z/OS

IBM

TSO/E Customization

Version 2 Release 1

SA32-0976-00

Note

Before using this information and the product it supports, read the information in “Notices” on page 745.

This edition applies to Version 2 Release 1, modification 0 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

About this document

This document supports z/OS® (5650-ZOS).

TSO/E provides many ways to customize the functions and services that it provides. This document describes how you can maintain and customize TSO/E to suit your installation's data processing requirements.

Who should use this document

This document is intended for system programmers who are responsible for maintaining and customizing TSO/E on z/OS. You should have a good understanding and working knowledge of TSO/E operations and functions and your installation's processing requirements.

How this document is organized

This document is organized by parts and within each part, by chapters. Each part contains information about a specific category of customization tasks. Installation exits are described under the particular task that the exit is for. For a quick overview of customization facilities, see Part 10, “Reference,” on page 697.

Parts in this document provide the following information:

- Part 1, “Introduction,” on page 1 provides an overview of the ways you can customize TSO/E and a summary of the TSO/E exits. The first chapter introduces both required and optional customization. Each topic explains what you can customize and then provides a reference to another topic or document where you can get detailed customization information.

  The chapter on exits describes the *standard-format* exits that TSO/E provides. It also discusses the *TSO/E standard exit parameter list* that the standard-format exits receive, the standard return codes that the exits support, how you can install the exits, and the example exit (IKJEXIT) that TSO/E provides. The chapter also explains how this document describes the individual exits and contains a list of the exits that TSO/E provides. This list gives a brief description of each exit and refers you to the appropriate section of the document for more information.
- Part 2, “Considerations for installing, migrating and activating the functions of TSO/E,” on page 43 gives considerations on installing TSO/E. When you have already installed TSO/E on your system and want to migrate to a higher z/OS level, then you find migration considerations in Chapter 4, “Considerations for migrating TSO/E,” on page 53. How to activate the TSO/E functions after installing or migrating TSO/E, is described in: Chapter 5, “Activating the functions of TSO/E,” on page 57.
- Part 3, “Setting up and customizing the TSO/E environment,” on page 65 describes different ways you can change the TSO/E environment, and includes explanations about how to:
  - Define and customize TSO/VTAM
  - Set up the logon process by providing TSO/E logon procedures, controlling or limiting user access to TSO/E, and limiting the user's region size
  - Customize the TSO/E logon process, which includes customizing logon messages, customizing the reconnect option, limiting incorrect logon attempts, logon performance evaluation, using the logon pre-prompt exit, the logon

pre-display and post-display exits, the logon post-prompt exit, the logoff exit, and using logon language csects and load modules
  - Define TSO/E to ISPF and ISPF/PDF
  - Specify the authorized commands and programs that you want to allow users to use, and how to prevent the use of specific commands in the background
  - Specify the unauthorized commands that you want to run on the command invocation platform
  - Enable users to use the TRANSMIT and RECEIVE commands
  - Customize the way HELP data is used by including HELP members within other HELP members and by extending the use of the prompt mode HELP function
  - Make host services available to PC users
  - Monitor TSO/E resources and why you might want to do this
  - Define TSO/E performance objectives
  - Protect TSO/E resources by limiting user access to commands and data sets
  - Customize TSO/E for different languages

  With RACF® installed, your installation can use security enhancements. This part also explains how the security enhancements affect customizing TSO/E.
- Part 4, “Maintaining the UADS, RACF data base, and broadcast data set,” on page 193 contains information about maintaining TSO/E users on your system in the user attributes data set (UADS), the RACF data base, and the broadcast data set. It explains how to create, reformat, and update the UADS and broadcast data set and how to convert UADS information to a format acceptable to the RACF data base so you can use the RACF data base to maintain TSO/E users. This part also explains how to change the number of records that are reserved in the broadcast data set for notices.
- Part 5, “Customizing TSO/E commands,” on page 217 describes how to customize the following TSO/E commands:
  - ALLOCATE
  - ALTLIB
  - CONSOLE and CONSPROF
  - EDIT and its subcommands
  - EXEC
  - FREE
  - OUTDES
  - OUTPUT, STATUS, and CANCEL
  - PARMLIB
  - PRINTDS
  - SEND, LISTBC, and OPERATOR SEND
  - SUBMIT
  - TEST
  - TESTAUTH
  - TRANSMIT and RECEIVE
  - TSOLIB

  This part also explains how to customize the ways that users submit jobs and process job output, and what you must do to enable users to use individual user logs instead of the broadcast data set to store messages. With RACF installed, your installation can use security protected individual user logs with security labels. This part explains how to set up the security protected individual user logs with security labels. It also describes how to use various functions to customize the way in which TSO/E users allocate and manage data sets, including the use of Storage Management Subsystem (SMS). It explains how you can define output descriptors using either OUTPUT JCL statements or the OUTDES command, and how to add TEST and TESTAUTH commands and subcommands to be invoked under TEST and TESTAUTH.

- Part 6, "Customizing CLIST and REXX processing," on page 491 describes how to write exits to customize CLIST and REXX processing.

  It also briefly describes REXX language processor environments and replaceable routines.
- Part 7, "Support for a REXX compiler," on page 527 describes the support that TSO/E Release 3.1 provides for the installation and execution of a REXX compiler runtime processor. This part describes the routines and interfaces that TSO/E uses during the execution of compiled execs under a compiler runtime processor. This part also describes programming routines that TSO/E provides for use by a compiler runtime processor.
- Part 8, "Session Manager," on page 557 describes how to set up and customize a Session Manager environment. This includes changes you may have to make before using the Session Manager and different ways you can tailor the default environment. It also describes how you can monitor Session Manager users using three exits that TSO/E provides.
- Part 9, "Information Center Facility.," on page 583 describes how to set up and customize the Information Center Facility. Set up procedures include library considerations, logon procedures for Information Center Facility. users, and changes to support the use of additional products. This part also describes how you can tailor Information Center Facility. start-up and termination processing, variables, and the processing of different services or products. It also describes how to use the Application Manager to add applications to the Information Center Facility. and add or change the panels for your own requirements. This part also contains information about the Information Center Facility. naming conventions, application, panel, and CLIST and REXX exec hierarchy, panels and their associated HELP panels, and the PANELID and TRACE commands, which help you locate any problems when you change the default panel hierarchy.
- Part 10, "Reference," on page 697 summarizes the customization facilities you can use to customize TSO/E. Also included in this part is the syntax for TSO/E macros.
- The appendices include a description of how to execute the terminal monitor program as a batch job, and an example for the logon pre-prompt exit.

How to use this document

First review the table of contents to become familiar with the document's structure. After that, read Part 1, "Introduction," on page 1 to learn about the different ways you can customize TSO/E and the exits TSO/E provides. Chapter 1, "Customization overview," on page 3 explains both required and optional customization tasks. It provides an overview of each task and refers you to another topic or document for detailed information. Chapter 2, "Writing exit routines," on page 23 is of particular interest to system programmers who write exits to customize different TSO/E functions. You may also want to browse through other chapters to determine how you want to customize TSO/E.

After you decide how to customize TSO/E, you will most likely refer to the individual chapter that describes a particular customization task. On an ongoing basis, you may use this document as a reference. For these purposes, you can refer to the table of contents and the index to find the information you need.

Where to find more information

Please see *z/OS Information Roadmap* for an overview of the documentation associated with , including the documentation available for z/OS TSO/E.z/OS

How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html).
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   US
4. Fax the comments to us, as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
  z/OS V2R1.0 TSO/E Customization
  SA32-0976-00
- The topic and page number that is related to your comment.
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at z/OS support page (http://www.ibm.com/systems/z/support/).

z/OS Version 2 Release 1 summary of changes

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

- *z/OS Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*
- *z/OS Introduction and Release Guide*

Part 1. Introduction

This part provides an overview of the TSO/E customization process and introduces TSO/E exits. Before you start to customize TSO/E, IBM® suggests that you develop a customization plan. In the plan, identify specific customization that you intend to do and how you intend to do it.

Chapter 1, “Customization overview,” on page 3, provides information to help you develop that plan. There you can read a short summary about each customization task that will help you decide whether to do that task. If you need more information to help you plan, a pointer directs you to another topic in this document or to another document.

Chapter 2, “Writing exit routines,” on page 23, introduces the exits that TSO/E provides. The introduction includes a list of the exits, a summary of the ways you can use each exit, and a pointer to detailed information you need for coding the exit. This chapter also discusses the standard exit interfaces and exit installation.

Chapter 1. Customization overview

TSO/E customization is the process whereby you tailor TSO/E functions to fit the specific needs of your installation. To tailor a TSO/E function, you might make a change or addition to TSO/E itself or to some other related IBM product or z/OS element, such as the Advanced Communications Function for VTAM® (Virtual Telecommunications Access Method).

Some customization is required before you can use TSO/E, but most is optional. If you elect not to do optional customization on a specific TSO/E function, that function then works according to IBM-provided defaults. If TSO/E is new to your installation, you initially might want to do a minimum amount of optional customization. Your users can then use TSO/E, and you can do additional customization only when inadequate TSO/E performance or reduced TSO/E capabilities indicate the need to do so.

There are a number of ways for you to customize TSO/E and the processing of jobs submitted through TSO/E. TSO/E itself provides exits for which you can write exit routines, initialization values you can change, macros you can use, and procedures you can change. Each of the following z/OS elements and products interact with TSO/E. They provide interfaces that you can use to customize the way TSO/E and the element work together. The elements are:

- MVS™
- DFSMSdfp
- ISPF/PDF
- Job Entry Subsystems (JES2 and JES3)
- Security Server RACF
- VTAM
- PSF for z/OS

The customization facilities discussed in this document are summarized in several places. For a description of the TSO/E macros, see Chapter 51, "Macro syntax," on page 707. For a brief description of TSO/E exits, see Chapter 2, "Writing exit routines," on page 23. For a list of the other customization facilities, see Chapter 50, "Overview of facilities for customizing TSO/E," on page 699.

Required customization

Before users can use TSO/E, you *must* do several customization tasks. After you complete required customization, users will be able to log on and issue commands from TSO/E READY mode. You must:

- Define TSO/E to VTAM
- Define those users who will be allowed to log on to TSO/E
- Create at least one TSO/E logon procedure for users

If you currently have TSO or TSO/E, you probably already have the required customization discussed in this topic. However, when you upgrade to a new release of TSO/E, IBM suggests that you ensure that the user definition data sets that you are now using are correctly formatted. The data sets are the user attributes data set (UADS) and the broadcast data set. If you have RACF installed, you can define your TSO/E user information in the RACF data base. For more information, see Part 4, “Maintaining the UADS, RACF data base, and broadcast data set,” on page 193. You must ensure that the UADS , the RACF data base, or both and the broadcast data set are synchronized.

Even if you don't currently have TSO/E, the UADS and broadcast data sets may already be cataloged on your system. MVS requires that they be cataloged unless you have customized MVS to eliminate the requirement.

Note: The IBM-provided MVS master JCL refers to the UADS as SYS1.UADS.

Access methods - VTAM

The access method, VTAM, provides an interface that enables terminals and TSO/E to communicate.

If your installation uses VTAM, you must write a procedure for starting TSO/VTAM, and you must use VTAM APPL definition statements to define the TSO/E application and to define the terminals from which users will be allowed to use TSO/E. In addition, because VTAM does not recognize the TSO/E LOGON command defined in *z/OS TSO/E Command Reference* you must define the LOGON command to VTAM. For details about defining TSO/E to VTAM, see Chapter 6, “Defining and customizing TSO/VTAM and TSO/TCAM time sharing,” on page 67.

User definitions

Before a user can log on to TSO/E, you must define that user to TSO/E. You do this by storing information about the user in the UADS or the RACF data base.

You can use the RACF data base only if your installation is using RACF. If your installation is using RACF, you can use the UADS only, the RACF data base only, or both. Even if you use only the RACF data base, z/OS requires that you keep the UADS cataloged unless you have customized MVS to eliminate this requirement.

For information about defining users, see Part 4, “Maintaining the UADS, RACF data base, and broadcast data set,” on page 193.

TSO/E logon procedure

A TSO/E logon procedure contains JCL statements that execute the required program and allocate the required data sets to enable a user to acquire the resources needed to use TSO/E. You must provide at least one logon procedure for your installation's TSO/E users. For information about how to write a logon procedure, see Chapter 7, “Setting up logon processing,” on page 75.

Optional customization

Optional customization enables you to change or extend the capabilities of TSO/E functions or to make other IBM products available to TSO/E users. This topic introduces the kinds of optional customization you can do. Other topics provide the detailed information you need to do a particular customization task. This topic introduces how you can:

- Customize the TSO/E environment, which includes: customizing VTAM, defining logon limits, customizing the logon process, making commands and programs available, specifying commands not supported in the background, specifying commands and programs to run on the command/program invocation platform, customizing HELP data, making host services or ISPF/PDF

available, setting performance objectives, monitoring and protecting TSO/E resources, and customizing TSO/E for different languages
- Maintain the UADS, the broadcast data set, and the RACF data base
- Customize the use of commands
- Customize CLIST and REXX processing
- Make Session Manager available and customize the Session Manager
- Make the Information Center Facility available and customize the Information Center Facility.

TSO/E environment

Customization of the TSO/E environment generally refers to customization that makes a TSO/E facility available or customization that changes default values that affect TSO/E. You can customize:
- VTAM -- You can change VTAM session protocols, provide substitute characters for unavailable keyboard characters, and override the default values used to start VTAM.
- Logon limits -- You can limit and manage the maximum number of concurrent logons, limit the user's region size, and limit user access to applications.
- The logon/logoff process -- You can change how often the system displays the `logon proceeding` message, limit the number of attempts a user can make at entering information in response to logon prompts, tailor the reconnect option, and suppress messages that are generated during the execution of the logon job. You can also review factors that affect logon performance, such as using STEPLIBs in logon JCL, and you can write exits to further customize the logon/logoff process. Your installation can use security labels (SECLABEL) if the proper products are installed.
- ISPF/PDF -- You can make ISPF/PDF available to TSO/E users.
- Authorized commands and programs -- You can select which authorized commands and programs users can use.
- Command/program invocation platform support -- You can invoke TSO/E commands and programs on the command/program invocation platform. Both authorized and unauthorized commands and programs are supported.
- Command availability in the background -- You can make specific commands unavailable for use in the background.
- TRANSMIT and RECEIVE availability -- You can make the TRANSMIT and RECEIVE commands available.
- HELP data set usage -- You can customize the use of HELP data set members.
- Host services availability -- You can make host services available to personal computer (PC) users.
- Language support -- You can provide information to users in their national language.

In addition, this topic also discusses how you can monitor and protect TSO/E resources and provides guidance for setting TSO/E performance objectives.

VTAM

VTAM installations have several customization options available. SYS1.PARMLIB member TSOKEY00, or an alternate member, contains values used to initialize TSO/VTAM. You have the option to change these values. If you plan to use Session Manager, you may have to make additional changes to TSOKEY00 or the alternate member. For information about SYS1.PARMLIB members, see *z/OS MVS*

Initialization and Tuning Reference. For information about changes you may have to make for Session Manager, see Chapter 45, “Setting up a Session Manager environment,” on page 559.

Session protocols define the rules that VTAM uses to manage a terminal session. If you wish, you may change the protocols. For information about changing the protocols, see Chapter 6, “Defining and customizing TSO/VTAM and TSO/TCAM time sharing,” on page 67.

Some terminal keyboards may not contain all of the characters your installation needs. If this is the case, you can provide translation tables that allow users to substitute other characters for the unavailable characters. For information about providing translation tables, see Chapter 6, “Defining and customizing TSO/VTAM and TSO/TCAM time sharing,” on page 67.

Concurrent logons

In z/OS, three factors determine the maximum numbers of users who can be logged onto TSO/E concurrently: the number of available logons, the number of address spaces that can execute concurrently, and the number of VTAM APPL definition statements defined to VTAM. The smaller of these three factors limits the number of concurrent logons. You should examine, and if necessary, adjust these factors to meet your installation's needs.

If you need to further control or limit logons, you can define groups and assign TSO/E users to these groups. You can then write a logon pre-prompt exit routine that allows or disallows a user to log on based on the group to which the user is assigned.

For more information about limiting concurrent logons, see Chapter 7, “Setting up logon processing,” on page 75.

Region size and user access to applications

You have the option to limit the size of a TSO/E user's region and the applications that a user can access. When users log on to TSO/E, they can specify a region size, or they can accept an installation-defined default. You can define a default that applies to all TSO/E users, a default that varies from one logon procedure to another, or a default that varies from one user to another. Through the use of the logon pre-prompt exit, you can monitor, and if necessary, adjust the region size requested by a user.

Different users or groups of users might need to access different sets of applications. Some users may need to access all applications while other users need to access only one or two applications. By providing several logon procedures, each providing access to different applications, and by restricting individuals to only certain logon procedures, you can limit the applications to which any user has access.

For more information about limiting these factors, see Chapter 7, “Setting up logon processing,” on page 75.

Logon and logoff process

There are several ways you can customize the logon and logoff process. By changing initialization values, you can:

- Change the frequency with which TSO/E issues the `logon proceeding` message
- Limit the number of attempts a user who is not defined to RACF can make at entering information in response to logon prompts

- Tailor the reconnect process
- Suppress messages generated during execution of the logon job

There are also steps you can take that may improve logon performance and exits that you can write to further customize logon processing. The remainder of this topic provides an overview of the ways you can customize logon processing.

The `logon proceeding` message tells the user that the logon process is taking place. TSO/E issues this message when the logon process has taken more than a predefined amount of time (the TSO/E-provided default is to issue the message every five minutes). If you wish, you can change the frequency with which TSO/E issues this message.

If a user supplies an incorrect operand when attempting to log on, TSO/E requests that the user reenter the operand. TSO/E allows the user to supply a predefined number of incorrect operands before TSO/E cancels the logon attempt. The TSO/E-provided default is 10. If a user who is *not* defined to RACF exceeds the allowable number of attempts, TSO/E discontinues the logon process for that user. To log on, that user must start over by again issuing the LOGON command. You have the option to change the total number of attempts a user can make to enter operands before having to reissue the LOGON command.

When a user disconnects from the system, the user's address space remains available to that user for a predefined amount of time. During the time the address space is available, the user can reconnect to the system without going through the logon process. After the time limit expires, the user can no longer reconnect and must log on to reaccess the system. If you wish, you can change the amount of time the address space remains available.

Each logon job produces a number of messages that the system writes to a system output (SYSOUT) data set. The data set, by default, is sent to output class A. Rarely is it necessary to refer to these messages. Therefore, you may wish to send these messages to a different class, one that you can hold and later purge.

There are a number of steps you can take that may improve the performance of the logon process. If you feel that the logon process takes too long, review the factors that can affect its performance and make the necessary changes.

The logon pre-prompt exits enable you to do additional customization of the logon process. Exit IKJEFLD lets you monitor, change, or supplement information provided on the LOGON command, cancel a logon, or interact with the user by sending messages and requesting a reply. You can use the authorized logon exit IKJEFLD1, which receives the standard TSO/E exit parameter list and lets you perform the same functions as IKJEFLD, as well as authorized functions. With IKJEFLD1, you can also:
- Specify the first TSO/E user command to be issued in the session.
- Specify job and SYSOUT classes.
- Bypass RACF processing.
- Specify the relative block address (RBA) of the user's mail directory entry in the broadcast data set.
- Provide a four-byte user word for communication with other logon and logoff exits.

- Specify a default security label (SECLABEL) to be used for the session. SECLABEL is recognized only when RACF is active and security label checking has been activated.
- Set up the console profile for the user.
- Specify primary and secondary languages to be used for displaying translated information.

TSO/E also provides the post-prompt exit IKJEFLD3 to further customize the logon process. After the prompting for logon input is complete, you can use this exit to add, examine, and modify JCL statements associated with the terminal job.

You can customize logon panels and logon help panels using logon panel modules. TSO/E supports these logon panel modules in different languages.

TSO/E provides pre-display and post-display logon exits that let you further customize the logon process. These exits, IKJEFLN1 and IKJEFLN2, allow you to supply default information, update information, validate user-supplied information, and re-prompt the user for information, if needed.

