RESP(RESPONSE)
                   RESP2(RESPONSE2)
              END-EXEC
      * Output an error if rewrite failed
              IF RESPONSE NOT = DFHRESP(NORMAL)
                 MOVE 'REWRITE ' TO ERROR-OPNAME
                 MOVE 'UMTNAME ' TO ERROR-FILE
                 PERFORM PROCESS-ERROR
              END-IF
           ELSE
      * Output an error if the read for update failed
              MOVE 'READUPDT' TO ERROR-OPNAME
              MOVE 'UMTNAME ' TO ERROR-FILE
              PERFORM PROCESS-ERROR
           END-IF
        END-IF.

      * Set flags to get next record for both UMT and data set
        MOVE 1 TO GET-NEXT-UMT.
        MOVE 1 TO GET-NEXT-DS.

       UMT-UPDATE-EXIT.
        EXIT.
        EJECT
```

```
      ****************************************************************
      * Exit from the program cleanly                                *
      ****************************************************************
       TRAN-FINISH SECTION.

      * End the browse operation for the UMT
        IF (UMT-STARTBR = 1)
           EXEC CICS ENDBR FILE('UMTNAME')
                RESP(RESPONSE)
                RESP2(RESPONSE2)
           END-EXEC
        END-IF.

      * End the browse operation for the data set
        IF (DS-STARTBR = 1)
           EXEC CICS ENDBR FILE('SOURCEDS')
                RESP(RESPONSE)
                RESP2(RESPONSE2)
           END-EXEC
        END-IF

      * Output a message to the screen if UMT was updated
        IF (REM-FILE = 0)
           EXEC CICS SEND TEXT
                FROM(MESSAGE-OUTPUT)
                ERASE
                RESP(RESPONSE)
                RESP2(RESPONSE2)
           END-EXEC
        ELSE
      * Output a message if either file was defined as remote
           EXEC CICS SEND TEXT
                FROM(REMOTE-OUTPUT)
                ERASE
                RESP(RESPONSE)
                RESP2(RESPONSE2)
           END-EXEC
        END-IF.


      * End the program and return to CICS
        EXEC CICS RETURN
        END-EXEC.

       TRAN-FINISH-EXIT.
        EXIT.
        EJECT
```

```
      ****************************************************************
      * Display error message on screen and exit program             *
      ****************************************************************
       PROCESS-ERROR SECTION.

      * Copy last return codes into the message
           MOVE RESPONSE TO ERROR-RESP.
           MOVE RESPONSE2 TO ERROR-RESP2.

      * Output message to the screen
           EXEC CICS SEND TEXT
                FROM(ERROR-OUTPUT)
                ERASE
                RESP(RESPONSE)
                RESP2(RESPONSE2)
           END-EXEC.

      * End the program and return to CICS
           EXEC CICS RETURN
           END-EXEC.