The pre-display exit IKJEFLN1 allows you to:
- Update information contained on the logon panel
- Process any installation-defined fields on the logon panel
- Update fields in the PSCB and the UPT

The post-display exit IKJEFLN2 allows you to:
- Process and validate fields on the logon panel
- Request display of help screens

With TSO/E, you can use the authorized logoff exit IKJEFLD2 to customize the logoff process. IKJEFLD2 receives the standard TSO/E exit parameter list, including the user-word value set by logon exit IKJEFLD1. IKJEFLD2 can perform clean-up operations, gather accounting information, control information written to the UADS and RACF data base, and issue the LOGOFF or LOGON command to control re-logons.

For detailed information about each customization procedure and the exits, see Chapter 8, “Customizing the logon and logoff process,” on page 83.

ISPF/PDF

To enable TSO/E users to use ISPF/PDF, you must define ISPF/PDF to TSO/E. You do this by modifying the users' logon procedures to allocate the ISPF/PDF data sets.

Optionally, you can specify the TSO/E commands that a user can issue from ISPF/PDF panels, and you can allow users to use the Session Manager from ISPF/PDF. By default, ISPF/PDF users can issue all TSO/E commands that reside in the link pack area (LPA) except:
- LOGOFF
- LOGON
- PRINTDS
- RACONVRT
- SYNC
- TEST

You can change this list by modifying an ISPF/PDF module.

In order for a user to keep a session journal, ISPF/PDF and Session Manager must run concurrently. To allow Session Manager and ISPF/PDF to run concurrently, you must install specific ISPF/PDF exit routines (ISPSC93 and ISPSC94) andpanels. See *z/OS ISPF Planning and Customizing* about these exits.

For more information about how to make ISPF/PDF available, see Chapter 9, "Defining TSO/E to ISPF and ISPF/PDF," on page 145.

Authorized commands and programs and unavailable background commands

You have the option to make authorized commands and programs available and to make specific commands unavailable in the background. TSO/E users cannot use authorized commands or programs until you define the authorized commands and programs you want them to be able to use. In addition, there may be TSO/E commands that you do not want invoked in the background. If there are such commands, you must also define them.

During installation, TSO/E requires you to specify certain authorized commands and programs that users are to be able to issue and certain unsupported background commands. You may also want to add to that list. For a list of the commands and programs that TSO/E requires you to specify, see SYS1.SAMPLIB member IKJTSO00. For more information about defining these commands or programs, see Chapter 11, "Specifying authorized commands/programs, and commands not supported in the background," on page 151.

Command/Program invocation platform

An application program can run commands and programs on a command/program invocation platform. Command/program invocation platforms eliminate the need for MVS task initialization and termination each time you invoke a command or program. The TSO/E service facility allows installations to create a command/program invocation platform on which to run TSO/E commands and programs. Authorized commands and programs can execute on a command/program invocation platform, even if you are not using the TSO/E service facility.

This support helps reduce the system overhead associated with the initialization and termination of TSO/E commands and programs by creating a command/program invocation task once per command or program invocation in a TSO/E environment.

You can identify which programs can execute on a program invocation platform by using the PLATPGM statement in SYS1.PARMLIB member IKJTSOxx.

You can identify which commands can execute on a command invocation platform by using the PLATCMD statement in SYS1.PARMLIB member IKJTSOxx.

You can use the TSO/E service facility to create the command/program invocation platform environment, execute commands and programs, and terminate the command/program invocation platform environment. For more information about specifying eligible commands and programs, see Chapter 12, "Specifying commands and programs for the command/program invocation platform," on page 157.

TRANSMIT and RECEIVE availability

To enable TSO/E users to use the TRANSMIT and RECEIVE commands, you must add the TRANSMIT and RECEIVE commands to the list of authorized commands, specify installation defaults for TRANSMIT and RECEIVE, and possibly modify job entry subsystem (JES) initialization statements. For information on how to make these commands available, see Chapter 13, “Setting up the TRANSMIT and RECEIVE environment,” on page 161.

HELP data

There are two ways you can customize the use of HELP data. You can set up HELP members so they merge information from other members for display to the user. You can also provide the prompt mode HELP function, which provides help from the HELP data set for a user who omits or incorrectly specifies a positional operand on a TSO/E command. The function provides help when the user requests a second-level message and none are available or all have been displayed. TSO/E, by default, provides the prompt mode HELP function for:

- ALTLIB
- ATTRIB
- CALL
- CANCEL
- EDIT
- EXEC
- HELP
- OUTPUT
- RACONVRT
- RUN
- SEND
- SYNC
- TRANSMIT

For information about customizing HELP processing, see Chapter 14, “Customizing the HELP data set,” on page 165.

Host services availability

MVSSERV, a TSO/E command processor, enables personal computer (PC) programs to use host computer resources. *Server* programs on the host computer provide services to *requester* programs on the PC. Before TSO/E users can use this facility, the PC must be properly configured, and you must make the servers and requesters available.

A program called an access method driver defines the protocol for managing server/requester communication. You can change this protocol by providing your own access method drivers.

For more information about what you must do to make MVSSERV available and information about customization options, see Chapter 15, “Making host services available to PC users,” on page 167.

Resource monitoring

After your installation starts to use TSO/E, you may want to monitor command use and transaction performance and keep statistics. By monitoring commands, you can:

- Identify the most frequently used commands; you may want to improve the performance of these commands
- Use information such as frequency of use as a basis for billing customers
- Audit for security violations

Monitoring TSO/E performance helps you determine whether TSO/E performance is satisfactory. For more information about monitoring TSO/E, see Chapter 16, “Monitoring TSO/E resources,” on page 169.

Performance objectives

Part of TSO/E customization should include setting performance objectives for TSO/E. When you set performance objectives, ask yourself the following kinds of questions:

- What kind of user response do I want TSO/E to provide?
- Which commands do I want to provide the best response?
- Do some users require better response time than others?
- What performance impact will TSO/E have on the rest of my system?

There are a number of factors to consider and trade-offs to make when you define performance objectives. Also, you should keep in mind that a change you make to improve TSO/E performance will probably have an opposite affect in some other part of your system. For more information about setting performance objectives, see Chapter 17, “Defining performance objectives for TSO/E,” on page 171.

TSO/E resource protection

To protect TSO/E resources, you can limit the commands that users can issue and limit user access to data sets. You can limit the commands that users can issue from TSO/E READY mode, from Session Manager, from the background, and from ISPF/PDF.

From TSO/E READY mode, by default, users cannot use the ACCOUNT, CONSOLE, CONSPROF, OPERATOR, RACONVRT, SYNC, SUBMIT, OUTPUT, STATUS, CANCEL, PARMLIB, and TESTAUTH commands. You give users authority to use these commands when you define them to TSO/E by using either the ACCOUNT command and/or RACF commands. You can optionally write TSO/E exits to authorize users to use the CONSOLE, CONSPROF, PARMLIB, and TESTAUTH commands. For the SUBMIT, OUTPUT, STATUS, and CANCEL commands, you can also write TSO/E exits and exits provided by JES2 and JES3 to customize and restrict how users submit jobs and process job output.

By default, users can use the SEND, LISTBC, TRANSMIT, RECEIVE, FREE, OUTDES, EXEC, and PRINTDS commands from TSO/E READY mode. You can write TSO/E exits to restrict the use of these commands. You can also write exits for the SEND subcommand of the OPERATOR command to restrict users who are authorized to use the OPERATOR command from using the SEND subcommand.

A user who is using Session Manager can, by default, issue all TSO/E commands and Session Manager commands. By writing Session Manager exits, you can limit the commands that users can issue from Session Manager.

Users, by default, can also issue most TSO/E commands from the background or from ISPF/PDF panels. By changing a SYS1.PARMLIB member or by coding a TSO/E CSECT, you can limit the use of commands in the background. Modifications you can make to an ISPF/PDF module allow you to limit the commands used from ISPF/PDF.

You also have the option to limit a user's access to data sets. You can enforce this option through RACF or through the MVS allocation input validation routine (IEFDB401).

With RACF installed, your installation can use security labels (SECLABELs) and your security administrator can activate security label checking. In this case, resources and users have security labels associated with them. Users can only access resources that have been authorized for them to use through RACF.

Security label checking affects the processing of several TSO/E commands, such as SEND, LISTBC, TRANSMIT, and RECEIVE. For information about the processing of these TSO/E commands with security labels, see the individual topics in this document.

For information about setting up security labels, see *z/OS Security Server RACF Security Administrator's Guide*. For more information about providing TSO/E resource protection, see Chapter 18, "Protecting the resources TSO/E users can access," on page 175.

Language support

TSO/E takes advantage of the MVS message service to allow you to provide TSO/E information to users in their national language. TSO/E information includes TSO/E messages, help information for TSO/E commands, and the TRANSMIT command full-screen panel. In addition, the TSO/E CONSOLE command supports the display of translated system messages issued during an extended MCS console session.

To provide translated information at your installation, you need to:
- Initialize and activate the MVS message service (MMS).
- Set up languages for users.

For help information, you also need to specify the help data sets for each language supported by your installation.

For each user, you can specify a primary language and a secondary language to be used for displaying translated information. Languages can be specified using one of the following methods:
- If your installation has RACF installed, you can use RACF commands to set up languages for users
- You can use the logon pre-prompt exit, IKJEFLD1, to set up languages for users
- Individual users can use the TSO/E PROFILE command to set up or change languages for themselves.

You can use the IKJTSOxx member of SYS1.PARMLIB to define the help data sets to be used for different languages. In IKJTSOxx, you can:
- Specify help for any number of languages
- Specify up to 255 data sets to be searched for help text in a particular language.

For more information about customizing TSO/E for language support, see Chapter 20, "Customizing TSO/E for different languages," on page 185.

UADS and broadcast data set maintenance

During daily operation, you will probably add new users to TSO/E, delete users, or change information about users. After a number of these changes, available space in the UADS and broadcast data sets may become fragmented. To reduce fragmentation, you can reformat and synchronize these data sets. If the data sets need more space, you can reallocate them. For information about maintaining these data sets, see Chapter 24, "Working with the UADS and broadcast data set," on page 199.

If you have RACF installed, you can define your TSO/E user information in the RACF data base. For more information, see Part 4, "Maintaining the UADS, RACF data base, and broadcast data set," on page 193. If you use the RACF data base, you must synchronize the RACF data base with the broadcast data set.

UADS to RACF conversion

If your installation uses RACF, you have the option to define TSO/E users through the RACF data base instead of through the UADS. To do so you must first convert the UADS information to a form acceptable to RACF. If you use the RACF data base, MVS still requires that the UADS remain cataloged unless you have customized MVS to eliminate the requirement. For information about converting from the UADS to RACF, see Chapter 25, "Using the RACF data base to maintain TSO/E users," on page 209.

Commands

TSO/E customization facilities enable you to customize a number of TSO/E commands. You can:

- Customize the way users run a console session using the CONSOLE and CONSPROF commands.
- Customize the way users allocate and manage data sets.
- Customize the way users define alternative libraries of REXX execs and CLISTs using the ALTLIB command.
- Customize the way both EDIT and EXEC work.
- Customize how users send and retrieve messages using the SEND and LISTBC commands, and the SEND subcommand of the OPERATOR command.
- Allow or prevent users from using TRANSMIT and RECEIVE. If you allow these commands, you can tailor their processing.
- Customize how the FREE, PARMLIB, and OUTDES commands work.
- Customize the way users submit jobs, process output, and print data sets.
- Customize the way users test assembler programs.
- Customize the way users define alternative load module libraries using the TSOLIB command.

Data set allocation and space management

There are several ways you can customize how users allocate and manage data sets. You can use Storage Management Subsystem, the MVS allocation input validation routine, output descriptors, or the Information Center Facilityspace management service. You can write exit routines to customize ALLOCATE command processing. You can set a default data set disposition for the ALLOCATE command in SYS1.PARMLIB member IKJTSOxx.

You can use Storage Management Subsystem to manage your system's data and storage and simplify how users allocate data sets. When users issue the TSO/E ALLOCATE command, they need not be concerned with different storage, data, or management-related operands. Instead, they can use ALLOCATE operands that are related to Storage Management Subsystem to define various data set attributes.

The MVS allocation input validation routine (IEFDB401) enables you to monitor each ALLOCATE command. You can check and, if necessary, change information that the user provides on the ALLOCATE command.

An output descriptor specifies processing options for a system output (SYSOUT) data set. You can use output descriptors to eliminate the need for users to specify output-related operands on the ALLOCATE or PRINTDS commands. If an output descriptor has been defined, when a user issues the ALLOCATE command to allocate a system output data set or issues the PRINTDS command, the user can specify the OUTDES operand and the name of an output descriptor instead of specifying output-related operands. You can define output descriptors by naming and coding OUTPUT JCL statements. If you have JES2 installed, you and your installation's users can also use the TSO/E OUTDES command to define output descriptors. TSO/E provides two exits for the OUTDES command that allow you to customize the use of OUTDES.

The Information Center Facilityspace management service manages data set space. It does this when the data set is almost full by either compressing the data set or by reallocating a larger data set. It can also allocate a data set that does not exist. This service is available to not only the Information Center Facility, but to anyone who wants to use it. Parameter values, available in a CLIST, define the parameters that this service uses to manage data sets. You have the option of changing the default parameter values.

TSO/E provides initialization and termination exits that you can use to customize the ALLOCATE command. The initialization exit can restrict users from using the ALLOCATE command, can change operands supplied by the user, or pass the address of a new command buffer. The termination exit can perform clean-up processing, such as releasing storage obtained by the initialization exit.

You can use member IKJTSOxx of SYS1.PARMLIB to set a default value for the data set disposition specified on the ALLOCATE command. You can set the default to SHR or OLD. If a user issues the ALLOCATE command without specifying a data set disposition, the disposition defaults to the setting in IKJTSOxx.

When you set a default data set disposition in IKJTSOxx, you can make it take effect immediately using the PARMLIB command with the UPDATE operand. You can also list the current ALLOCATE default using the LIST operand of PARMLIB.

For information about data set allocation and the space management service, see Chapter 27, “Customizing how users allocate and manage data sets,” on page 219.

ALTLIB command

TSO/E provides the ALTLIB command, which lets users define alternative libraries of REXX execs and CLISTs for implicit execution. TSO/E also provides initialization and termination exits that you can use to customize the ALTLIB command. The initialization exit can restrict users from using the ALTLIB command, can change operands supplied by the user, or pass the address of a new command buffer. The termination exit can perform clean-up processing, such as releasing storage obtained by the initialization exit. For more information, see Chapter 28, “Customizing the ALTLIB command,” on page 229.

CONSOLE and CONSPROF commands

The CONSOLE command allows users with CONSOLE command authority to perform MVS operator functions from a TSO/E terminal. The CONSOLE command establishes an extended MCS console session with MVS console services. During the console session, users can issue MVS system and subsystem commands and obtain responses to those commands (solicited messages). Users can also receive unsolicited system messages.

The CONSPROF command lets users with CONSOLE command authority establish or change their console profiles used to tailor message processing during an extended MCS console session.

You can customize the use and processing of these commands by:

- Setting up a console profile for each user
- Defining console attributes for each user
- Defining installation defaults for the user's message tables in SYS1.PARMLIB member IKJTSOxx
- Writing CONSOLE and CONSPROF exits

You can set up a console profile for a user using the authorized logon pre-prompt exit IKJEFLD1. The console profile controls the display of system messages issued during an extended MCS console session. For more information about using IKJEFLD1, see "Possible uses" on page 109.

The console attributes define various characteristics of a user's console session, including the user's MVS command authority. You can use the MVS VARY command to specify the console attributes for a user. If you have RACF installed, you can optionally specify the console attributes in the OPERPARM segment of the user's RACF profile using the RACF ADDUSER or ALTUSER command.

You can use SYS1.PARMLIB member IKJTSOxx to define installation defaults for users' message tables. Message tables hold system messages the user does not want displayed during a console session. IKJTSOxx contains a CONSOLE statement you can use to specify the initial and maximum sizes of the message tables. For more information about SYS1.PARMLIB member IKJTSOxx, see "Defining installation defaults for the CONSOLE command" on page 234.

TSO/E provides exits you can use to customize the use and processing of the CONSOLE and CONSPROF commands. Using these exits, you can:

- Change operands that users specify on the CONSOLE and CONSPROF commands
- Control the size of the message tables
- Take action if a message table reaches 80% or 100% capacity
- Grant or deny CONSOLE command authority to users
- Customize the console profile message, IKJ55351I, or issue an installation defined message
- Request termination of the console session

For more information on the CONSOLE and CONSPROF commands, see Chapter 29, "Customizing the CONSOLE and CONSPROF commands," on page 233.

EDIT command

TSO/E provides several ways for you to customize the EDIT command. Using the facilities provided, you can:

- Define additional data set types
- Change the attributes of data set types
- Write syntax checkers and syntax checker exits for installation-defined data set types
- Write an exit to change how the RENUM, MOVE, and COPY subcommands handle line numbering
- Define additional subcommands
- Preallocate EDIT utility work data sets

TSO/E has defined a number of data set types. If these types fail to meet your needs, you can either change the attributes of the data set types or you can define your own data set types.

For each data set type that you define, you can write a syntax checker and a syntax checker exit. The syntax checker can detect errors when a user edits a data set of the type recognized by the syntax checker. You can use the exit to obtain information that the user specifies on the EDIT command.

The EDIT subcommands, COPY, MOVE, and RENUM all change line numbers or add line numbers to a data set. If the numbering algorithms these subcommands use do not meet your needs, or you wish to change the way these subcommands work, you can write an exit. One exit serves all three subcommands.

If the functions provided by the EDIT command are inadequate, you can supplement those functions. You do this by writing your own subcommand processors and adding them to the system.

The EDIT subcommands, MOVE and COPY, both require the allocation of data sets for work space. Depending on a number of factors, the subcommands use either temporary space or permanent space. You have the option to either preallocate the space or to allow EDIT to allocate it dynamically as it's needed.

For a description of EDIT and its subcommands, see *z/OS TSO/E Command Reference*.

For more customization information, see Chapter 30, “Customizing the EDIT command,” on page 255.

EXEC command

TSO/E provides two exits that enable you to customize the EXEC command. The initialization exit enables you to monitor, change, or supplement information that users provide on the EXEC command. The termination exit enables you to do any clean-up that may be needed because of processing the initialization exit performs and change the default CONTROL options for CLISTs. For information, see Chapter 31, “Customizing the EXEC command,” on page 277.

FREE command

TSO/E provides two exits that enable you to customize the FREE command. The initialization exit enables you to monitor, change, or supplement information that users provide on the FREE command. The termination exit enables you to do any clean-up that may be needed because of processing the initialization exit performs. For information, see Chapter 32, “Customizing the FREE command,” on page 283.

LISTBC, OPERATOR SEND, and SEND commands

Users issue the SEND command to send messages to other users. Users who are authorized to use the OPERATOR command can issue the SEND subcommand to send messages to users. The SEND command and the OPERATOR SEND subcommand may store messages in the broadcast data set depending on the operands the user specifies. Users issue the LISTBC command to retrieve messages that SEND stores.

By default, SEND and OPERATOR SEND store messages in the broadcast data set and LISTBC retrieves stored messages from this data set. You can define individual user logs that SEND, OPERATOR SEND, and LISTBC use to store and retrieve messages instead of using the broadcast data set.

To customize SEND, OPERATOR SEND, and LISTBC processing, you can use SYS1.PARMLIB member IKJTSOxx to define installation defaults. You can:
- Specify whether users can use the SEND command
- Specify whether users who are authorized to use the OPERATOR command can use the SEND subcommand
- Specify whether user logs are used and name those logs
- With RACF installed, specify whether security labels (SECLABELs) will be used to protect messages

To further tailor the process, you can write exits for LISTBC, OPERATOR SEND, and SEND. Using these exits, you can:
- Provide additional restrictions on who can or cannot send messages
- Override defaults that you previously set in IKJTSOxx
- Tailor the use of user logs
- Change operands that users specify on LISTBC, OPERATOR SEND, or SEND
- Change message text or format messages

For more information, see Chapter 36, “Customizing how users send and retrieve messages,” on page 333.