       PROCESS-ERROR-EXIT.
           EXIT.
```

Chapter 6. Troubleshooting data tables

Use the trace and dump information that is produced by CICS to help you determine the cause of a problem with shared data tables.

Explanations of the diagnostic messages and abend codes produced by shared data tables are contained in CICS messages.

Trace information for data table services

The trace table produced by CICS helps you determine the cause of a problem. It shows the flow of control through the CICS modules. The entries described here are included in the trace table by data table services.

For information on the contents of the trace table and how to obtain it, see Using CICS trace.

There are two types of trace points:

- Entry and exit trace points for each of the services provided by shared data table support. File control level-2 tracing must be enabled to obtain these trace points.
- Exception trace points.

Entry and exit trace points for shared data tables

These entry and exit trace points are provided by shared data table services.

0B13
Entry to Remote Read service

0B14
Exit from Remote Read service

0B1B
Entry to Initialize Data Table Support service

0B1C
Exit from Initialize Data Table Support service

0B1D
Entry to Logon service

0B1E
Exit from Logon service

0B1F
Entry to Load service

0B20
Exit from Load service

0B21
Entry to Open, Close, Set Enablement and Statistics services

0B22
Exit from Open, Close, Set Enablement and Statistics services

0B23
Entry to local read services

0B24
Exit from local read services

0B25
Entry to update (add record, add, replace, delete) services

0B26
Exit from update services

0B2D
Entry to Connect and Disconnect services

0B2E
Exit from Connect and Disconnect services

The format of each of these trace points is described in Using CICS trace.

Function and qualifier flags for shared data tables

For shared data tables, each entry and exit trace point contains a function field, and most of them contain a qualifier flags field. The function field is a byte that identifies the function that was being performed; the qualifier flags field is a byte that contains flags that qualify some of the functions.

The values of these fields are:

Table 6. Function and qualifier flags and values

Function	Qualifier flags
X'00' Initialize	X'00' as shared data table server X'80' as shared data table requester
X'02' Add entry from source	X'00' add issued as a result of a data set to table read request X'40' add issued by load transaction
X'03' Write entry to table	X'00' completed write X'80' pre-write for CMT
X'04' Rewrite entry in table	X'00' completed rewrite X'80' pre-rewrite for CMT
X'05' Delete entry in table	X'00' completed delete X'80' pre-delete for CMT
X'06' Commit user-maintained data table updates made by this unit of work	
X'07' Roll back user-maintained data table updates made by this unit of work	
X'08' Load data table (on exit trace only)	X'00' load OK X'80' source file is empty
X'09' Point at a record	X'80' equal match X'40' greater than match X'20' less than match (the above can be in various combinations) X'10' test if data table is enabled

Table 6. Function and qualifier flags and values (continued)	
Function	**Qualifier flags**
X'0A' Retrieve record by key	X'80' equal match X'40' greater than match X'20' less than match (the above can be in various combinations) X'10' test if data table is enabled
X'0B' Retrieve record by token	X'80' equal match (internal fastpath for a sequence of records) X'40' greater than match X'20' less than match (the above can be in various combinations) X'10' test if data table is enabled
X'0C' Logon as a server	
X'0E' Open a data table	
X'0F' Close a data table	
X'10' Collect statistics	
X'11' Set enablement state	X'00' enable data table X'80' disable data table X'40' force disablement (always combined with disable)
X'15' Connect to a shared data table	
X'16' Break connection to a shared data table	
X'17' Process the completion of loading	

Response codes for shared data tables

Each exit trace point for shared data tables contains a two-byte response-code and reason-code field.

The first byte is the response code, for which the possible values are:

X'01'
Successful

X'02'
Exception

X'03'
Disaster

X'04'
Invalid

X'06'
Purged

Reason codes for shared data tables

Each exit trace point for shared data tables contains a two-byte response-code and reason-code field.

The second byte is the reason code, for which the possible values are given below. This reason code might have accompanying error code information. The error code is a four-byte field that is also reported in either an error message or an exception trace point. The possible values are described in CICS messages, “Analyzing errors from the data tables SVC” on page 60, and “Analyzing errors from data tables cross-memory services” on page 63.

X'01'
Record not in data table

X'02'
Duplicate (record already in data table)

X'03'
Data table full (already contains the maximum number of records)

X'04'
Record rejected by user exit

X'05'
Failed to get storage

X'06'
Record not in data table (and table is known to be complete)

X'07'
Data table service failed

X'08'
Not authorized to connect to file

X'09'
Resource is not a data table

X'0A'
Remote system has not logged on as a server

X'0B'
Load request failed

X'0C'
Data table is disabled

X'0D'
Add request (from DASD) deliberately not processed

X'0E'
Record too long

X'0F'
Data table token invalid

X'10'
Record not in data table (but might be in source data set)

X'11'
Data table not closed as other files are still using it

X'12'
Reserved

X'13'
Record is in data table but not currently valid

X'14'
File cannot be closed as it is disabled

X'15'
Protocol error

X'16'
CICS is not an MVS subsystem

X'17'
Not authorized to connect to this file

X'18'
CICS cannot use cross-memory services

X'19'
Interface parameter block format not recognized

UMT and other flags for shared data tables

This flag byte is included in the entry trace point on OPEN.

The significant bits at open time are:

B'1.......'
CICS-maintained data table

B'01......'
Recoverable user-maintained data table

B'00......'
Nonrecoverable user-maintained data table

Exception trace points for shared data tables

These exception trace points are provided by shared data table services.

AP 0B0A
Unrecognized function on call to DFHDTRE

AP 0B0B
Unrecognized function on call to DFHDTRR

AP 0B0C
Unrecognized function on call to DFHDTUP

AP 0B0D
Unrecognized function on call to DFHDTST

AP 0B0E
Unrecognized function on call to DFHDTSS

AP 0B0F
Unrecognized function on call to DFHDTRC

AP 0B10
Error on initializing record management

AP 0B11