PARMLIB command

TSO/E provides the PARMLIB command, which lets users list TSO/E options that are in effect on the system and update the options as specified in IKJTSOxx members of SYS1.PARMLIB. You can also use the CHECK parameter of PARMLIB to check the syntax of any IKJTSOxx member of SYS1.PARMLIB including the active member.

TSO/E provides initialization and termination exits that you can use to customize the PARMLIB command. The initialization exit can restrict users from using the PARMLIB command, change operands supplied by the user, or pass the address of a new command buffer. The termination exit can perform clean-up processing, such as releasing storage obtained by the initialization exit.

TEST and TESTAUTH commands

The TEST command lets users test unauthorized assembler programs. The TESTAUTH command lets users test authorized assembler programs. Users can also test APPC/MVS transaction programs written in assembler with the TEST and TESTAUTH commands. You can customize these commands in the following ways:
- Supply installation-written subcommands of the TEST and TESTAUTH commands, and installation-written command processors to be invoked under TEST and TESTAUTH. You must define those subcommands and command processors to TEST and TESTAUTH using CSECT IKJEGSCU or member IKJTSOxx in SYS1.PARMLIB.
- You can write exit routines to tailor the processing of the TEST and TESTAUTH commands and their subcommands.

  The command and subcommand exits for TEST and TESTAUTH allow you to:
  - Restrict certain users from using the command or subcommands.
  - Change the operands a user specifies on the command or subcommands.
  - Perform clean-up processing before the command or subcommand ends.

For more information, see Chapter 37, “Customizing the TEST command,” on page 369 and Chapter 38, “Customizing the TESTAUTH command,” on page 383.

TRANSMIT and RECEIVE commands

If you have made TRANSMIT and RECEIVE available, you may wish to customize the use of those commands. Through the customization facilities provided, you can:

- Restrict who can use the commands and on which network paths
- Allow, disallow, or require encryption of transmitted data
- Modify data encryption and decryption processing
- Enable users to transmit and receive data set types that TRANSMIT and RECEIVE do not support
- Notify the sender when an acknowledgment is available to receive
- Delete transmissions from the JES spool
- Collect and report on statistics related to network usage

If you add the TRANSMIT and RECEIVE commands to the table of authorized commands, the TSO/E default allows all users to use those commands. By writing exits, you can limit the use of TRANSMIT and RECEIVE to specific users. You can also use exits to restrict the nodes to which a user can send transmitted data, to allow users to receive data sets intended for other users, and to allow users to transmit and receive data set types that TRANSMIT and RECEIVE do not support.

TSO/E allows you to decide on an installation-wide basis whether transmitted data will be encrypted. You can require all transmitted data to be encrypted, you can allow the user to decide whether to encrypt data, or you can disallow encryption altogether. If you allow encryption, you can use TRANSMIT and RECEIVE exits to selectively prevent encryption or decryption. You can also use exits to modify the encryption and decryption options the user specifies, or to use your own encryption algorithm instead of the one TSO/E invokes by default. By default, TSO/E users are not notified when they have a transmission to receive. You can use exits to provide that notification. JES2 and JES3 provide exits for notifying the receiver that a transmission has arrived. JES2, JES3, and TSO/E all provide exits for notifying the sender that an acknowledgment is available to receive.

Periodically you might need to delete from the JES spool those data sets that have been sent to incorrect user IDs, or to users who have not logged on for a given period of time. JES2 and JES3 provide exits you can use for that purpose.

TSO/E enables you to collect statistics on network usage and to report those statistics. Two TSO/E exits that you can write collect the statistics. SMF macros, that you code in those exits, report the statistics.

For more information, see Chapter 39, “Customizing TRANSMIT and RECEIVE,” on page 395.

TSOLIB command

TSO/E provides the TSOLIB command, which lets users dynamically link load module libraries of their choice from within their TSO/E sessions. TSO/E also provides initialization and termination exits that you can use to customize the TSOLIB command. The initialization exit can restrict users from using the TSOLIB command, can change operands supplied by the user, or pass the address of a new command buffer. The termination exit can perform clean-up processing, such as releasing storage obtained by the initialization exit. For more information, see Chapter 40, “Customizing the TSOLIB command,” on page 487.

Job submission and output processing

IBM provides a number of facilities you can use to customize the way MVS processes jobs and job output. You can use these facilities to:

- Override the IBM-provided default job processing characteristics
- Monitor, change, or supplement the JCL users use to submit a job
- Cancel a job
- Tailor the way the TSO/E SUBMIT, CANCEL, OUTPUT, and STATUS commands work
- Delete, release, or reroute job output
- Submit jobs as a surrogate user

JES2 or JES3, depending on which you have, provides the default job processing characteristics. You can use JES initialization statements to change these defaults.

TSO/E, JES2, JES3, and SMF all provide exits that enable you to further customize job and output processing. TSO/E provides two exits, one for the SUBMIT command and one for the CANCEL, OUTPUT, and STATUS commands. TSO/E also provides default routines for these exits. The SUBMIT default exit allows users to use the SUBMIT command as they wish. The default exit for the CANCEL, OUTPUT, and STATUS commands allows users to examine the status of any job. It prohibits users, however, from using the CANCEL or OUTPUT commands on any job other than their own. For more information, see Chapter 34, “Customizing the SUBMIT command and job output processing,” on page 293.

Printing data sets

Users print data sets using either the ALLOCATE or PRINTDS commands or by invoking an application that uses the printer support service. You can customize how users print data sets in several ways.

TSO/E provides two exits for the PRINTDS command, an initialization exit and a termination exit. You can use the initialization exit to change, delete, or add operands to a PRINTDS command, to change the PRINTDS operands that have fixed default values, or to prohibit specific users from using the command. You use the termination exit to perform any clean-up that is required because of an action taken by the initialization exit. When users issue the ALLOCATE command to allocate a system output (SYSOUT) data set or issue the PRINTDS command, they can specify the OUTDES operand and the name of an output descriptor. The output descriptor defines printer-related information and simplifies the use of the commands. Without an output descriptor, users must provide printer-specific operands. You can define output descriptors by including OUTPUT JCL statements in the user's logon procedure. If you have JES2 installed, you and your installation's users can also define output descriptors using the TSO/E OUTDES command.

To use the printer support service to print a data set, a user simply invokes an installation-written application, and selects a printer and print options from a display panel. There is no need for the user to code JCL statements or issue the ALLOCATE or PRINTDS commands. To make printer support available, the Information Center Facility must be available. You must define your printers using Information Center Facility panels and provide the necessary application programs that invoke printer support CLISTs.

With RACF installed, your installation can use security labels. If your installation is using security labels, the security label is printed on each page of output. Printing of the security label can be overridden with the correct RACF access authority.

For more information, see the following documents or topic:

- For a description of the PRINTDS, ALLOCATE, and OUTDES commands, see *z/OS TSO/E Command Reference*.
- For an explanation of security controls on printed output, see *z/OS Security Server RACF Security Administrator's Guide*.
- For an explanation of how to customize the way users print data sets, see Chapter 35, "Customizing how users print data sets," on page 315.

CLIST and REXX processing

TSO/E provides exits that you can use to customize the processing of REXX execs and CLISTs.

Customizing CLISTs

TSO/E provides two exits you can use to customize CLIST processing. One exit is for installation-written built-in functions, and the other is for installation-written statements.

The exit for installation-written built-in functions enables you to define and process your own built-in functions. The exit receives control each time TSO/E encounters a CLIST built-in function that it does not recognize.

The exit for installation-written statements enables you to define and process your own CLIST statements. You can also use this exit to do your own processing of TSO/E commands. This exit receives control each time TSO/E encounters a TSO/E command or an unrecognized statement in a CLIST.

For more information, see Part 6, "Customizing CLIST and REXX processing," on page 491.

Customizing REXX processing

TSO/E provides load modules, exits, and replaceable routines that you can use to customize REXX processing.

For REXX, TSO/E provides three default parameters modules that are load modules containing the default values for initializing language processor environments. A language processor environment is the environment in which a REXX exec executes. The parameter modules are:

- IRXTSPRM for TSO/E
- IRXISPRM for ISPF
- IRXPARMS for MVS.

You can change the default values used to initialize a language processor environment by providing your own parameters module. For more information, see "Characteristics for a language processor environment" on page 504.

The following describes the REXX exits:

- Pre-environment initialization exit (IRXINITX)

  IRXINITX performs processing before a new language processor environment is initialized. Use it to prevent the initialization of a language processor environment, change parameters used for initializing a language processor environment, or perform special pre-environment initialization processing.
- Post-environment initialization exit (IRXITTS or IRXITMV)

IRXITTS performs processing after the new language processor environment is initialized for an environment integrated with TSO/E. Use it to perform special processing for a language processor environment initialized with the parameters of the IRXINITX exit.

IRXITMV performs processing after a new language processor environment is initialized for an environment not integrated with TSO/E. Use it to perform special processing for a language processor environment initialized with the parameters of the IRXINITX exit.

- Environment termination exit (IRXTERMX)

  IRXTERMX performs processing before a language processor environment is terminated. Use IRXTERMX to prevent the termination of a language processor environment or perform special processing with the parameters of the IRXINITX exit.
- Exec processing exit

  Use the exec processing exit to prevent the execution of a REXX exec or perform special processing before a REXX exec is executed.
- Exec initialization exit

  Use this exit to access and update REXX variables before the first clause in the exec is processed.
- Exec termination exit

  Use this exit to access and update REXX variables before the REXX variable pool is terminated.
- Attention handling exit

  Use the attention handling exit to perform special attention processing. This exit can only be used for an environment integrated with TSO/E.

For information about the REXX exits, see Part 6, "Customizing CLIST and REXX processing," on page 491.

In addition to the REXX exits, TSO/E supplies replaceable routines you can use to customize REXX processing. TSO/E provides the following routines that handle different types of system services, which you can replace:
- Loading and freeing execs
- Reading and writing output
- Handling data stack services
- Obtaining and freeing storage
- Obtaining the user or terminal ID
- Determining whether the message ID is displayed with a REXX error message
- Handling host commands for a specific language processor environment

For information about the replaceable routines, see "Replaceable routines" on page 513.

Session Manager

To enable users to use Session Manager, you must make several modifications. The user's TSO/E logon procedure must invoke a program that attaches the Session Manager initialization task. Therefore, you must either change the existing logon procedure or write a new one. You may have to modify the message control program and the message handler. You may also have to modify initialization parameters in a SYS1.PARMLIB member, TSOKEY00 or its alternate.

For information about how to make Session Manager available to your users, see Chapter 45, “Setting up a Session Manager environment,” on page 559.

There are several ways you can customize Session Manager. You can customize the IBM-provided environment definition, you can provide your own environment definitions, and you can write Session Manager exits. The environment definition defines the user's screen layout, program function key definitions, and stream defaults. You can use the IBM-provided definition as is, or you can modify the IBM-provided definition. You can also provide multiple definitions so different users can use different definitions. Multiple definitions allow you to customize the environment definition to individual user needs.

Three Session Manager exits allow you to monitor a user's interaction with Session Manager. You can write an initialization exit, a stream monitoring exit, and a termination exit. IBM provides no defaults for these exits.

For information about defining and customizing Session Manager, see Part 8, “Session Manager,” on page 557.

Information Center Facility

Before the administrator can use the Information Center Facility, you must allocate the required libraries. In addition, you must provide the administrator with a logon procedure that accesses the Information Center Facility. If you are upgrading to a new level of TSO/E, you may have to convert Information Center Facility libraries and tables to be compatible with the new release level. You also have the option to move the programs that provide dialog functions from SYS1.CMDLIB to a different library.

After the Information Center Facility is made available, you can customize it to fit the needs of your installation. You can:

- Add your own applications and the panels for accessing the applications
- Tailor application definitions for specific groups of users or for individual users
- Modify the start-up and termination process
- Add commands to the list of commands that users can issue from Information Center Facility panels
- Change the default values of Information Center Facility variables
- Write exits for ADRS-II, the names service, or Application Manager

For more information, see Part 9, “Information Center Facility.,” on page 583.

Chapter 2. Writing exit routines

TSO/E provides exit points for many TSO/E functions and commands. An exit point is a specific point in a function's or command's processing where the function or command invokes an exit routine if one exists. You write an exit routine to perform special processing and customize how the function or command works. When your exit routine completes its processing, it returns control to the function or command that invoked the routine.

Exit routines let you change default values or actions or extend a TSO/E function. You can write an exit routine to change default processing, change the information a user specifies on a command, or extend the processing capabilities of a TSO/E command or function. TSO/E commands base their processing on input that users enter when they issue the command. You can write an exit routine to change the operands the user specified and therefore change the command's processing. You may also want to restrict users from using certain operands of a command.

For some functions or commands, TSO/E provides a default exit routine. If you do not write your own exit routine, the function or command invokes the default routine. If you do write your own exit routine, you replace the routine that TSO/E provides with your own exit routine. The function or command then invokes your exit rather than the default exit TSO/E provides.

For other functions or commands, TSO/E does not provide a default exit routine. If you do not write your own exit routine, the function or command simply continues processing without invoking an exit routine. If you do write an exit routine, the function or command invokes your routine.

The individual descriptions of the exits in this document describe whether TSO/E provides a default exit routine for the particular exit.

Overview of exits that TSO/E provides

The following sections give an overview on the exits that TSO/E provides for functions and commands. Also shown are the exit names if they must follow a naming convention.

Many TSO/E exits receive a *TSO/E standard exit parameter list*. Other TSO/E exits receive parameter lists that are different from the standard parameter list and which vary for each exit. To distinguish the exits that receive the standard exit parameter list from exits that receive different parameter lists, this document refers to TSO/E exits that receive the standard list as *standard-format exits*. "TSO/E standard-format exits" on page 32 describes standard-format exits and the standard exit parameter list. In the following sections the standard-format exits are designated by a footnote[1] following the name of the exit.

The individual descriptions of the exits in this document describe the parameter list each exit receives. The descriptions also describe whether TSO/E provides a

1. Name of standard-format exit.

default exit routine for the exit and if so, what processing the default exit routine performs. TSO/E does not provide default exit routines for the standard-format exits.

Exits for TRANSMIT and RECEIVE commands

Use the various TRANSMIT and RECEIVE exits together to perform different types of processing. Use the exits to monitor transmission activity or customize how TRANSMIT and RECEIVE operate. For example, you can use the exits to:

- Collect statistics about network usage
- Control which users can use a particular network path
- Support data sets that the Interactive Data Transmission Facility does not support
- Tailor encryption and decryption processing

Refer to Chapter 39, "Customizing TRANSMIT and RECEIVE," on page 395.

Note: If you have a standard-format exit and a "non-standard" exit installed for the same TRANSMIT or RECEIVE function, the command processor invokes only the standard-format exit. For example, suppose you have two TRANSMIT start-up exits INMXZ01 and INMXZ01R. TRANSMIT invokes only INMXZ01R. The INMXZ01 exit does not receive control.

- Common exit:
  - NAMES data set pre-allocation: INMCZ21R[1]
- TRANSMIT exits:
  - Start-up: INMXZ01, INMXZ01R[1]
  - Log data set pre-allocation: INMXZ21R[1]
  - Encryption: INMXZ03, INMXZ03R[1]
  - Termination: INMXZ02, INMXZ02R[1]
- RECEIVE exits:
  - Initialization: INMRZ01, INMRZ01R[1]
  - Log data set pre-allocation: INMRZ21R[1]
  - Data set pre-processing: INMRZ11, INMRZ11R[1]
  - Data set decryption: INMRZ13, INMRZ13R[1]
  - Notification: INMRZ04, INMRZ04R[1]
  - Acknowledgment notification: INMRZ05R[1]
  - Pre-acknowledgment notification: INMRZ06R[1]
  - Data set post-processing: INMRZ12, INMRZ12R[1]
  - Post-prompt: INMRZ15R[1]
  - Termination:INMRZ02, INMRZ02R[1]

Exits for logon and logoff processing

- Logon pre-prompt: IKJEFLD

  Tailor the TSO/E logon process. Verify, change, or supply logon parameters and system characteristics, cancel logon requests, provide your own JCL statements, or display your own full-screen logon panel.
- Logon pre-prompt: IKJEFLD1[1]

  Perform the functions of IKJEFLD, plus authorized functions. Specify the first TSO/E command, return Job and SYSOUT classes, bypass RACF, return the RBA, pass data to the logoff exit, specify SECLABEL of current logon session, set up a console profile, and set up languages for users. Refer to "Writing a logon pre-prompt exit (IKJEFLD/IKJEFLD1)" on page 90.

- Logon pre-display: IKJEFLN1

  Edit logon information prior to display or re-display of the logon panel. Process installation-defined fields on the logon panel. Refer to "Writing a logon pre-display exit (IKJEFLN1)" on page 112.
- Logon post-display: IKJEFLN2

  Validate and process fields on the logon panel after the user has entered the information on the logon panel. Request logon panel reprompting. Request help panels. Refer to "Writing a logon post-display exit (IKJEFLN2)" on page 124.
- Logon post-prompt: IKJEFLD3

  Edit TSO/E logon JCL and control blocks after prompting has completed. Refer to "Writing a logon post-prompt exit (IKJEFLD3)" on page 134.
- Logoff: IKJEFLD2[1]

  Tailor the TSO/E logoff process. Gather accounting information, control UADS and RACF data base updates, control re-logons. Refer to "Writing a logoff exit (IKJEFLD2)" on page 137.

Exit for OUTPUT, STATUS, and CANCEL commands

- IKJEFF53:

  Tailor the way users can handle the processing of batch jobs and their output.

Refer to "Writing an exit for the OUTPUT, STATUS, and CANCEL commands" on page 306.

Exit for SUBMIT command

- IKJEFF10:

  Check submitted JCL statements and accept, reject, or modify them.

Refer to "Writing an exit for the SUBMIT command" on page 296.

Exit for EDIT command

- Syntax checkers:

  Write an exit for syntax checkers that your installation provides. The exit fills in the option word with information the user specifies on the EDIT command. Refer to "Writing an exit for syntax checkers" on page 263.
- RENUM, MOVE, and COPY subcommands:

  Customize how the subcommands handle line numbering whenever a user issues the subcommands. Refer to "Writing an exit for the RENUM, MOVE, and COPY subcommands" on page 265.

Exits for Session Manager

- Initialization:

  Indicate which streams you want to monitor and whether the Session Manager should log line mode output while users are executing full-screen programs.
- Stream monitoring:

  Monitor the individual streams you specified in the initialization exit and perform required processing.
- Termination:

Perform special processing before the Session Manager ends.

Refer to "Writing Session Manager exits" on page 571.

Exits for Information Center Facility

- ADRS:

  Add processing, such as displaying panels or allocating data sets, whenever a user selects the ADRS option from the Information Center Facility. Refer to "Writing an exit for ADRS" on page 642.
- Names service:

  Keep track of changes that are made to the private and master directories whenever Information Center Facility users use the names service. Refer to "Writing an exit for the ICF names service" on page 643.

Exits for ALLOCATE command

- Initialization: IKJEFD47

  Check and change the command users issue or provide pseudo-operands.
- Termination: IKJEFD49

  Perform clean-up processing. Specify an alternative return code.

Refer to "Writing exits for the ALLOCATE command" on page 223.

Exits for Application Manager

- Function pre-initialization: ICQAMFX1[1]

  Check a user's authorization to use an application, allocate data sets for an application, and prepare to gather accounting data.
- Function post-termination: ICQAMFX2[1]

  Free data sets the function pre-initialization exit allocated and summarize accounting data.
- Panel pre-display: ICQAMPX1[1]

  Set default values for the panel to be displayed.
- Panel post-display: ICQAMPX2[1]

  Validate the information the user entered on the panel.

Refer to "Writing exits for Application Manager" on page 645.

Exits for CLIST processing

- Built-in functions: IKJCT44B[1]

  Add installation-written built-in functions.
- Statements: IKJCT44S[1]

  Add installation-written CLIST statements.

Refer to Chapter 41, "Customizing CLIST processing," on page 495.