Error on record manager OPEN

AP 0B12
Error on record manager CLOSE

AP 0B15
Unexpected error on call to retrieval PC

AP 0B19
Error calling data tables SVC when initializing as server

AP 0B1A
Error calling data tables SVC when initializing as requestor

AP 0B27
CLOSE could not find table block

AP 0B28
CLOSE could not find file block

AP 0B29
Error calling data tables SVC when logging on as server

AP 0B2A
Error calling data tables SVC when connecting or disconnecting

AP 0B2B
XDTRD exit returned invalid record length (that is, it changed the length for a CMT, or increased the length for a UMT)

AP 0B2C
Connect index exceeds maximum supported size

AP 0B2F
Disastrous error when acquiring storage to pass parameters to loading transaction

The format of each of these trace points is described in Using CICS trace.

Analyzing errors from the data tables SVC

Following an error from a call to the data tables SVC, an exception trace point is always made, including an error code field to identify the reason for the error. These trace points are AP 0B12, 0B19, 0B1A, 0B29 and 0B2A.

There are three categories of SVC error:

1. Conditions that are expected to occur, such as the remote file on a connect attempt not being a data table, or the remote system not having logged on as a shared data tables server. CICS takes the appropriate action for such conditions, and no diagnostic information is needed.
2. Errors that could be caused by problems in the environment that might be possible to correct. For these errors, a message is issued with the reason code for the error. The explanation of the reason code is included in the explanation of the message in CICS messages.
3. Errors that indicate some sort of logic problem, or a misuse of the routines, possibly in an attempt to circumvent integrity or security checks. These errors are treated by CICS file control as disastrous errors, resulting in a system dump (if you have enabled such dumping) and, in most cases, in the transaction being abended with an AFCZ ABEND. For these, the value of the response and reason field is normally X'0215'.

The following topics explain the error codes for the third category of errors. These error codes are seen only in the exception trace entry. The format of the error code is X'ffaaaaaa', where *ff* identifies the type of failure, and *aaaaaa* is additional information provided for some of the failures. The possible values of *ff* for each trace point are described in the following topics.

Values for all shared data tables trace points

The following error codes can occur for the 0B12, 0B19, 0B1A, 0B29, and 0B2A exception trace points.

X'01'
A function was specified that requires the caller to be authorized via the CICS AFCB (authorized function control block), but the caller was not authorized.

X'0A'
The caller passed an invalid function code.

X'0B'
The caller specified an invalid format of SVC call.

X'0C'
An invalid parameter list address was passed to the SVC.

X'0D'
A function was specified that requires the value passed in register 1 to be 0, but it was not. The additional information contains the low-order three bytes of the value passed.

X'12'
A function was specified that requires the caller to be in Key 0 supervisor state, but the caller was not.

Values for 0B12 trace point

The AP 0B12 exception trace point is issued if an error is returned by the SVC on adding or deleting an access list entry when a shared data table is being closed.

In addition to the errors that can occur at all trace points, the following are possible:

X'02'
Shared data table services have not yet initialized (an anchor block for the region has not been created).

X'0E'
The specified data space STOKEN is invalid or the caller is not authorized to use it.

X'0F'
The CICS region has not completed initialization as a server.

X'13'
An attempt to delete an access list entry failed because the specified entry was not created by the data tables SVC.

All other errors result in a message being issued that contains the error code.

Values for 0B19 trace point

The AP 0B19 exception trace point is issued if an error is returned by the SVC on initializing as a shared data table server.

In addition to the errors that can occur at all trace points, the following are possible:

X'02'
An attempt is being made to add an access list entry before the CICS region has performed shared data table initialization (an anchor block for the region has not yet been created).

X'0E'
The specified data space STOKEN is invalid or the caller is not authorized to use it.

X'0F'
An attempt was being made to add an access list entry before the CICS region had completed server initialization.

All other errors result in a message being issued that contains the error code.

Values for 0B1A trace point

The AP 0B1A exception trace point is issued if an error is returned by the SVC on initializing a shared data table requester.

In addition to the errors that can occur at all trace points, the following is possible:

X'05'
The CICS region has already initialized as a shared data table requester, but is now running under a different request block from when it originally initialized.

All other errors result in a message being issued that contains the error code.

Values for AP 0B29 trace point

The AP 0B29 exception trace point is issued if an error is returned by the SVC on logging on as a shared data table server.

In addition to the errors that can occur at all trace points, the following are possible:

X'02'
The CICS region that is attempting to register (logon) as a server has not yet been initialized (an anchor block for the region has not been created).

X'04'
This CICS region has already registered (logged on) as a shared data tables server.

X'0F'
The CICS region has not completed server initialization.

X'14'
The AFCS anchor block does not exist.

X'15'
The CICS security block does not exist.

X'16'
Either the caller is not running in a user protection key (its PSW key is less than 8), or the caller's TCB does not normally execute in a user protection key (TCBPKF is less than 8).

All other errors result in a message being issued that contains the error code.

Values for 0B2A trace point

If the function code field contains X'15', the AP 0B2A exception trace point indicates an error on CONNECT (that is, on attempting to establish a connection to a remote file).

In addition to the errors that can occur at all trace points, the following are possible:

X'02'
Shared data table services have not yet initialized (an anchor block for the region has not been created).

X'03'
The requesting region has not completed initialization as a shared data tables requester.