Exits for REXX processing

- Pre-environment initialization: IRXINITX

  Perform processing before a new language processor environment is initialized.
- Post-environment initialization: IRXITTS or IRXITMV

Perform processing after a language processor environment is initialized.
- Environment termination: IRXTERMX

  Perform processing before a language processor environment is terminated.
- Exec processing:

  Perform special processing before a REXX exec is executed.
- Attention handling:

  Perform special attention processing in an environment integrated with TSO/E.
- Exec initialization:

  Access or update REXX variables.
- Exec termination:

  Access or update REXX variables.

Refer to Chapter 42, "Customizing REXX processing," on page 503.

Exits for EXEC command

- Initialization: IKJCT43I[1]

  Change the command the user issues.
- Termination: IKJCT43T[1]

  Perform clean-up processing and set the defaults for the control characteristics of the CLISTs or REXX execs.

Refer to Chapter 31, "Customizing the EXEC command," on page 277.

Exits for ALTLIB command

- Initialization: IKJADINI[1]

  Change the command the user issues.
- Termination: IKJADTER[1]

  Perform clean-up processing.

Refer to Chapter 28, "Customizing the ALTLIB command," on page 229.

Exits for PARMLIB command

- Initialization: IKJPRMX1[1]

  Change the command the user issues.
- Termination: IKJPRMX2[1]

  Perform clean-up processing.

Refer to "Writing exits for the PARMLIB command" on page 287.

Exits for TEST command

- Initialization: IKJEGMIE[1]

  Change the command the user issues.
- Termination: IKJEGMTE[1]

  Perform clean-up processing.
- Subcommand initialization: IKJEGCIE[1]

  Change the subcommand the user issues.
- Subcommand termination: IKJEGCTE[1]

Perform clean-up processing.

Refer to “Writing exits for the TEST command” on page 372.

Exits for TESTAUTH command

- Initialization: IKJEGAUI[1]
  Perform security verification.
- Termination: IKJEGAUT[1]
  Perform clean-up processing.
- Subcommand initialization: IKJEGASI[1]
  Change the subcommand the user issues.
- Subcommand termination: IKJEGAST[1]
  Perform clean-up processing.

Refer to “Writing exits for the TESTAUTH command” on page 384.

Exits for FREE command

- Initialization: IKJEFD21
  Check and change the command users issue or provide pseudo-operands.
- Termination: IKJEFD22
  Perform clean-up processing.

Refer to Chapter 32, “Customizing the FREE command,” on page 283.

Exits for LISTBC command

- Initialization: IKJEESX5[1]
  Initialize the environment for later exits, restrict users from using the command, or change the operands a user specifies on the command.
- Pre-display: IKJEESX6[1]
  If using individual user logs, provide special formatting, append diagnostic information to a message, and support special features of output devices.
- Pre-list: IKJEESX7[1]
  If using individual user logs, modify user log data set names and prepare for the pre-read exit.
- Pre-read: IKJEESX8[1]
  If using individual user logs, tailor I/O.
- Pre-allocate: IKJEESX9[1]
  If using individual user logs, allocate the user log data set.
- Failure: IKJEESXA[1]
  If using individual user logs, perform failure processing and clean-up after an I/O failure.
- Termination: IKJEESXB[1]
  Perform clean-up or special termination processing.

Refer to “Writing exits for the SEND, OPERATOR SEND, and LISTBC commands” on page 341.

Exits for OUTDES command

- Initialization: IKJEFY11

  Check and change the command users issue or provide pseudo-operands.
- Termination: IKJEFY12

  Perform clean-up processing.

Refer to "Writing exits for the OUTDES command" on page 319.

Exits for PRINTDS command

- Initialization: IKJEFY60[1]

  Tailor the fixed default values for specific operands. Restrict users from using the command or change the operands a user specifies on the command.
- Termination: IKJEFY64[1]

  Perform clean-up processing.

Refer to "Writing exits for the PRINTDS command" on page 323.

Exits for SEND command and OPERATOR SEND subcommand

- Initialization: IKJEESX0[1], IEEVSNX0[1]

  Initialize the environment for later exits, change the defaults in SYS1.PARMLIB, restrict users from using the command, provide different user log data set names, and reroute messages by changing the user ID of the target user.
- Pre-display: IKJEESX1[1], IEEVSNX1[1]

  If using individual user logs, provide special formatting, add diagnostic information, and support special features of output devices.
- Pre-save: IKJEESX2[1], IEEVSNX2[1]

  If using individual user logs, override the user log data set name, support special I/O, and perform open and write operations. If used with the LISTBC pre-read exit, process the message or add information to it such as a sequence number, compress the message, and change parameters.
- Failure: IKJEESX3[1], IEEVSNX3[1]

  If using individual user logs, perform failure processing and clean-up after an I/O failure.
- Termination: IKJEESX4[1], IEEVSNX4[1]

  Perform clean-up or special termination processing.

Refer to "Writing exits for the SEND, OPERATOR SEND, and LISTBC commands" on page 341.

Exits for CONSOLE command

- Pre-parse: IKJCNXPP[1]

  Check and, if necessary, change the command the user issues.
- Activation: IKJCNXAC[1]

  Establish a communication area, end a console activation request, change settings specified by the user, and grant or deny CONSOLE command authority to the user.
- 80% message capacity: IKJCNX50[1]

  Take action when the solicited or unsolicited message table becomes 80% full.

- 100% message capacity: IKJCNX64[1]

  Take action when the solicited or unsolicited message table becomes 100% full.
- Deactivation: IKJCNXDE[1]

  Perform clean-up processing.

Refer to "Writing exits for the CONSOLE command" on page 235.

Exits for CONSPROF command

- Initialization: IKJCNXCI[1]

  Check and, if necessary, change the command the user issues, update the console profile with an installation portion, and grant or deny CONSOLE command authority to the user.
- Pre-display: IKJCNXCD[1]

  Add information to the console profile display message, IKJ55351I, or issue an installation-defined message instead of IKJ55351I.
- Termination: IKJCNXCT[1]

  Perform clean-up processing and store the installation portion of the console profile in a permanent place.

Refer to "Writing exits for the CONSPROF command" on page 247.

Exit for PUTGET and GETLINE processing

- IKJEFXG1:

  Tailor PUTGET and GETLINE processing.

Refer to Chapter 19, "Customizing PUTGET and GETLINE processing," on page 181.

Exits for TSOLIB command

- Initialization: IDYTSINI[1]

  Change the command the user issues.
- Termination: IDYTSTER[1]

  Perform clean-up processing.

Refer to Chapter 40, "Customizing the TSOLIB command," on page 487.

General programming considerations

An exit routine must follow standard linkage conventions. Whenever an exit receives control, the contents of the general registers are:

Register 0
: Unpredictable

Register 1
: Address of the parameter list

Registers 2–12
: Unpredictable

Register 13
: Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

When an exit receives control, it must save the caller's registers. Register 13 contains the address of the register save area. Before the exit returns control to the invoking program, it must restore the caller's registers and set a return code in register 15, if the exit supports return codes.

When an exit receives control, register 1 contains the address of a parameter list. The parameter list provides information the exit needs to perform its processing. Exits also use the parameter list to return information to the invoking program.

All exits, except the logon pre-prompt exit IKJEFLD and the three Session Manager exits, support the use of return codes. The exit routine passes back the return code to the invoking program in register 15. Return codes indicate what action the invoking program should take. For example, a return code may indicate that the exit routine's processing was successful and that the invoking program should continue processing. A different return code may indicate that the invoking program should terminate processing.

If the invoking program expects specific values for a return code and the exit returns a value that the invoking program does not recognize, the invoking program either:
- Issues an error message and terminates processing, or
- Terminates processing without displaying an error message to the user.

The standard-format exits support a standard set of return codes. "Standard return codes" on page 39 describes the return codes for these exits.

When you write an exit routine, there are several things you must consider:
- The environment of the invoking program because the exit routine runs as an extension of that environment. This includes such things as AMODE, RMODE, state, and key.
- Programming considerations, including whether the exit should be reentrant, refreshable, and reusable.
- Error conditions. The exit routine must handle its own error conditions and provide its own error recovery.
- Testing. You should test your exit before you integrate it into the system. You can easily test exits using a logon STEPLIB. Testing exits in this way does not affect other user's work on the system. You must ensure that the STEPLIB has the correct authorization to perform the testing.

The TSO/E functions and command processors may encounter errors when they invoke an exit or while an exit is processing. The function or command processor then displays an error message to the user. *z/OS TSO/E Messages* lists the messages that TSO/E displays.

The description of each exit in this document includes details about the registers on entry and return, the return codes, parameter lists, and environment considerations. "Format of the exit descriptions" on page 41 highlights the format that this document uses to describe each individual exit.

TSO/E standard-format exits

Many TSO/E exits receive a *TSO/E standard exit parameter list* and are referred to in this document as standard-format exits. "TSO/E standard exit parameter list" describes the standard parameter list for these exits.

If TSO/E provides a standard-format exit for a function or command processor, TSO/E provides both an initialization exit and a termination exit. For some TSO/E commands, TSO/E provides other exits in addition to an initialization and termination exit. For example, for the SEND command, TSO/E provides the following exits:
- Initialization
- Pre-display
- Pre-save
- Failure
- Termination.

See "Overview of exits that TSO/E provides" on page 23 for a lists all of the exits that TSO/E provides.

You use the initialization exit to receive control at the beginning of the execution of a specific command or function and to change the default processing or extend processing capability. The initialization exit for a TSO/E command processor receives control before the command processor invokes the parse service routine to parse the command. You can use initialization exits to change the operands a user specifies on a command and to perform other processing to customize a command's or a function's processing.

Termination exits receive control just before a function's or command processor's completion. You can use termination exits for clean-up processing. If the initialization exit or other exits for a command or function obtain storage, you must write a termination exit to free the storage.

TSO/E provides an example exit (IKJEXIT) in SYS1.SAMPLIB that uses the standard exit parameter list. The example may be helpful in writing exit routines for standard-format exits. "Example installation exit" on page 40 describes the example.

TSO/E standard exit parameter list

The TSO/E standard exit parameter list is a standard set of data that all of the standard-format exits receive. Register 1 points to a list of addresses. Each address points to a *parameter entry* that has a key, length, and data format. The first nine parameter entries are standard for all of the standard-format exits.

Following the standard set of data (the first nine parameter entries), an exit can also receive *exit-dependent* data. The end of the parameter list is indicated by the high-order bit being on in the parameter list pointed to by register 1.

Figure 1 on page 33 shows the standard exit parameter list that the standard-format exits receive. The figure also illustrates that an exit can receive exit-dependent data starting at offset +36 (decimal) in the parameter list. In Figure 1 on page 33, the offsets for the parameter entries are shown in decimal. The key, length, and data fields for each parameter entry are in hexadecimal notation.

"Parameter entries" on page 34 describes the parameter entries and the key, length, and data fields.

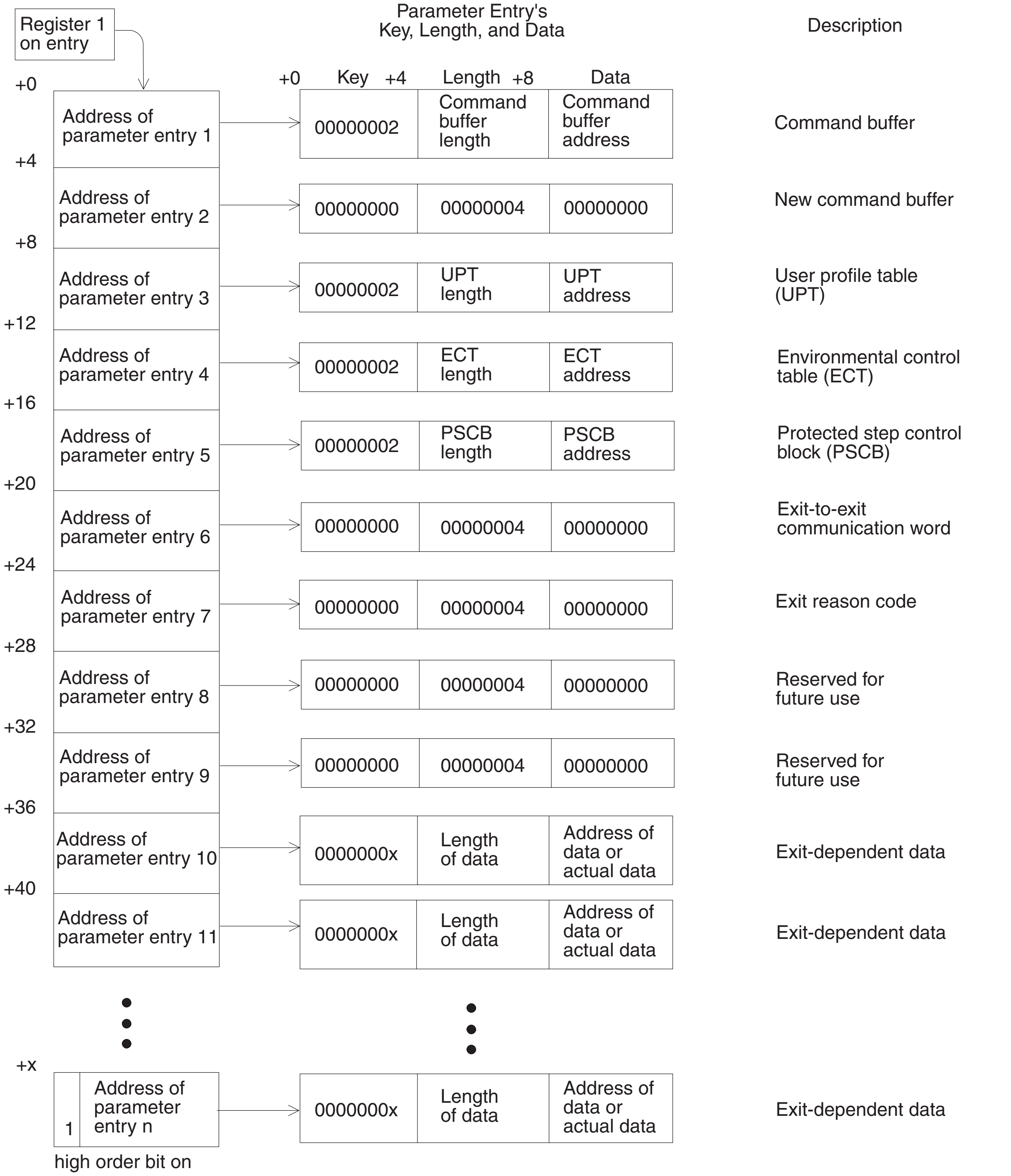


Figure 1. Format of the TSO/E standard exit parameter list and exit-dependent data for TSO/E Standard-Format Exits

Parameter entries

Data is passed as parameter entries in a key, length, and data format. Figure 1 on page 33 shows that Register 1 points to a list of addresses. Each address in the list points to a parameter entry.

The three parameter entry fields (key, length, and data) are described below:

Key
The key field is one fullword that contains a hexadecimal value. Table 1 shows the keys that have been defined for communication between the exit and the invoking program. Any other values are reserved and must not be used.

Table 1. Definition of keys for standard exit parameter list

KEY	Description
X'00'	No data is passed in this parameter entry.
X'01'	The data field contains the actual data for the parameter entry.
X'02'	The data field contains the address of the actual data for the parameter entry.
X'03'	This is a special key that only certain standard-format exits support. The exit requests that the function or command processor that invoked the exit routine use the exit reason code as the function's or command processor's return code. A key of X'03' is valid only for the exit reason code parameter entry (parameter entry 7). For more information, see "Exit reason code" on page 38.
X'04'	This is a special key that only certain standard-format exits support. The field is validated. This key indicates that the field contains valid data. When this key is set, the field is locked and cannot be altered.

Length
The length field is one fullword that contains the length of the data, in hexadecimal. For a key of X'01', the length is the length of the data in the data field. For a key of X'02', the length is the length of the data that the address in the data field points to.

The length field cannot be X'00'. If no data is passed in a parameter entry (key of X'00'), the length field must contain X'04'. If you use a key of X'03', the length field must also contain X'04'.

Data
The data field contains either the address of the data, the actual data, or a value of X'00'.

The value in the data field depends on the value in the key field. The following shows the relationship between the key and the value in the data field:

```
Key         Data Field

X'00'          X'00'
X'01'          Actual data
X'02'          Address of the data
X'03'          Exit reason code
X'04'          Field is validated
```

Figure 1 on page 33 shows the values of the parameter entries that the first exit for a function or command processor receives. The first exit is usually the "initialization" exit for the function or command. The individual descriptions of each exit in this document describe any exceptions to the standard exit parameter list.

If information about a particular parameter is unavailable to a function or command processor, the function or command processor passes the following for the parameter entry:

Key	X'00'
Length	X'04'
Data	X'00'

An exit can change only the following parameter entries in the standard exit parameter list:
- New command buffer.
- Exit-to-exit communication word.

Depending on the individual exit, the exit may be able to change exit-dependent data that it receives. The individual descriptions of the exits in this document describe any exit-dependent data and whether the exit can change the parameter entries for this data. If the exit changes any of the other parameter entries, unpredictable results can occur.

If an exit changes the new command buffer, the exit-to-exit communication word, or any exit-dependent data, the function or command processor passes the changes to the next exit it invokes.

The following topics describe each of the standard parameter entries and the exit-dependent data.

Command buffer

Figure 2 shows the format of the command buffer.

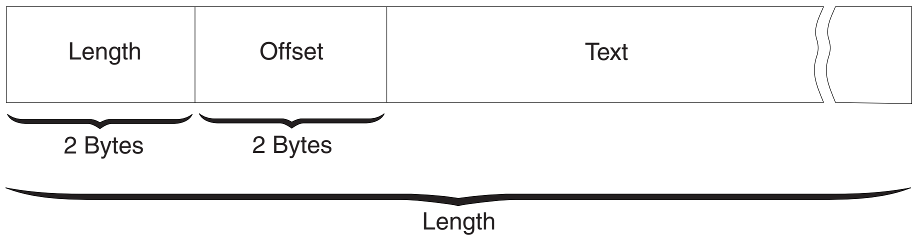


Figure 2. Format of the command buffer

When a command processor gets control, the command buffer contains:
- A 4-byte header that consists of:
  - A 2-byte length field that contains the length of the command buffer. The length includes the 4-byte header.
  - A 2-byte offset field. The content of the offset field depends on whether the user specifies operands on the command and when the exit routine receives control:
    - If the user specifies operands on the command and the exit routine receives control before the parse service routine, the offset field contains the number of text bytes that precede the first operand.
    - If the user does not specify any operands on the command or the exit routine receives control after the parse service routine, the offset field contains the length of the text field of the command buffer.

- A text field that contains the command name followed by any operands that the user specifies on the command.

In the standard exit parameter list, the length field of the command buffer (parameter entry 1) is the same value as the value in the 2-byte length field of the command buffer. This length includes the 4-byte header in the command buffer.

For example, suppose the user issues the following command:

```
PRINTDS DATASET(TEST.DATA) CLASS(B)
```

In the standard exit parameter list, the length field for the command buffer would be X'23'. In the command buffer itself, the:
- 2-byte length field would also be X'23'
- 2-byte offset field would be X'08'
- Text field would contain the command the user issues:
  ```
  PRINTDS DATASET(TEST.DATA) CLASS(B)
  ```

An exception is the command buffer for the SEND subcommand of the OPERATOR command. The command buffer does not contain the four-byte header field. It contains only the text field.

Some initialization exits for commands receive control before the command processor invokes the parse service routine to check the syntax of the command and convert operands to uppercase characters. Therefore, when these initialization exits receive the command buffer:
- The command the user issued may not be syntactically correct, and
- The command and operands are passed exactly the same way the user specified them. That is, they may be all uppercase, all lowercase, or mixed case.

These exits must not change the command buffer they receive. These exits must use the new command buffer (parameter entry 2) to return an updated command buffer to the invoking command processor. For more information about returning a new command buffer, see "New command buffer."

New command buffer

The new command buffer allows an initialization exit to change the command that a user issues. Figure 2 on page 35 shows the format of the command buffer.

To change the command buffer, the exit must:
- Obtain storage for a new command buffer
- Build the new command buffer. When you build the new command buffer, make sure the 4-byte header of the command buffer contains the correct length and offset values.
- Update the key, length, and data fields in the parameter list for the new command buffer as follows:

  Key X'02'

  Length
  the length of the new command buffer

  Data the address of the new command buffer

If the exit returns a new command buffer, the command processor uses the new command buffer for the remainder of its processing. The command processor passes the new command buffer to all subsequent exits that it invokes.

You must also provide a termination exit to free the storage that the initialization exit obtained for the new command buffer.

User profile table (UPT)

The exit receives the address of the user profile table (UPT), which allows you to use the TSO/E service routines. For more information about the service routines, see *z/OS TSO/E Programming Services*.

Environmental control table (ECT)

The exit receives the address of the environmental control table (ECT), which allows you to use the TSO/E service routines. For more information about the service routines, see *z/OS TSO/E Programming Services*.

Protected step control block (PSCB)

The exit receives the address of the Protected Step Control Block (PSCB), which allows you to use the TSO/E service routines. For more information about the service routines, see *z/OS TSO/E Programming Services*.

Exit-to-Exit communication word

You can use the exit-to-exit communication word to pass information between the different exits that a specific command processor or function invokes. The command processor or function passes the exit-to-exit communication word to each exit it invokes. The exit routines themselves are responsible for using and maintaining this parameter entry.

To return a value in the exit-to-exit communication word, update the parameter entry key, length, and data fields. If the exit returns the actual data in the data field, the exit updates the key, length, and data fields as follows:

Key	X'01'
Length	X'04'
Data	data

If the exit returns the address of the data in the data field, the exit updates the key, length, and data fields as follows:

Key	X'02'
Length	the length of the data
Data	the address of the data

For example, the SEND command processor may invoke five exits:
1. Initialization
2. Pre-display
3. Pre-save
4. Failure
5. Termination

If the initialization exit returns a value in the exit-to-exit communication word, SEND passes the value to the next exit it invokes, for example, the pre-display exit. If the pre-display exit changes the exit-to-exit communication word, SEND passes the new value to the next exit it invokes.

Exit reason code

Use the exit reason code parameter to return an exit reason code to the function or command processor.

If an exit sets a return code of 12, the command processor or function displays an error message to the user and then terminates processing. The error message indicates that the exit requested termination of the command or function. The message also contains an exit reason code. If the exit does not return an exit reason code, the command processor or function specifies a reason code of 0 in the error message. If the exit returns an exit reason code, the error message specifies the reason code that the exit returned. By using return code 12 and different exit reason codes, you can determine why a particular exit requested termination of a function or command.

To return an exit reason code, the exit must update the key and data fields for the exit reason code (parameter entry 7) as follows:

Key	X'01'
Data	the value of the exit reason code

For certain exits, you can request that the invoking program use the exit reason code as its return code by using a key of X'03'. Only some exits support a key of X'03'. The individual descriptions of each exit in this document describe whether you can use a key of X'03'.

To have the function or command processor use the exit reason code as its return code, the exit must update the key and data fields for the exit reason code (parameter entry 7) as follows:

Key	X'03'
Data	the value of the exit reason code

If you write several exits for a command and more than one exit indicates that the command should use the exit reason code as its return code, the command processor uses the reason code from the last exit it invokes. For example, suppose you write an initialization and a termination exit for the EXEC command. If both exits request that the exit reason code is used as the return code from EXEC, the EXEC command processor uses the reason code from the termination exit.

You must ensure that the exit reason code you want to use as a command's return code is not the same as an existing return code for the command processor. For a list of the TSO/E return codes for each command, see *z/OS TSO/E Command Reference*.

Reserved for TSO/E

Parameter entries 8 and 9 are reserved for TSO/E. Do not use these two parameter entries.

Exit-Dependent data

Starting at offset +36 (decimal) in the parameter list, the remaining parameter entries are for exit-dependent data. An individual function or command processor may pass additional data to an exit in these parameter entries. The individual descriptions of each exit in this document indicate whether the exit receives exit-dependent data.

The end of the parameter list is indicated by the high-order bit being on in the parameter list pointed to by register 1.

Standard return codes

Table 2 shows the standard return codes that the TSO/E standard-format exits support. Some standard-format exits support other return codes in addition to the three standard return codes. The individual descriptions of the exits in this document describe any additional return codes that each exit supports.

Table 2. Standard return codes for the TSO/E standard-format exits

Return code (decimal)	Description
0	Exit processing was successful. The function or command processor that invoked the exit continues processing.
12	Exit processing was unsuccessful. The function or command processor that invoked the exit issues an error message to the terminal user and then terminates processing. The error message indicates that the exit routine requested termination of the command or function. The error message that the function or command processor displays includes an exit reason code. For more information about using exit reason codes with return code 12, see "Exit reason code" on page 38.
16	Exit processing was unsuccessful. The function or command processor that invoked the exit terminates processing without issuing an error message to the terminal user.

If an exit returns with an undefined return code, the function or command processor terminates processing without displaying a message to the user.

Installing the standard-format exits

Installing the TSO/E standard-format exits is simpler than the installation procedures for other TSO/E exits that do not use the standard exit parameter list. All standard-format exits must have a specific name and be link-edited as separate load modules.

You can link-edit all of the standard-format exits in a separate load library that is exclusively for TSO/E exits or in an existing library that contains other routines. The standard-format exits can reside in:

- The link pack area (LPA), which makes them available to all of your users. If you change the exit and want to make the changes available in LPA for all of your users, you must re-IPL your system.
- Linklist (LNKLST), which makes them available to all of your users. You can also easily change the exit.
- A logon STEPLIB, which is helpful for limited use and for testing the exit before you integrate it into your system. In this case, you can change the exit. However, the use of a STEPLIB is not suggested for all of your users because of the extra search time required to locate and invoke the exit.

For more information about LPA, LNKLST, and STEPLIB, see *z/OS MVS Initialization and Tuning Guide*.

TSO/E determines the exact location of each exit (STEPLIB, LNKLST, or LPA). TSO/E then invokes the exit from that location and only from that location during that entire TSO/E session. If the location of the exit is changed, the change only goes into effect when a new TSO/E session is initiated. The search order is STEPLIB, LPA, and then LNKLST.

You can also install the exits using System Modification Program (SMP/E), which allows you to maintain a record of the exits you have installed. For more information about SMP/E, see the following publications:
- *SMP/E for z/OS Reference*
- *SMP/E for z/OS User's Guide*

If you use SMP/E, you must ensure that you generate your own functional module ID (FMID) that does not duplicate any of the FMIDs that IBM uses.

The individual descriptions of each exit in this document discuss how you install TSO/E exits that do not use the standard exit parameter list.

Example installation exit

TSO/E provides an example installation exit in SYS1.SAMPLIB. The member name of the example is IKJEXIT. IKJEXIT is an example of how you can write an exit routine only for the standard-format exits because it uses the standard exit parameter list.

IKJEXIT has the following attributes:
- Reusable
- Refreshable
- Reentrant
- AMODE(ANY)
- RMODE(31)

IKJEXIT can be either APF-authorized or not APF-authorized. APF-authorization depends on how you link-edit the exit.

You must assemble IKJEXIT with the High Level Assembler, or an assembler that provides equivalent function.

IKJEXIT performs the following:
- Identifies each parameter entry and associates the parameter entry it is currently processing with a hexadecimal value
- Displays the key, length, and data fields for each parameter entry

  For the key and length fields, the values IKJEXIT displays are the actual values contained in the fields. For the data field, IKJEXIT displays the address of the data and then lists the actual data.
- Detects any errors with the parameter entry key or parameter entry length fields and displays an appropriate informational message.

Figure 3 on page 41 shows the output that IKJEXIT displays for each parameter entry. In the figure, the:
- ZZ fields indicate the parameter entry that is displayed. For example, if the information is for the UPT (parameter entry 3), ZZ would be 03.

- X..X fields are fullwords that indicate the:
  - Address of where the parameter entry starts
  - Key for the parameter entry
  - Length of the data
  - Address of the data.

```
IKJEXIT

PARAMETER_ENTRY_ZZ IS AT -> X..X  (ADDRESS OF WHERE THE
                                  PARAMETER_ENTRY STARTS)

PE_ZZ KEY -> X..X LENGTH OF DATA -> X..X ADDRESS OF DATA -> X..X
DATA   +00000000  E3C8C9E2  40C9E240  E3C8C540  C4C1E3C1
       +00000010  40C6D6D9  40E3C8C9  E340C5E7  C1D4D7D3
       +00000020  C5
:

PARAMETER_ENTRY_ZZ IS AT -> X..X  (ADDRESS OF WHERE THE
                                  PARAMETER_ENTRY STARTS)

PE_ZZ KEY -> X..X LENGTH OF DATA -> X..X ADDRESS OF DATA -> X..X
DATA   +00000000  E3C8C9E2  40C9E240  E3C8C540  C4C1E3C1
       +00000010  40C6D6D9  40E3C8C9  E340C5E7  C1D4D7D3
       +00000020  C5
```

Figure 3. Output of the TSO/E example installation exit IKJEXIT

For more information, refer to the prologue of the example in SYS1.SAMPLIB(IKJEXIT).

Format of the exit descriptions

This section describes each of the individual exits using the same format. The format of the description is shown below.

Functional Description
: provides an overview of the function or command that the exit is for. It describes at what point in the function's or command's processing the exit receives control. It also highlights how you can use the exit.

TSO/E-Supplied Exit
: describes the processing of the default TSO/E exit routine if TSO/E provides one.

Entry Specifications
: describes the contents of the registers on entry.

Parameter Descriptions
: describes the contents of the parameter list that the exit receives from the invoking program.

Return Specifications
: describes what the registers must contain when the exit returns control to the invoking program. It also describes the possible return codes from the exit.

Programming Considerations
: describes various considerations for writing the exit. It contains information, such as whether the exit must be reentrant, refreshable, and reusable, APF-authorization, and the TSO/E service routines the exit can use. It also includes information about:

- Installing the exit, including:
  - Naming conventions, if any
  - Where the exit can reside (for example, STEPLIB, LPALIB)
  - Linkage considerations.
- Environment

  Includes the state and key that the exit executes in, AMODE and RMODE requirements, and other operational considerations.
- Restrictions and Limitations

  Lists any restrictions or limitations for the exit itself or any effect on other system processing because of exit processing.

Possible Uses
describes different tasks you can perform using the exit.

Part 2. Considerations for installing, migrating and activating the functions of TSO/E

Chapter 3. Considerations for installing TSO/E

This section describes planning considerations for installing TSO/E from a previous TSO/E release.

REXX parameter modules

If you have modified the parameter modules (IRXISPRM, IRXTSPRM, or IRXPARMS), you may want to copy them to another data set, because they are overlaid at install time.

Using the LINKPGM, ATTCHPGM, LINKMVS, and ATTCHMVS host command environments

If you have created your own REXX parameter modules for your installation or have tailored the parameter modules (IRXREXX1, IRXREXX2, and IRXREXX3 respectively TSOREXX1, TSOREXX2 and TSOREXX3 for TSO/E 2.5 and lower) supplied by IBM, you must add entries to the host command environment table defined in those modules, in order to access the new LINKPGM, ATTCHPGM, LINKMVS, and ATTCHMVS host command environments.

Installing the APPC/MVS administration dialog

The APPC/MVS administration dialog is shipped with the Information Center Facility. An installation file is shipped in AICQILIB and is provided for installing the APPC/MVS administration dialog on the Information Center Facility as an option on the main administrator panel, ICQADMIN. The APPC/MVS administration dialog can also be invoked as a REXX exec from an ISPF panel.

Allocating the user attributes data set (SYS1.UADS)

When TSO/E is installed, member IBMUSER0 will be placed in the data set pointed to by the UADS DDDEF or DD statement in the SMP/E procedure. If you have previously installed TSO/E, you should allocate a dummy user attributes data set (UADS) for installation. This will prevent SMP/E from putting IBMUSER0 back in your UADS data set (if you have deleted it), or replacing it with the default IBMUSER entry (if you use the IBMUSER0 user ID and have changed its attributes).

The dummy UADS data set may have any high-level qualifier, and need not be cataloged in the master catalog. It should be available for SMP/E processing, in case the IBMUSER0 default entry is changed, or other entries are added, by service.

If you are installing TSO/E for the first time, allocate a SYS1.UADS data set and catalog it in the master catalog.

Note:

1. If you are installing TSO/E for the first time and reallocate the UADS data set, or if you have deleted IBMUSER0 and allow SMP/E to replace it in your

production UADS data set during the installation of TSO/E you will need to use the SYNC command to re-synchronize the UADS and BRODCAST data sets.

2. SYS1.UADS must be allocated with LRECL=80 for installation. When the SMP/E installation is complete, it should be reallocated with LRECL=172. Any high level qualifier can be used for UADS when installing. However, when you IPL, a SYS1.UADS must exist, be cataloged, and contain at least one user.

Optimize the block size of SYS1.UADS to minimize waste of space. For more information about allocating the UADS, see Chapter 24, "Working with the UADS and broadcast data set," on page 199.

Migrating customized parts

Installation of TSO/E may replace exits, CSECTs or tables that you changed in the course of the customization of a previous release. If you have customized any of the parts mentioned in the following three sections, take the suggested action.

Deleted exit routines and CSECTs

Installing TSO/E replaces the following exit routines and CSECTs:

- SUBMIT exit, IKJEFF10
- OUTPUT/STATUS/CANCEL exit, IKJEFF53
- TRANSMIT and RECEIVE exits:
  - TRANSMIT startup exit (INMXZ01 or INMXZ01R)
  - TRANSMIT termination exit (INMXZ02 or INMXZ02R)
  - TRANSMIT encryption exit (INMXZ03 or INMXZ03R)
  - RECEIVE initialization exit (INMRZ01 or INMRZ01R)
  - RECEIVE termination exit (INMRZ02 or INMRZ02R)
  - RECEIVE notification exit (INMRZ04 or INMRZ04R)
  - RECEIVE data set pre-processing exit (INMRZ11 or INMRZ11R)
  - RECEIVE data set post-processing exit (INMRZ12 or INMRZ12R)
  - RECEIVE data set decryption exit (INMRZ13 or INMRZ13R)
- The TRANSMIT and RECEIVE CSECT INMXPARM
- The APF-authorized command and program CSECTs, IKJEFTE2, IKJEFTE8, and IKJEFTAP
- CSECT IKJEFTNS, which contains commands not supported in the background.

Installing TSO/E will delete the following exit routines and CSECTs, if they were installed using SMP/E and have the same FMID as the TSO/E version being deleted in the installation logic:

- TRANSMIT/RECEIVE NAMES data set pre-allocation exit (INMCZ21R)
- TRANSMIT log data set pre-allocation exit (INMXZ21R)
- RECEIVE acknowledgment notification exit (INMRZ05R)
- RECEIVE pre-acknowledgment notification exit (INMRZ06R)
- RECEIVE post-prompt exit (INMRZ15R)
- RECEIVE log data set pre-allocation exit (INMRZ21R)
- Session Manager exits ADFEXIT1, ADFEXIT2, and ADFEXIT3

Use SYS1.PARMLIB member IKJTSO00 (IKJTSOxx) instead of CSECTs INMXPARM, IKJEFTE2, IKJEFTE8, IKJEFTAP, and IKJEFTNS. For more information see Chapter 11, "Specifying authorized commands/programs, and commands not supported in the background," on page 151 and Chapter 13, "Setting up the TRANSMIT and RECEIVE environment," on page 161.

Either SYS1.PARMLIB member IKJTSOxx or CSECTs INMXPARM, IKJEFTE2, IKJEFTE8, IKJEFTAP, and IKJEFTNS must contain certain commands and programs for TSO/E to function properly.

Changed exit routines

Some exit routine requirements have changed since TSO/E 2.3:

- If you currently have a SUBMIT exit, IKJEFF10, that retrieves passwords from the TSB, you might need to change the SUBMIT exit. The password is no longer stored in the TSB. See "Writing an exit for the SUBMIT command" on page 296 for more information about the SUBMIT exit and for information about changing your logon pre-prompt exit to have the password stored in the TSB.

New information center facility tables

If you previously installed the Information Center Facility and you made changes to the tables listed below, you should save the tables in a temporary library, otherwise they will be deleted when you install JTE26D0. Also, plan to reinstall those parts that were customized by manual updates or usermods.

Table name	SMP/E system library
ICQABT00	ICQ.ICQTABLS
ICQADT00	ICQ.ICQTABLS
ICQAPL01	ICQ.ICQAPL
ICQSB001 through ICQSB021	ICQ.ICQABTXT

Using SYS1.PARMLIB member IKJTSOxx

See Chapter 11, "Specifying authorized commands/programs, and commands not supported in the background," on page 151 for more information about the usage of IKJTSOxx.

New TRANSREC NODESMF parameter

Since TSO/E 2.5 the TRANSREC NODESMF parameter accepts wild card notation for the *node/smfid* parameter. If this support is not documented in the version of *z/OS MVS Initialization and Tuning Reference* that you have received, browse SYS1.PARMLIB member IKJTSO00 and read the comments included in the text.

Migrating TSO/E commands

During the installation of TSO/E, SMP/E will link edit all TSO/E commands into the data set pointed to by the CMDLIB DDDEF or the CMDLIB DD statement in the SMP/E procedure. If you have previously copied any TSO/E commands to LPALIB, and do not identify them to SMP/E, they will not be deleted or replaced by the installation.

Add SYS1.CMDLIB to LNKLSTxx in SYS1.PARMLIB if:
- the TSO logon procedures do not specifically have a STEPLIB assigned to SYS1.CMDLIB, or
- the commands were not copied to SYS1.LPALIB.

If you have **not** previously copied TSO/E commands from CMDLIB to LPALIB, skip the rest of this section.

If you **have** previously copied TSO/E commands from CMDLIB to LPALIB, the old copies must be deleted from LPALIB or replaced with the new version of the commands. IBM suggests you to MLPA the commands or add CMDLIB to the LPA list rather than copy the commands to LPALIB.

Note: If you choose to add CMDLIB to the LPA list, it must also be added to the APF list.

However, if you choose to copy the commands to LPALIB, you should also identify this dual residency to SMP/E, so that future product and service installations do not cause problems.

Do not move TSO/E commands from CMDLIB; if the commands are copied to LPALIB, do not delete them from CMDLIB. However, if you choose to move modules to LPALIB from CMDLIB, this movement should be identified to SMP/E, so that future product and service installations do not cause problems.

You can use the SMP/E UCLIN command to identify dual residency, or movement, of the modules for TSO/E commands. For more information about UCLIN, see the *SMP/E for z/OS Reference*.

National language considerations

If you are currently using a multicultural support feature of TSO/E, it is deleted when you install TSO/E.

LOGON considerations

This section contains LOGON considerations when installing TSO/E.

Naming conventions for the load modules and CSECTs

The naming convention of the modules and CSECTs containing the definitions of the logon panels are:

- The first five characters are:
  - IKJLP for default logon panels
  - IKJLQ for logon panels with password phrase support
  - IKJLH for logon help panels
- The last three characters are the 3-character code for the language that the user has selected.

Your installation might have any number of these load modules. There is one load module for each language that your installation uses.

The mixed case U.S. English (ENU) and uppercase U.S. English (ENP) panels are supplied with TSO/E. Panels for other languages are supplied with the corresponding language feature.

Logon panel customization

With z/OS V1R10, new logon panels were added to allow for password phrases. If you customized your logon panels and plan to allow password phrases using the LOGON PASSPHRASE(ON) parmlib option, these new panels need to be customized as well. If password phrase support is active, IKJLQENU (mixed case U.S. English) or IKJLQENP (uppercase U.S. English) is used instead of IKJLPENU or IKJLPENP. After you complete your modifications to the logon panel modules IKJLPxxx, IKJLHxxx, or IKJLQxxx (where xxx is the 3-character language code), you must replace these modules in the LPA using SMP/E. Source code for the logon panel modules for mixed case U.S. English (IKJLPENU, IKJLQENU, and IKJLHENU) and uppercase U.S. English (IKJLPENP, IKJLQENU and IKJLHENP) can be found in SYS1.SAMPLIB.