X'05'
The CICS region is running under a different request block (RB) from when it initialized as a data table requester. The additional information part of the error code contains the RB address the call was made under.

X'72'
The LINK to the user-replaceable DFHACEE module to find the home address space's security userid has failed. The additional information part of the error code contains two bytes of the ABEND code from the LINK. The response and reason field accompanying this error is X'020B'.

All other errors result in a message being issued that contains the error code.

If the function code field contains X'16', the 0B2A exception trace point indicates an error on DISCONNECT (that is, on attempting to break the connection to a remote file). In addition to the errors that can occur at all trace points, the following are possible:

X'02'
Shared data table services have not yet initialized (an anchor block for the region has not been created).

X'03'
The requesting region has not completed initialization as a shared data tables requester.

X'05'
The CICS region is running under a different request block (RB) from when it initialized as a data table requester. The additional information part of the error code contains the RB address the call was made under.

X'07'
The caller has supplied an invalid index into the vector of file connections. The additional information part of the error code contains the low-order three bytes of the caller's index.

X'10'
The specified connection was broken previously and no longer exists. The additional information part of the error code contains the low-order three bytes of the caller's index into the vector of file connections.

All other errors result in a message being issued that contains the error code.

Analyzing errors from data tables cross-memory services

Following an unexpected error from data tables cross-memory services, an **X'0B15' exception trace entry** is made. This includes the response and reason codes and an error code field identifying the cause of the error. Such errors are all caused by a corruption of the routines or the system, or by a possible misuse of the routines.

For a response and reason of X'0215', the format of the error code is X'ffaaaaaa', where *ff* identifies the type of failure and *aaaaaa* is additional information provided for some of the failures. The possible values of *ff* are:

X'01'
An attempt to locate the CICS AFCB (authorized function control block) made by either the cross-memory retrieval routine or the connect vector lookup routine has failed.

X'02'
The requesting CICS region has not yet performed shared data tables initialization (an anchor block for the region has not yet been created and set up).

X'03'
The requesting region has not completed initialization as a shared data tables requester.

X'05'
The retrieval request was issued under a request block different from the one that performed initialization as an shared data table requester.

X'06'
The connect vector entry for the remote file does not contain the correct linkage index.

X'07'
The index of the connect vector entry for the remote file is beyond the end of the connect vector.

X'08'
The connect vector entry for the remote file is not marked as being in use.

X'09'
The cross-memory retrieval routine has not been called via the correct mechanism.

A response and reason of X'0400' means that the function code passed to the record management code running in the server region was an unrecognized value.

Dump information for data tables

Information relevant to data tables is included in a CICS system dump to help you determine the cause of a problem.

For information about the contents of dumps and how to obtain them, see Using dumps in problem determination.

The major control blocks that are used by shared data table support are included in the FILE CONTROL area of a formatted dump of the file-owning region. These control blocks are named:

Data Table Global Area
This is also known as the *shared data table header block*, so it uses the eye-catcher DFHDTHEADER.

Data Table Base Area
This is also known as the *shared data table block*, so it uses the eye-catcher DFHDTTABLE.

Data Table Path Area
This is also known as the *shared data table file block*, so it uses the eye-catcher DFHDTFILE.

The data table contents are not included in the CICS system dump because the data space storage in which the data table resides is not part of the CICS address space. The table entries reside in data space DFHDT001, the index nodes in DFHDT002, and the record data in data spaces DFHDT003, DFHDT004, DFHDT005, and so on, with new data spaces being added as required. If you want to see the contents of the data table, ask the system operator to use the MVS **DUMP** command to request a dump of the appropriate data space owned by the appropriate CICS startup job.

The operator command `DISPLAY J,CICS-startup-jobname` shows information about a CICS job, including the DSPNAMEs of data spaces that it owns. To dump the contents of data space DFHDT003, you can use the MVS **DUMP** command as follows:

1. Enter

   ```
   DUMP COMM=(title for your dump)
   ```

2. This generates an MVS console message

   ```
   * id IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
   ```

3. Reply to the message with

   ```
   REPLY id, DSPNAME='jobname'.DFHDT003
   ```

   and the data space storage is dumped.

   Note: It is possible, using the following asterisk notation, to dump the contents of *all* the data spaces owned by CICS:

   ```
   REPLY id, DSPNAME='jobname'.DFHDT*
   ```

   However, this should be used with care, because if there are many data spaces the dump data set could be huge.

4. Use `DISPLAY DUMP,TITLE` to see which SYS1.DUMPnn data set has been used.

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

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 5 (CICS TS 5.5) are included in the following sections of the online product documentation:

- Developing applications
- Developing system programs
- CICS security
- Developing for external interfaces
- Reference: application development
- Reference: system programming
- Reference: connectivity

Information that is NOT intended to be used as a Programming Interface of CICS TS 5.5, but that might be misconstrued as Programming Interfaces, is included in the following sections of the online product documentation:

- Troubleshooting and support
- Reference: diagnostics

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 5.5 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 5.5, 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

G

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

X