Figure 4 shows the SMP/E job to move logon panels.

```
++USERMOD(UM99999) .
++VER(Z038) FMID(HTE23D2)
  /*
PROBLEM DESCRIPTION:
 MOVE LMODS IKJLQENU AND IKJLQENP FROM LINKLIB TO LPALIB
 TEMPORARILY.  EXPECT RC=4 DURING THE RECEIVE SINCE NO OBJECT
 DECKS ARE SUPPLIED.  EXPECT RC=0 DURING THE APPLY.  DO NOT
 ACCEPT THIS USERMOD.
  */ .
++MOVE(IKJLQENU) SYSLIB(LINKLIB) TOSYSLIB(LPALIB) LMOD.
++MOVE(IKJLQENP) SYSLIB(LINKLIB) TOSYSLIB(LPALIB) LMOD.
```

Figure 4. SMP/E job to move logon panels

Considerations for katakana devices

If using a Katakana device with the U.S. English language, set the language to English Uppercase (ENP). If lower case U.S. English characters are used on a Katakana device, the characters are Katakana characters and the text that is displayed is in uppercase English characters with some Katakana characters.

Using security labels

With Security Server (RACF), you can use security labels to protect system resources. Use Security Server (RACF) to define security labels for TSO/E users. When TSO/E users log on, the security label (SECLABEL) is associated with their TSO/E logon session. You must define all TSO/E users through Security Server (RACF) and create TSO segments for those users so that the system saves each user's SECLABEL from logon session to logon session.

You can also protect the security classification of messages that are sent from one user to another. Messages must be stored in the *logname.userid* user log (not the broadcast data set), where logname is the name specified on the LOGNAME operand of the SEND PARMLIB parameter and userid is the user's user ID. Users can only view their *logname.userid* user logs by using the TSO/E LISTBC command. Procedures to protect the user logs are described in Chapter 5, “Activating the functions of TSO/E,” on page 57.

Note: If you want to use security labels, do not activate the Information Center Facility. The Information Center Facility does not support the security enhancements.

TSO/E full-screen logon panel

If you have installed Security Server (RACF), the TSO/E full-screen logon panel has a SECLABEL field. This field allows users to specify a security label for their TSO/E session.

TSO/E logon pre-prompt exit

If you plan to use security labels and the logon pre-prompt exit (IKJEFLD1), the exit has been updated to include the SECLABEL operand in the parameter list. SECLABEL is a field on the full-screen logon panel and a operand on the LOGON command.

Reviewing macro libraries for TSO/E

You must review and modify, as necessary, the names of macro libraries specified in the following:
- JCL procedures
- ISPF-generated JCL
- Installation and maintenance procedures.

The following identifies the **distribution** libraries for TSO/E.
- **ATSOMAC** is the distribution library that contains executable and mapping macros used for general programming interfaces.
- **AMODGEN** is the distribution library that contains executable and mapping macros that are *not* used for general programming interfaces. These macros can be used to customize product function, diagnose product problems, and monitor or tune the product.

The following identifies the **target** libraries for TSO/E:
- **MACLIB** is the target library that contains executable and mapping macros used for general programming interfaces.
- **MODGEN** is the target library that contains executable and mapping macros that are *not* used for general programming interfaces. These macros can be used to customize product function, diagnose product problems, and monitor or tune the product.

Installing a REXX compiler

If you plan to install a REXX compiler on your system, the compiler's run-time processor and its associated compiler interface routines are identified to REXX through the use of the compiler programming table. This table needs to be link edited into the compiler programming table module, IRXCMPTM. An example of the source to create the module IRXCMPTM is shipped as member IRXREXX4 in SYS1.SAMPLIB.

For more information about the compiler programming table, see Chapter 43, “Routines and interfaces to support a REXX compiler,” on page 529.

Installing TSO/E for the first time

Do not customize TSO/E until the SMP/E installation of TSO/E is complete.
- Customize your installation's sample logon procedure, ICQAPROC, in SYS1.SAMPLIB.

Installing the information center facility for the first time

This section applies to new Information Center Facility installations.

- Execute ICQPOST1 and ICQPOST2 to distribute Information Center Facility parts to the appropriate execution libraries. Both macros are in SYS1.SAMPLIB.
- You can customize all non-display panels to meet your installation's needs.

For information about activating the Information Center Facility, see “Activating the information center facility” on page 61.

Chapter 4. Considerations for migrating TSO/E

The following topics describe migration considerations for TSO/E.

Migrating from one TSO/E release to another

The following sections apply to installations that are migrating from previous releases of TSO/E to the new release of TSO/E.

- CLIST libraries are originally installed in fixed-blocked format. If you previously reformatted your CLIST libraries to variable-blocked format for your SYSPROC concatenation, you must either convert the CLIST libraries for TSO/E to variable-blocked format, or convert any other CLIST libraries that are concatenated with those for TSO/E to fixed-blocked format.
- During the installation process, all non-display panels are replaced, and may be in a different format from the existing ones. If you want to preserve prior customization, do not restore backed-up copies of the old, customized panels. Re-do the customization in the equivalent non-display panels of the new release.

  For example, the non-display panel ICQSIEAM is replaced when TSO/E is installed. Your installation may have customized ICQSIEAM. See "Changing Information Center Facility defaults" on page 591 for the new fields in ICQSIEAM.
- To preserve customized data, do not execute the ICQPOST1 and ICQPOST2 jobs. These jobs replace tables that contain current Information Center Facility data for your installation.
- A new version of the sample logon procedure, ICQAPROC, is distributed with TSO/E. Compare it with your existing user and administrator logon procedures to determine any changes that need to be made.
- If you used the SYSDEF function to customize user enrollment profile defaults, you must customize those defaults again.

Preparing application manager tables for TSO/E

This section describes how to prepare Application Manager Tables for TSO/E.

Preparing modified or new application manager tables

If you used the Application Manager either to modify the Information Center Facility distributed applications or to add new applications, you must convert your current Application Manager tables as follows:

- Convert your current Application Manager tables using the conversion routine, ICQAMCR1. The conversion procedures are described in section "Migrating application manager tables from previous releases of the information center facility" on page 54.
- Use the ADD function of the Application Manager to install the new installation files. These files install the new Information Center Facility functions available with TSO/E. They are described on page 55.

Preparing unmodified application manager tables

If you either have not modified the Information Center Facility distributed applications or have not added new applications, do *not* convert your Application Manager tables. Instead, execute the ICQPOST1 job, which replaces tables that contain current Information Center Facility data for your installation.

Migrating application manager tables from previous releases of the information center facility

This section contains the steps to convert the Application Manager tables from previous releases of the Information Center Facility.

- Ensure that you are in an active ISPF session (for example, use the TSO/E option under ISPF) to make the Information Center Facility inactive.
- Activate the ISPF log data set, so that messages are logged into it. For example, select option 0.2 on the ISPF/PDF main panel to use or change the log data set default options.
- Select ISPF option 6 (COMMAND) on the ISPF/PDF main panel.
- Allocate the following file names from the previous release:
  - ICQAMTAB data set for the administrator Application Manager tables that are to be converted.
  - ICQCMTAB data set for the user Application Manager tables that are to be converted.
- Execute the following command to convert your existing Application Manager tables. You must specify the panel and message libraries. For example, *exec* is the command that is executing the conversion CLIST (ICQAMCR1) and *panels* and *messages* are the parameters passed to ICQAMCR1.

```
exec 'icq.tsoe25.icqcclib(icqamcr1)' 'panels(icq.tsoe24.icqplib)
messages(icq.tsoe24.icqnew.mlib)'
```

  Note: The data set names mentioned in the preceding example should be changed to those of the actual CLIST, panel, and message libraries that were used to install TSO/E. Specify the data set names fully qualified and without quotes.

  The TSO/E panel and message libraries are required because ICQAMCR1 needs the constant values in the non-display panel, ICQSIEAM, and in the new messages. The conversion procedure may be performed before or after customizing the values in ICQSIEAM. Customizing the values in ICQSIEAM is described in Chapter 47, “Preparing the Information Center Facility for use,” on page 585.

  Messages on your screen report the progress of the conversion, first a “Beginning conversion” indicator, then table-by-table status messages, indications of applications being converted, and finally an “Ending conversion” indicator. These messages are also written to the ISPF log data set.

  The conversion routine creates the following data sets, which contain the newly converted tables.

  - (prefix).ICQAMCNV - for the new system administrator library
  - (prefix).ICQCMCNV - for the new system user library

  If you set the prefix using the TSO PROFILE(PREFIX) command, the prefix is *syspref*. If you do not set the prefix, the system defaults it to SYSUID.
- Free the old allocations to ICQAMTAB and ICQCMTAB.
- Either rename the data sets (containing the newly converted tables) using the names specified in the logon procedure for ICQAMTAB and ICQCMTAB, or change the logon procedure to include the correct data set names.
- Use the converted tables either to allocate the new data sets to the file names ICQAMTAB and ICQCMTAB, or to re-logon with the updated logon procedure described in the previous step.

- After the tables have been converted, start the Information Center Facility as an Application Manager administrator.
- To complete the installation process of TSO/E, install the applications using the *member names* shown in Table 3. Several of the applications are replacements for older applications. Therefore, compare the new installation files with your current application definitions to determine any customization that may have to be re-done.

  The applications are provided in ICQ.ICQILIB. When installing the applications from ICQ.ICQILIB, set the **Available** field on the main application definition panels (ICQAME20, ICQAME30, and ICQAME35) to Y. It is important that this field is correctly set for ICQAPPLMGR.

  The first seven applications shown in Table 3 contain Application Manager functions and must be installed first. Then install the last four. ICQAPPC must be installed last.

  Table names shown are the names that are distributed with TSO/E 2.3.0 and later. If you have used Application Manager to convert the Information Center Facility prior to TSO/E 2.3.0, the created names may be different from those listed in Table 3.

Table 3. Information center facility applications for TSO/E

Application name	Application description	Member name	Table name
ICQPVTAMGR	TSO/E Private Application Manager	ICQFF004	ICQMFAA6
ICQGRPAMGR	TSO/E Group Application Manager	ICQFF000	ICQMFABD
ICQGRPSPEC	TSO/E Group Specification	ICQFF001	ICQMFABE
ICQGCPRT	End-User Interface to Print Services	ICQFF002	ICQMFABF
ICQISPFDEF	Set ISPF System Defaults	ICQFF003	ICQMFAAQ
ICQAPPLMGR	TSO/E Application Manager	ICQFF005	ICQMFAAE
ICQENROLL	TSO/E User Enrollment	ICQFF006	ICQMFAAJ
ICQPROGRAM	TSO/E Programmer Services	ICQFP000	ICQMPAA#
ICQUTILITY	Information Center Facility Utilities	ICQFP002	ICQMPABG
ICQUSER	TSO/E User Services	ICQFP001	ICQMPAA9
ICQAPPC	APPC/MVS Administration Dialog	ICQFF007	ICQMFABH

Chapter 5. Activating the functions of TSO/E

After installing TSO/E, you must perform procedures to activate certain functions. The following section is an overview of the activation procedures for TSO/E. You must perform certain steps before users can use TSO/E. Other steps are required before users can use specific functions of TSO/E. For details about the procedures, see other publications as indicated.

Using TSO/E

The following steps must be performed before users can use TSO/E:

1. Initialize time sharing using the VTAM access method.

   Before terminal users can log on to TSO/E, VTAM must be active on the system. For how to initialize VTAM, see Chapter 6, “Defining and customizing TSO/VTAM and TSO/TCAM time sharing,” on page 67

2. To define users to the system, you can use either the user-attribute data set (SYS1.UADS) or the RACF data base, as described in Part 4, “Maintaining the UADS, RACF data base, and broadcast data set,” on page 193.

   If your installation has not installed TSO/E before and you intend to use SYS1.UADS rather than the RACF data base, make the following changes to the UADS:

   - Use the UADSREFM program to reformat SYS1.UADS, expanding it to 172 bytes for each user ID.
   - Optionally, use the ADD or CHANGE subcommand of the TSO/E ACCOUNT command to assign the RECOVER user attribute. Users must be authorized by the RECOVER attribute to use TSO/E EDIT command recovery. Also, assign installation defaults for the following user attributes:

     HOLD
     to assign a default for output class

     JOBCLASS
     to assign a default for job class

     MSGCLASS
     to assign a default for message class

     SYSOUT
     to assign a default for SYSOUT class

   You can convert some or all of the user IDs from SYS1.UADS to the RACF data base using the RACONVRT command. After UADS has been converted, you can use RACF commands to maintain the user information in the RACF data base. For more information about converting UADS, see Chapter 25, “Using the RACF data base to maintain TSO/E users,” on page 209.

3. If you have not installed TSO/E before and you are not using individual user logs, you must reformat the broadcast data set using the SYNC command to allow the new broadcast data set to be used. Reformatting with the SYNC command also reduces channel, control unit, and device busy time when a new record is written to the broadcast data set. For more information about the SYNC command, see *z/OS TSO/E System Programming Command Reference*. For more information about reformatting the broadcast data set, see Chapter 24, “Working with the UADS and broadcast data set,” on page 199.

To reduce contention for the broadcast data set, you can use individual user logs to store messages (mail) for users instead of using the broadcast data set. To use individual user logs, update the SEND command defaults in SYS1.PARMLIB member IKJTSOxx, as described in Chapter 36, “Customizing how users send and retrieve messages,” on page 333.

4. If you have existing versions of CSECTs INMXPARM, IKJEFTE2, IKJEFTE8, IKJEFTNS, and IKJEFTAP, you can reinstall the CSECTs, or you can use SYS1.PARMLIB member IKJTSOxx to perform the functions of those CSECTs.

   If you have no existing versions of the CSECTs, use IKJTSOxx instead. If you decide to use the CSECTs, you must modify them as described.

   For more information on CSECTs and IKJTSOXX, see Chapter 11, “Specifying authorized commands/programs, and commands not supported in the background,” on page 151 and on INMXPARM see Chapter 13, “Setting up the TRANSMIT and RECEIVE environment,” on page 161.
5. New RACF programs need to get control authorized. If you have the existing version of CSECT IKJEFTE8, you can reinstall it, or you can use SYS1.PARMLIB member IKJTSOxx to perform the function of IKJEFTE8.

   If you have no existing version of CLIST IKJEFTE8, use IKJTSOxx instead. If you decide to use IKJEFTE8, you must modify it as described.
6. IPL, specifying “clpa” to refresh the LPA.

Using specific functions of TSO/E

The following steps must be performed before users can use specific functions of TSO/E.

Activating base TSO/E functions

To activate specific functions of base TSO/E, perform the following:

- If you have not already done so, add the subcommands of the TSO/E TEST command (AND, OR, LISTVSR, and UNALLOC) to the SMF CSECT IEEMB846. If the subcommands are not in IEEMB846, they are recorded as *OTHER on Type 32 SMF records. See *z/OS MVS System Management Facilities (SMF)* for the SMF record format.
- If your installation plans to record CONSOLE subcommands, add the subcommands to the SMF CSECT IEEMB846. If the subcommands are not in IEEMB846, they are recorded as *OTHER on Type 32 SMF records. For information about SMF record format, see *z/OS MVS System Management Facilities (SMF)*.
- Reinstall your installation's versions of TSO/E exit routines that were deleted or replaced when you installed TSO/E. See Chapter 3, “Considerations for installing TSO/E,” on page 45 for information about the exit routines that are deleted or replaced when you install TSO/E.
- For the TESTAUTH, TESTA, and MVSSERV commands to work under ISPF, you must update the ISPF command table, ISPTCM. For information about updating the table, see *z/OS ISPF Planning and Customizing*.
- To use security labels to protect system resources, do the following:
  – Use RACF to define security labels and to activate security label checking.
  – Define all TSO/E users through RACF and specify a security label (SECLABEL) for each user's user ID. Each user may have a TSO segment created within that user's RACF profile.

Note: If you want to use security labels, do not activate the Information Center Facility. The Information Center Facility does not support the security enhancements.

- If your installation plans to use the RACF resource classes JESJOBS and JESSPOOL, you should reinstall the OUTPUT/CANCEL/STATUS sample exit, IKJEFF53, supplied in SYS1.SAMPLIB. For more information about the sample exit, see Chapter 34, "Customizing the SUBMIT command and job output processing," on page 293.
- To protect the security classification of messages, do the following:
  - Change the SEND PARMLIB parameter by editing the IKJTSOxx member of SYS1.PARMLIB (where *xx* is a member name suffix) to:
    - Set the LOGNAME operand to the high-level qualifiers for the user log data set name, which must be other than SYS1.BRODCAST or *.
    - Set the MSGPROTECT operand to ON so that the *logname.userid* user log is protected and the message can be viewed only if the user is logged on with the proper security label. With this setting, the user log data set naming convention is logname.userid and the sender's security label is associated with the message.
    - Set the USEBROD operand to OFF so that messages are not stored in the broadcast data set. Instead, they are stored in the *logname.userid* user log. Users can only view their *logname.userid* user log by using the TSO/E LISTBC command or logging on.

    For more information about the USEBROD and MSGPROTECT operands, see Chapter 36, "Customizing how users send and retrieve messages," on page 333.
- During IPL, the IKJTSO00 member is read. Edit the IKJTSO00 member, so that security protection is automatically activated at each IPL time.

  If you do not update IKJTSO00, you need to issue the dynamic PARMLIB command using the UPDATE operand and specify the suffix of the member (xx) you have edited. You must have UPDATE authority to the RACF security resource class, TSOAUTH, to issue the dynamic PARMLIB command.
- If you are using security labels, create a generic profile for *logname.** (user-log data-set name) with a universal access of NONE, and specify SYSHIGH for the SECLABEL to protect each user's individual user log, which contains protected messages. Creating this generic profile prevents other users from viewing the contents of the user's user log and defines the user log as *system-high* because it may contain any level of information.
- Define the broadcast data set to RACF with UACC(READ) and SECLABEL(SYSLOW). This allows system notices to be stored in and retrieved from the broadcast data set.
- To control the use of the TSO/E SEND and LISTBC commands when using the security enhancements, see *z/OS Security Server RACF Security Administrator's Guide*.
- To control and audit the use of the TSO/E SEND command, define the RACF security resource class, SMESSAGE, for your users. For more information about using SMESSAGE to control the use of the TSO/E SEND command, see *z/OS Security Server RACF Security Administrator's Guide*.

Activating language enablement

This section explains how to activate Language Enablement.

Translated TSO/E messages: If your installation plans to enable TSO/E to provide translated TSO/E messages, you must compile the message skeletons and initialize the MVS Message Service (MMS). In order to compile the message skeletons, you need the install message files that contain U.S. English message skeletons and translated message skeletons for each language you want to enable.

The message skeletons for U.S. English, listed below, are contained in SYS1.MSGENU.
- ADFSMREN
- IKJEDTEN
- IKJSCHEN
- IKJTSTEN
- IKJUTLEN
- IKJXMTEN

The message skeletons for uppercase U.S. English, listed below, are contained in SYS1.MSGENP.
- ADFSMREP
- IKJEDTEP
- IKJSCHEP
- IKJTSTEP
- IKJUTLEP
- IKJXMTEP

For more information and other languages see Chapter 20, “Customizing TSO/E for different languages,” on page 185.

Translated help information for TSO/E commands: You can use SYS1.PARMLIB member IKJTSOxx to define help data sets for different languages.

U.S. English help remains in SYS1.HELP. Corresponding help text in uppercase U.S. English is contained in SYS1.HELPENP. For more information and other languages see Chapter 20, “Customizing TSO/E for different languages,” on page 185.

Activating TRANSMIT and RECEIVE

Perform the following steps to activate TRANSMIT and RECEIVE:

- If TRANSMIT and RECEIVE are not already in your tables of authorized commands, add them. Use SYS1.PARMLIB member IKJTSO00 to authorize commands. For more information about using IKJTSO00 to add authorized commands, see Chapter 11, “Specifying authorized commands/programs, and commands not supported in the background,” on page 151. You can also add them to the tables in CSECTs IKJEFTE2 and IKJEFTE8.
- Specify defaults for the TRANSMIT and RECEIVE commands in member IKJTSO00, copied from SYS1.SAMPLIB to SYS1.PARMLIB, or in CSECT INMXPARM. If you choose to use an existing version of INMXPARM, you must reinstall and reassemble it. For more information, see Chapter 13, “Setting up the TRANSMIT and RECEIVE environment,” on page 161.
- If you use JES2, you must specify the JES2 initialization parameter TSUCLASS OUTPUT=YES. Otherwise, the TRANSMIT command will not work because the SYSOUT data from the TRANSMIT command will be deleted.
- JES2, JES3, and MVS are unaware that data sets being routed among nodes in a network are in transit to other nodes. Consequently, data sets are not checked for valid destinations during transmission. Check periodically for invalid destinations.

  If you have JES2, you can do either of the following steps:

  - Use the $DF command to list all data sets in transit, and then use the RECEIVE command with the USERID parameter to delete or reroute data sets with invalid destinations.
  - Use JES2 exit 13, the TSO/E TRANSMIT and RECEIVE screening and notification exit, to check for invalid destinations. See *z/OS JES2 Installation Exits* for information about that exit.

  If you have JES3, you can use exit IATUX42 to check for invalid destinations. See *z/OS JES3 Customization* for information about that exit.
- You can alter the limit on punch-card output for TSO/E sessions, which governs the size of a data set a TSO/E user can send using the TRANSMIT command, using the JES2 parameter ESTPUN (for FMID HJE1367 or later). Either:
  - Set ESTPUN to a high value to avoid D37 ABENDs when the TRANSMIT command is used, or
  - Use IKJTSO00 or INMXPARM to control output limits, as described in Chapter 11, "Specifying authorized commands/programs, and commands not supported in the background," on page 151 and Chapter 13, "Setting up the TRANSMIT and RECEIVE environment," on page 161.
- If your installation wants users to be notified when transmitted data arrives, you must use either RECEIVE exit INMRZ05R, exit 13 for JES2, or exit IATUX42 for JES3. If there is no exit routine, JES does not notify the user receiving the data.

Activating the session manager

Perform the following steps to activate the Session Manager:

- If you do not already have a logon procedure for Session Manager, create one. See Chapter 45, "Setting up a Session Manager environment," on page 559 for more information, including a sample logon procedure for the Session Manager.
- For performance, consider adding the load modules ADFIDF00, ADFMDF01 and ADFMDF03 to the MLPA.

Activating MVSSERV

To activate MVSSERV, if you have not done so previously, install any servers and initialize the input parameter data set as described in *z/OS TSO/E Guide to SRPI*.

Activating the information center facility

This section describes activating the Information Center Facility for existing and new users.

Note: If you want to use security labels, do not activate the Information Center Facility. The Information Center Facility does not support the security enhancements.

- Certain data sets are required for the Information Center Facility. You must create any of the data sets listed in Table 4 on page 62 that are not present on your system. The attributes for these data sets must conform to the attributes of your ISPF/PDF table libraries. The minimum space allocation for each data set is 221 blocks. You may want to allocate more space to allow room for future expansion.

  The tables distributed with the Information Center Facility are copied to the data sets shown in Table 4 on page 62 by the ICQPOST1 and ICQPOST2 jobs. These jobs are in SYS1.SAMPLIB.

Table 4. Information center facility data sets to allocate

Data set	Description
ICQ.ICQABTAB	Course abstracts
ICQ.ICQGCTAB	User requests
ICQ.ICQAATAB	Names
ICQ.ICQANTAB	News
ICQ.ICQTLIB	User enrollment tables
ICQ.ICQAPTAB	Printer support
ICQ.ICQAMTAB	Administrator Application Manager tables
ICQ.ICQCMTAB	User Application Manager tables

The data sets are assigned to DD names in the Information Center Facility logon procedure. Ensure that data set names match their DD names to simplify installation and problem determination. However, TSO/E does not define the data set names explicitly, so you can rename them to match naming conventions used on your system.

If you rename data sets or DD names, you must update the Information Center Facility logon procedure with the data-set or DD names.

Exception: If you rename any of the data sets or DD names listed below, you must use the Information Center Facility application manager to change the data set names or DD names listed to the right.

If you rename:	Change DD or data set names for:
ICQGCTAB or ICQ.ICQGCTAB	Administrator and user courses, administrator and user names
ICQTABL or ICQ.ICQTLIB	User enrollment, ISPF defaults, user types, and administrator names

You can also use a single data set for all functions by using the same data set name on all DD statements. However, using separate data sets allows you to identify a different administrator for each function. You could then use RACF or another protection facility to restrict access as needed.

- If you activated the Information Center Facility in TSO/E Version 2.1.0, run the conversion routine described in "Migrating application manager tables from previous releases of the information center facility" on page 54 to preserve customization.
- If you have never activated the Information Center Facility before, copy, edit, and execute members ICQPOST1 and ICQPOST2 in ICQ.ICQSAMP. When you execute ICQPOST1 and ICQPOST2, certain shipped library members are copied to other libraries.
- The Information Center Facility provides some non-display panels for customization. For example, if your installation calls IEBCOPY differently, you need to modify the following command in non-display panel ICQSIE00.

```
&IEBCOPY = 'TSOEXEC CALL ''''SYS1.LINKLIB(IEBCOPY)'''''
                     /* TSO command to invoke IEBCOPY.
                     /* Extra quotes are required for
                     /* passing the string into the CLIST.
```

For more information about customizing the non-display panels, see Chapter 48, “Customizing the Information Center Facility,” on page 615.

- If you have APL installed and have not done so for a previous release, create a physical sequential data set with the name @PL.@W000051.ICQUPDTS. The data set must have a RECFM of FBS, an LRECL of 80, and a BLKSIZE of 4240. The minimum space you can allocate is 5 blocks. You might want to allocate more space to allow room for future expansion.

  Copy member ICQAPL01 from ICQ.ICQAPL to @PL.@W000051.ICQUPDTS. If you have changed any workspace-naming variables in module APLYUOPT, make certain you use the same names in naming workspace 51 ICQUPDTS and in panel ICQSIECR. The default data set name is @PL.@W000051.ICQUPDTS. Panel ICQSIECR is a member of ICQ.ICQPLIB.
- If you have not already done so, create two logon procedures; one for administrators and one for end users. If you created the logon procedures previously, you might need to modify them. You can find a sample procedure, ICQAPROC, in ICQ.ICQSAMP. Edit ICQAPROC to create the administrator logon procedure. Then make a copy of ICQAPROC, and edit it to create the end user logon procedure, ICQPROC.

  ICQAPROC is based on an existing ISPF logon procedure. ICQAPROC uses CLIST ICQICF, which resides in ICQ.ICQCCLIB. ICQAPROC calls ICQICF to invoke ISPF and use application manager to invoke the initial application. You might want to rename and modify ICQICF for your installation. ICQICF checks the user's profile data set to see if the EDIT and PROFILE tables exist. The ISPF profile must be cataloged, if it exists.

  Read the prologue for instructions if ICQAPROC on DD statements to delete.
- The CLIST libraries ICQ.ICQACLIB and ICQ.ICQCCLIB are distributed with a RECFM of FB and an LRECL of 80. If your production ISPF CLIST data sets have a RECFM of VB, run the CLIST ICQSMC00, a member of ICQ.ICQSAMP, against ICQ.ICQACLIB and ICQ.ICQCCLIB. ICQSMC00 converts the CLIST data set members from a RECFM of FB to a RECFM of VB.

  Maintenance for TSO/E CLISTs is supplied through PUT tapes. For SMP/E users, PTFs are held for documentation by the ++HOLD card.

Part 3. Setting up and customizing the TSO/E environment

When setting up and customizing TSO/E, you must perform certain tasks before users can use TSO/E. Other customization tasks are optional and can be done at any time. This part describes how to set up and customize TSO/E. For more information about TSO/E customization and required and optional customization tasks, see Part 1, "Introduction," on page 1.

Before users can use TSO/E, you must define TSO/E to an access method, VTAM. After you complete the definitions, you can customize how TSO/E operates with VTAM. Chapter 6, "Defining and customizing TSO/VTAM and TSO/TCAM time sharing," on page 67 describes the tasks you perform to define and tailor the access methods.

Users must have access to the system before they can log on. You must write at least one logon procedure that users can use to log on to the system. Chapter 7, "Setting up logon processing," on page 75 describes how you control system access, limit address space size, and write logon procedures for different types of users.

You can customize the logon process in many ways. You can tailor logon messages and the use of the reconnect option. With RACF installed, you can define security labels and users can specify a security label (SECLABEL) for their TSO/E session when they log on. You can also hold and purge the SYSOUT data set that logon processing creates and limit the number of times TSO/E prompts a user for logon information. Chapter 8, "Customizing the logon and logoff process," on page 83 describes how to customize logon processing, including writing a logon pre-prompt exit and a logoff exit, and different ways you may improve performance during logon.

If your installation uses ISPF and ISPF/PDF, you must define TSO/E to ISPF and ISPF/PDF. You must write a logon procedure that allocates the required data sets. You can also specify which TSO/E commands users can issue from ISPF/PDF panels. If your installation uses Session Manager, you can define the Session Manager to ISPF/PDF so users can use Session Manager from ISPF/PDF. Chapter 9, "Defining TSO/E to ISPF and ISPF/PDF," on page 145 describes how to perform these tasks.

To allow users to issue authorized commands and programs, you must define the commands and programs to the system. You can also specify commands that users cannot issue from the background. Chapter 11, "Specifying authorized commands/programs, and commands not supported in the background," on page 151 describes how to specify these commands and programs using either SYS1.PARMLIB member IKJTSOxx or CSECTs IKJEFTE2, IKJEFTE8, IKJEFTAP, and IKJEFTNS.

The TSO/E service facility allows application programs to run unauthorized commands on a command invocation platform. In order for commands to run on the platform, you must define the eligible commands to the system. Chapter 12, "Specifying commands and programs for the command/program invocation platform," on page 157 describes how to specify the eligible commands using SYS1.PARMLIB member IKJTSOxx.

Before users can use the TRANSMIT and RECEIVE commands, you must set up the TRANSMIT and RECEIVE environment. Chapter 13, "Setting up the TRANSMIT and RECEIVE environment," on page 161 describes the tasks you must perform to make TRANSMIT and RECEIVE available to your users, including how to write the installation options CSECT, INMXPARM, or how to specify TRANSMIT and RECEIVE defaults using SYS1.PARMLIB member IKJTSOxx.

Chapter 14, "Customizing the HELP data set," on page 165 discusses the HELP data set and how you can customize it to include installation-specific information. The prompt mode HELP function lets users obtain additional on-line HELP information about a command's positional operands. By default, TSO/E provides this function for several TSO/E commands. You can update members in the HELP data set to provide the prompt mode HELP function for other TSO/E commands and for subcommands with positional operands. In addition you can use the include control character to include help information contained in a separate member.

The TSO/E Enhanced Connectivity Facility allows PC users to access host services. To use the Enhanced Connectivity Facility, you must perform initialization tasks related to the input parameter data set, diagnostic data sets, and servers and initialization/termination programs. Chapter 15, "Making host services available to PC users," on page 167 describes how to set up and customize the TSO/E Enhanced Connectivity Facility.

You can use other facilities, such as System Management Facilities (SMF) and RMF™, to monitor the way users issue TSO/E commands and monitor the performance of TSO/E users. Monitoring TSO/E resources gives you the statistics that help you analyze and improve performance of on-line users and batch transactions. Chapter 16, "Monitoring TSO/E resources," on page 169 discusses ways to monitor performance.

After you have monitored your TSO/E resources, you can decide what kinds of response time to give TSO/E users in general, and whether certain individuals and special groups should have better response time. Information about performance objectives is described in Chapter 17, "Defining performance objectives for TSO/E," on page 171.

You can limit users from issuing certain commands and accessing certain data sets. Chapter 18, "Protecting the resources TSO/E users can access," on page 175 provides an overview of the commands you can restrict from TSO/E READY mode, Session Manager, background mode, and ISPF/PDF panels. It also describe how to control the access TSO/E users have to certain data sets by using the MVS allocation input validation routine or RACF.

TSO/E provides support for displaying TSO/E information to users in different languages. The CONSOLE command also supports the displaying of translated system messages issued during a console session. Chapter 20, "Customizing TSO/E for different languages," on page 185 describes what you must do to provide information to users in their national language.

With RACF installed, you can use security enhancements. Chapter 21, "Security considerations for customizing TSO/E," on page 189 gives a brief description of the security considerations for customizing TSO/E and gives references to additional information.

Chapter 6. Defining and customizing TSO/VTAM and TSO/TCAM time sharing

Before users can log on to TSO/E, you must define TSO/E to the terminal access method of Advanced Communications Function for VTAM (Virtual Telecommunications Access Method).

You can then customize how TSO/E works with VTAM to suit the needs of your installation. This section describes how to define TSO/E to VTAM, and how you can customize TSO/VTAM.

Defining and customizing TSO/VTAM time sharing

The following is an overview of the steps you must perform to define TSO/E to VTAM:

- Define the primary TSO/E address space and TSO/E user address spaces to VTAM. For more information, see “Defining TSO/E address spaces to VTAM” on page 68.
- Write a procedure to start TSO/E and include it in a procedure library. For more information, see “Writing the procedure that starts TSO/VTAM time sharing” on page 68.

The following steps are optional:

- Tailor the session protocols, or rules, VTAM uses to start a session between a terminal and TSO/E. For more information, see “Tailoring VTAM session protocols” on page 69.
- Define the TSO/E LOGON command to VTAM. See “Defining the LOGON command to VTAM” on page 69 for more information.
- Create SYS1.PARMLIB member TSOKEY00, or an installation-defined alternate member, to override the default values that are used to start TSO/VTAM. For more information, see *z/OS MVS Initialization and Tuning Reference*.
- If you plan to use the TSO/E Session Manager, see “SYS1.PARMLIB changes” on page 559 about changes you may need to make to the SYS1.PARMLIB member to support a Session Manager environment.
- Build translation tables for your installation's requirements. For more information, see “Building translation tables for TSO/VTAM users” on page 69.
- Write VTAM exit routines to customize the connection between TSO/E and VTAM. For example, you can write exit routines to:
  – Tailor default logon error messages
  – Set the terminal type and buffer size
  – Support terminals not supported by IBM
  – Modify the way attention interrupts are handled. You may want to intercept and suppress attention interruptions that interfere with important processing.

Before a user can log on to TSO/E, both VTAM and the terminal control address space (TCAS) must be active in the system. The system operator enters the START command to start VTAM. After VTAM has been started, the system operator enters the START command to start TSO/E and activate TCAS. TCAS accepts logons from TSO/VTAM users and creates an address space for each user.

When a user logs on, the VTAM terminal I/O coordinator (VTIOC) is initialized. VTIOC controls the movement of data between TSO/E and VTAM. SYS1.PARMLIB member TSOKEY00 or an installation-defined alternate member contains parameters that are used during VTIOC initialization. If a member other than TSOKEY00 is used, the operator must include the member name either on the START command or in the procedure that the START command invokes. For a description of TSOKEY00, see *z/OS MVS Initialization and Tuning Reference*.

The system operator uses the MODIFY command to modify TSO/VTAM and the STOP command to stop TSO/VTAM. For more information on the START, MODIFY, and STOP commands and how they pertain to TSO/VTAM time sharing, see *z/OS MVS System Commands*.

Defining TSO/E address spaces to VTAM

To define TSO/E address spaces to VTAM, use VTAM APPL definition statements. You must code one APPL definition statement to define the primary TSO/E address space, and at least as many APPL definition statements as there will be users logged on to TSO/E at one time. For example, if you want to allow 50 users to use TSO/E simultaneously, code 51 APPL definition statements: one for the primary TSO/E address space and one for each user address space. You might want to code more APPL definition statements than you currently need, to accommodate users you plan to add to your system later. For more information about using the APPL definition statement, see *z/OS Communications Server: New Function Summary* and *z/OS Communications Server: SNA Resource Definition Reference*.

Writing the procedure that starts TSO/VTAM time sharing

You must write a procedure for starting TSO/VTAM time sharing. Include the procedure in either SYS1.PROCLIB or in an installation-defined procedure library.

Figure 5 is an example of starting TSO/VTAM time sharing.

```
//TCAS      PROC    MBR=TSOKEY00
//STEP1     EXEC    PGM=IKTCAS00,PARM='&MBR',TIME=1440
//PRINTOUT  DD      SYSOUT=A,FREE=CLOSE
```

Figure 5. Sample procedure to start TSO/VTAM time sharing

The procedure must contain the following statements:

PROC
names the procedure and, optionally, assigns default values to symbolic parameters defined in the procedure. In Figure 5, the PROC statement assigns the name `TCAS` to the procedure and designates a default SYS1.PARMLIB member (TSOKEY00). To start TSO/VTAM time sharing, the operator enters:

```
START TCAS
```

EXEC
identifies the program to be executed (IKTCAS00), a parameter to be passed on the invocation of IKTCASS00, and the maximum amount of time allowed for executing STEP1 (1440). The parameter passed to program IKTCASS00 is the symbolic parameter &MBR, which is defined in the PROC statement.

PRINTOUT DD
identifies where the time sharing parameters used are written. In Figure 5, the

PRINTOUT statement specifies that time sharing parameters be written to the device associated with output class A, and that the output data set be deallocated when it is closed.

Tailoring VTAM session protocols

VTAM supplies a logon mode table that defines default protocols. For information about the default protocols, see *z/OS Communications Server: SNA Customization*.

To tailor the protocols, you can either:

- Modify the existing logon mode table to modify the defaults for all users, including TSO/E, or
- Create supplementary tables to modify the defaults for TSO/E users only.

To specify that a logon mode table other than the default is used, specify the name of the table on the VTAM:

- PU, LU, LOCAL, or TERMINAL definition statements, or
- USSPARM macro statement.

For more information about tailoring or creating a logon mode table and using the USSPARM macro statement, see *z/OS Communications Server: SNA Customization*. For more information about using definition statements, see *z/OS Communications Server: SNA Resource Definition Reference*.

Defining the LOGON command to VTAM

VTAM does not recognize the TSO/E LOGON command as described in *z/OS TSO/E Command Reference*. For information about the LOGON command format VTAM expects, see *z/OS Communications Server: SNA Customization*. Most installations define either the TSO/E LOGON command or an installation-defined logon command to VTAM. To define a LOGON command to VTAM, use the following VTAM macro statements:

- INTAB
- LOGCHAR
- ENDINTAB.

For information about how to use these macro statements, see *z/OS Communications Server: SNA Customization*.

Building translation tables for TSO/VTAM users

Certain characters are unavailable on some types of keyboards. For example, on correspondence keyboards, the characters “<”, “>”, and “|” are unavailable.

Translation tables allow TSO/VTAM users to internally replace unavailable characters with characters that are available on the keyboard. For example, you can represent the following, unavailable characters as different characters:

Character:	**Represented As:**
<	[
>	]
|	!

For character translation, either default translation tables (ones supplied by IBM) or your own translation tables (ones that you write) are used. The default translation tables are part of the TSO/VTAM programs. They translate each character to itself.

When translation tables are in use, input translations are done after TSO/VTAM translates the line code to EBCDIC characters. Output translations take place before TSO/VTAM translates the EBCDIC characters to line code.

With translation tables, a terminal user can use the TERMINAL command and its operands TRAN and CHAR to specify replacement characters. The TERMINAL command invokes the STTRAN macro instruction to set up the translation tables.

A user can specify the TERMINAL command with different combinations of the CHAR and TRAN operands. The following describes the different combinations. See *z/OS TSO/E Command Reference* for more information about the syntax of the TERMINAL command and the TRAN and CHAR operands.

CHAR (*characters*)
Using the CHAR (*characters*) operand alone results in a copy of the default translation tables in the user's storage to be updated according to the *characters* that the user specifies. The system then uses the updated tables to translate all inbound and outbound characters. This method of translation takes place until the user specifies the TERMINAL command with the NOTRAN or NOTRAN operand or until the user's terminal session ends.

TRAN (*name*)
Using the TRAN (*name*) operand alone results in a copy of the translation tables located in *name* to be used to translate all inbound and outbound characters. This method of translation takes place until the user specifies the TERMINAL command with the NOTRAN operand or until the user's terminal session ends.

CHAR (*characters*)
TRAN (*name*)
Using the CHAR (*characters*) and TRAN (*name*) operands results in a copy of the translation tables located in *name* to be updated according to the *characters* which the user specifies. The system then uses the updated tables to translate all inbound and outbound characters. This method of translation takes place until the user specifies the TERMINAL command with either the NOTRAN or NOCHAR operand or until the user's terminal session ends.

The following steps describe how to build translation tables.

1. Code a pair of translation tables.

   You need one table for input (to TSO/E) and one table for output (to the terminal), with each pair in a control section. Each control section must consist of a fullword containing the:
   - Address of the output table, followed by
   - A 256-byte EBCDIC table (on a fullword boundary) for translating the inbound code, followed by
   - A 256-byte EBCDIC table for translating the outbound code.

   Format the tables according to the rules for the TRANSLATE instruction. For information about the TRANSLATE instruction, see *z/Architecture® Principles of Operation, SA22-7832*. Translation of numbers and uppercase letters is not allowed.
2. Assemble the translation tables.

3. Link-edit the translation tables into a load module library. One CSECT is allowed per member. The translation tables must be link edited as RMODE(24) to ensure correct addressing to the TERMINAL command module.
4. Place a JOBLIB DD or STEPLIB DD statement, containing the name of the load module library, into a logon procedure. The user can specify the logon procedure when logging on.

Figure 6 on page 72 shows translation tables that perform the following translation:

Translates:
To:

[(X'AD')
< (X'4C')

] (X'BD')
> (X'6E')

! (X'5A')
| (X'4F')

In the figure, these characters are highlighted. All other characters are translated to themselves.

Suppose the tables are located in member TRTAB1, and the data set which contains the member was specified in a logon procedure when the user logged on. To use the translation tables, the user would enter:

```
terminal tran (trtab1)
```

```
TRTAB1   CSECT
OUTADR   DC   A(OUTTAB)
INTAB    DS   0F
         DC   X'000102030405060708090A0B0C0D0E0F'   0X
         DC   X'101112131415161718191A1B1C1D1E1F'   1X
         DC   X'202122232425262728292A2B2C2D2E2F'   2X
         DC   X'303132333435363738393A3B3C3D3E3F'   3X
         DC   X'404142434445464748494A4B4C4D4E4F'   4X
         DC   X'505152535455565758594F5B5C5D5E5F'   5X
         DC   X'606162636465666768696A6B6C6D6E6F'   6X
         DC   X'707172737475767778797A7B7C7D7E7F'   7X
         DC   X'808182838485868788898A8B8C8D8E8F'   8X
         DC   X'909192939495969798999A9B9C9D9E9F'   9X
         DC   X'A0A1A2A3A4A5A6A7A8A9AAABAC4CAEAF'   AX
         DC   X'B0B1B2B3B4B5B6B7B8B9BABBBC6EBEBF'   BX
         DC   X'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'   CX
         DC   X'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'   DX
         DC   X'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'   EX
         DC   X'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'   FX
OUTTAB   DS   0F
         DC   X'000102030405060708090A0B0C0D0E0F'   0X
         DC   X'101112131415161718191A1B1C1D1E1F'   1X
         DC   X'202122232425262728292A2B2C2D2E2F'   2X
         DC   X'303132333435363738393A3B3C3D3E3F'   3X
         DC   X'404142434445464748494A4BAD4D4E5A'   4X
         DC   X'505152535455565758595A5B5C5D5E5F'   5X
         DC   X'606162636465666768696A6B6C6DBD6F'   6X
         DC   X'707172737475767778797A7B7C7D7E7F'   7X
         DC   X'808182838485868788898A8B8C8D8E8F'   8X
         DC   X'909192939495969798999A9B9C9D9E9F'   9X
         DC   X'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'   AX
         DC   X'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'   BX
         DC   X'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'   CX
         DC   X'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'   DX
         DC   X'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'   EX
         DC   X'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'   FX
         END
```

Figure 6. Example of a CSECT containing translation tables

Defining and customizing TSO/TCAM time sharing

The following is an overview of the steps for defining TSO/E to TCAM:

1. Write a message control program (MCP) procedure to start TSO/TCAM and include it in a procedure library. For more information, see “Writing an MCP procedure to start TSO/TCAM” on page 73.

   IBM provides a default MCP, which you can tailor to suit your installation's needs. For more information, see *ACF/TCAM Version 2 Base Installation Guide*.

   If you plan to use the Session Manager, see Chapter 45, “Setting up a Session Manager environment,” on page 559 about changes you may need to make to the MCP to support a Session Manager environment.
2. Optionally, you can create SYS1.PARMLIB member IKJPRM00 (or an installation-defined alternate member) to override the default values used to start TSO/TCAM. For more information, see *z/OS MVS Initialization and Tuning Reference*.

   If you plan to use the Session Manager, see “SYS1.PARMLIB changes” on page 559 about changes you may need to make to the SYS1.PARMLIB member to support a Session Manager environment.

Before a user can log on, TCAM must be active in the system and the terminal I/O controller (TIOC) must be initialized. The initialization of the TIOC completes the initialization for the time sharing subsystem and allows TCAM to accept logon commands and pass them to the TIOC for processing.

To start TCAM, the system operator enters the following START command. In the command, *tcam* is the name of a procedure that executes the TCAM MCP.

```
start tcam
```

After TCAM has been started, the system operator enters the MODIFY command to activate the TIOC as a subtask of TCAM:

```
modify tcam,ts=start
```

If you use a SYS1.PARMLIB member other than IKJPRM00 to override the parameters used to start TSO/E, the operator must include the member name on the MODIFY command. For example, if you want to use SYS1.PARMLIB member IKJPRM01, the operator must enter the following command:

```
modify tcam,ts=start,ikjprm01
```

For more information about IKJPRM00, see *z/OS MVS Initialization and Tuning Guide*.

To terminate all time sharing users' connections with the system, the system operator must issue the MODIFY command as follows:

```
modify tcam,ts=stop
```

For more information about the START and MODIFY commands, see *z/OS MVS System Commands*.

Writing an MCP procedure to start TSO/TCAM

You must write a message control program (MCP) procedure for starting TSO/TCAM time sharing. Include the procedure in either SYS1.PROCLIB or in an installation-defined procedure library. In the procedure, you must include an EXEC statement and DD statements. Figure 7 is a sample procedure for starting TSO/TCAM time sharing.

```
//TCAM EXEC PGM=IEDQTCAM,DPRTY=(13,9) TIME=1440
//
//R5041    DD  UNIT=0A1
//         DD  UNIT=0A2
//         DD  UNIT=0A3
//L3270    DD  UNIT=01D
```

Figure 7. Sample procedure to start TSO/TCAM time sharing

EXEC
identifies the MCP, IEDQTCAM, to be executed. If you omit the name of an MCP on the EXEC statement, the first MCP listed in the TST (TCAM subtask table) is attached. For more information, see *TCAM Operations Guide*.

If you use an MCP other than IEDQTCAM, you must specify the name of the MCP in the TCAM program properties table (PPT) and mark it nonswappable. The PPT describes the environment TCAM requires to operate properly. For more information, see *z/OS MVS Initialization and Tuning Reference*.

DD identifies the line addresses dedicated to TCAM. The ddnames must be the names identified on the DDNAME parameter of the TCAM LINEGRP macro statement used in TCAM initialization. For more information about the LINEGRP macro statement, see *ACF/TCAM Version 2 Base Installation Guide*.

Chapter 7. Setting up logon processing

Before users can log on to TSO/E, you must give them access to the system. You must review and, if necessary, adjust the variables that limit the number of users who can be logged on at one time. You must also review the size of users' address spaces and, if necessary, change the size. You can specify a default size that applies to all users, a default that varies from one logon procedure to another, or a default that applies to individual users.

You must write and make available the logon procedures that give users access to the resources they need.

Controlling logons

Several related factors, such as the number of available logons, the number of address spaces that can execute concurrently in your system, the number of VTAM APPL definition statements, and what operating system you are using, determine the number of users who can log on to TSO/E at one time. You can control both the total number of users who can be logged on to TSO/E at one time and the number of users in a group who can be logged on.

Controlling the total number of users who can log on

You can control the number of users who can be logged on to TSO/E at one time. Several factors determine the total number of users. These factors are:

- The number of available logons. You must ensure that you specify enough logons to accommodate the number of users you want to have logged on to TSO/E at one time. To specify the number of users who can be logged on at one time, use the USERMAX parameter in SYS1.PARMLIB member TSOKEYxx. For more information about the USERMAX parameter, see *z/OS MVS Initialization and Tuning Reference*.
- The number of address spaces that can execute concurrently in your system. Because each TSO/E user requires an address space, you must ensure that the total number of address spaces is at least as large as the number of available TSO/E logons. To specify the number of address spaces, use the following parameters in SYS1.PARMLIB member IEASYSxx: RSVNONR, RSVSTRT, and MAXUSER. For more information about these parameters, see *z/OS MVS Initialization and Tuning Guide*.
- The number of VTAM APPL definition statements that have been specified at your installation. You must ensure that you have at least as many VTAM APPL definition statements as available TSO/E logons. For more information about specifying VTAM APPL definition statements, see "Defining TSO/E address spaces to VTAM" on page 68 and *z/OS Communications Server: SNA Resource Definition Reference*.
- The operating system that you are using.

Each of these factors can limit the number of users who can log on. For example, if you have fewer APPL definition statements than the number of available logons, the number of users who can be logged on concurrently is limited by the number of APPL definition statements.

Controlling logons within groups of users

You can control the number of TSO/E logons by limiting them according to groups, such as application developers or end users, and using the logon pre-prompt exit (IKJEFLD or IKJEFLD1) to limit the number of users in each group who can be logged on concurrently. Limiting the logons within groups of users allows you to give certain groups better access to the system than other groups have. For more information about using the logon pre-prompt exit, see "Writing a logon pre-prompt exit (IKJEFLD/IKJEFLD1)" on page 90.

Duplicate logons

In z/OS V1R4, JES2 stopped preventing duplicate TSO/E logons. In z/OS V2R1, JES3 can be configured to allow duplicate TSO/E logons. See the DUPLOGON keyword for the OPTIONS statement in *z/OS JES3 Initialization and Tuning Reference*.

As a result, duplicate instances of a given userid can log on at the same time to different systems, even in the same JES2 MAS or JES3 JESplex, depending on the scope of the SYSIKJUA enqueue. If the enqueue has a scope of SYSTEM, multiple TSO/E logons are allowed. This configuration is supported, although users should note the following restrictions:

- Messages sent to the users who are logged on more than one system at a time might not go to the expected instance of the user.
- Logon reconnect will only allow users to reconnect to an instance of their userid on the same system where they attempt to log on.
- ISPF profile sharing or changes to default ISPF data set names might be needed to avoid errors in ISPF with multiple logons. See *z/OS ISPF Planning and Customizing* for more details about configuring ISPF to support multiple logons.

To prevent duplicate logons, take the following steps:

1. Merge all existing versions of SYS1.UADS into a single version of the data set.
2. Modify the default RNLs:
   a. Delete the SYSDSN entry for SYS1.UADS from the SYSTEMS exclusion RNL. See "Global resource serialization" on page 207 for more details on treating SYS1.UADS as a global resource.
   b. Add SYSIKJUA as a generic qname in the SYSTEM inclusion RNL.

Alternate ways to prevent duplicate logons are:

- Use JES2 Exit 44. See *z/OS JES2 Initialization and Tuning Guide* for more information about this exit.
- Use JES3 Exit IATUX29 where the customer will need to add code to check for duplicate names. See *z/OS JES2 Initialization and Tuning Guide* for more information about this exit.

Limiting the size of each user's address space

After you have limited the number of users who can log on to TSO/E, you may want to limit the size of users' address spaces (region size) so they cannot run commands or programs that use large amounts of virtual storage. You can then override, for individual users, the default limit you set; so they can run programs or commands that require additional storage.

You can use several methods to limit the size of each user's address space, or region:

- To specify a default region size that applies to all TSO/E users and jobs submitted by TSO/E users, use the JES2 TSUCLASS initialization statement. For more information about the TSUCLASS statement, see *z/OS JES2 Initialization and Tuning Reference*. You can override the region size for individual users or jobs by using either the MVS IEALIMIT exit for storage below 16 MB in virtual storage, or the MVS IEFUSI exit for storage above 16 MB in virtual storage.
- To specify a default region size for a certain set of users, use the REGION operand of the EXEC statement in the logon procedure.
- To specify a default region size for an individual user, you can use either the ACCOUNT command, or the RACF ADDUSER or ALTUSER command. Which command you use depends on whether you store information about users in the RACF data base or in UADS. (If you have RACF installed, you can define user information in the RACF data base. For more information, see Part 4, "Maintaining the UADS, RACF data base, and broadcast data set," on page 193.)

  Users can override the default region size when they log on. You can limit the region size with which a user can log on by specifying the maximum region size on either the ACCOUNT command or the RACF ADDUSER or ALTUSER command. Users can log on with a region size that is larger than the maximum only if you specify in the logon pre-prompt exit that neither the UADS nor the RACF data base be checked to verify logon information.
- To override the region size a user specifies on the LOGON command, use the logon pre-prompt exit. If you use the logon pre-prompt exit to override the region size, you must specify that neither the UADS nor the RACF data base be checked to verify logon information.

  TSO/E uses the following search order to determine the region size for a TSO/E user:
  1. Logon pre-prompt exit specifies the size
  2. The SIZE operand of the LOGON command
  3. The default region size defined for the user
  4. The REGION operand on the EXEC statement in the user's logon procedure
  5. The JES2 TSUCLASS initialization statement

  When TSO/E obtains a valid region size, it stops searching

Writing and giving users access to logon procedures

To enable users to log on to TSO/E and access system resources, you must write logon procedures, and give users access to them. A logon procedure:
- Specifies the system resources available to the user logging on
- Specifies the number of data sets the user can have dynamically allocated
- Specifies which program is invoked after the user has logged on. The program can be the TSO/E terminal monitor program (TMP), Session Manager, or an installation-written program.

Deciding the number of logon procedures you will make available

The number of logon procedures you need depends on the different types of users at your installation and the applications and data sets to which the users need to access. Using multiple logon procedures allows you to give users access to only the resources they need. For example, you can create a logon procedure that gives users access to line-mode TSO/E only, or one that also gives users access to the TSO/E Information Center Facility.

However, you should not write more logon procedures than required, for the following reasons:

- If users want to access a different logon procedure than the one with which they logged on, they must log off and log back on.
- Minimizing the number of logon procedures at your installation makes it easier to maintain the users on the system, because you must keep track of and give users access to each additional logon procedure.

One way of minimizing the number of logon procedures at your installation is to create a few logon procedures that allocate the data sets most users require, and allocate the data sets required by individuals in a CLIST or REXX exec. Individuals can maintain their own CLISTs or REXX execs. Specify the name of the CLIST or REXX exec in the logon procedure as the first command that is executed when the user logs on.

Note, however, that when you allocate data sets from CLISTs or REXX execs, the data sets are not permanently allocated, and a user can accidentally deallocate them. To avoid the possibility of a user accidentally deallocating a data set, you can write a command processor to permanently allocate the data sets for the duration of the user's session. For more information about writing command processors, see *z/OS TSO/E Programming Guide*.

Removing complexity from logon procedures with the TSOLIB command

The TSOLIB command, introduced with TSO/E Release 5, allows users to dynamically link to load module libraries of their choice while remaining in their active TSO/E session. The TSOLIB command saves your installation from having to maintain several versions of logon procedures, or several user IDs with different logon procedures, for programmers who require the allocation of different versions of load module libraries.

Using the TSOLIB command to link to libraries or products — only when required — allows your installation to reduce the number of different logon procedures and to simplify existing logon procedures. You can reduce the number of default data set allocations at logon time, thus gaining improved performance.

See *z/OS TSO/E Command Reference* for command details and advantages of the TSOLIB command. Also, see Chapter 40, “Customizing the TSOLIB command,” on page 487 for details about writing exits.

Writing a logon procedure

Figure 8 on page 79 shows a sample logon procedure. The statements specify the TSO/E-supplied TMP (IKJEFT01) for execution, and define the CLIST library, the work data sets for the assembler, and the assembler macro library, and specify that SYSIN and SYSPRINT are to be directed to the user's terminal.

```
//AFPROC    EXEC    PGM=IKJEFT01,DYNAMNBR=20
//SYSUT1    DD      DSN=&SYSUT1,UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSUT2    DD      DSN=&SYSUT2,UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSUT3    DD      DSN=&SYSUT3,UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSPROC   DD      DSN=CLIST.PROC.LIB,DISP=SHR
//SYSLIB    DD      DSN=SYS1.MACLIB,DISP=SHR
//SYSIN     DD      TERM=TS
//SYSPRINT  DD      TERM=TS
```

Figure 8. Sample logon procedure

The only statement you must include in a logon procedure is an EXEC statement that identifies the TMP. You can only include one EXEC statement in each logon procedure. Logon procedure names (for example, AFPROC) must begin with A-Z; other characters can be alphanumeric. The name must not match that of any subsystem.

The TMP provided by TSO/E is named IKJEFT01. To use an installation-written TMP for a procedure, substitute the module name for IKJEFT01 in the PGM= operand in the EXEC statement. The following are additional operands you can use on the EXEC statement:

REGION
: limits the address spaces of users who log on with this logon procedure.

DYNAMNBR
: defines the number of data sets that can be dynamically allocated at the same time. A constant of 2 is always added to the DYNAMNBR value you specify. If DYNAMNBR= is omitted, the number of allocations is determined by the number of DD DYNAM statements. If DD DYNAM statements and DYNAMNBR= are both present, the number of concurrent allocations equals the combined total.

  Using DYNAMNBR allows data sets to be more quickly reallocated because control blocks for data sets remain in storage, even after the data sets have been deallocated. You should choose the value for DYNAMNBR carefully. The value should be large enough so that it is not readily exceeded by the number of dynamic allocation requests made during the user's session. However, the larger the value you specify for DYNAMNBR the more virtual storage is used. The actual amount of virtual storage depends on the number of data sets the user allocates and deallocates in a session. The value cannot exceed the number of concurrently-allocated resources specified in the SYS1.PARMLIB member ALLOCxx, parameter TIOT SIZE. For details refer to *z/OS MVS Initialization and Tuning Reference*. For more information about DYNAMNBR, see *z/OS MVS JCL Reference*.

PARM
: specifies a command, CLIST, or REXX exec to be interpreted as the first line of input from the terminal. The command, CLIST, or REXX exec is the first to execute after the user has logged on. For example, you could specify the name of a CLIST that allocates data sets required by the user.

  Note: TSO/E does not execute a command the user enters in the COMMAND field of the logon panel if the command specified in the PARM field of the logon procedure fails. For example, a command the user enters will fail if TSO/E cannot find the command in the PARM field on the EXEC statement of the logon procedure.

TIME
provides job step timing. The value you specify overrides the default values you specified via JES initialization statements. You can use JES exits to override the job step timing for individual users. For more information about the JES initialization statements you can use, see either *z/OS JES2 Initialization and Tuning Reference* or *z/OS JES3 Initialization and Tuning Guide*. For more information about the JES exits you can use, see either *z/OS JES2 Installation Exits* or *z/OS JES3 Customization*.

Defining data sets in a logon procedure

Data sets required by the user can either be defined dynamically, through the ALLOCATE command or dynamic allocation, or they can be defined in a logon procedure. Defining the data sets in a logon procedure has the following advantages:

- May improve on-line performance. Data sets required by a command or program do not have to be allocated each time the command or program is run.
- Allows volumes to be mounted. If the data set is on an unmounted volume, the operator is notified. Ordinarily, unless the user is authorized to mount volumes, dynamic allocation requests do not allow volumes to be mounted. If the volume is not already mounted, the request fails.
- Provides recovery from an off-line device condition. Messages tell the operator to vary the device on-line.

However, using DD statements in a logon procedure may increase the time it takes for users to log on. To improve the performance of the logon process, you may want to minimize the number of data sets you allocate in the logon procedure. For more information about the factors that improve logon performance, see "Improving the performance of the logon process" on page 87.

The following DD statements have special meaning and can be included in the logon procedure:

SYSEXEC
defines the current REXX exec library concatenation to the EXEC command when users use the implicit form of the command. By default, the system searches SYSEXEC first, followed by SYSPROC.

The data set described by this DD statement must be partitioned, and have a record format of V, VB, F or FB. Instead of allocating the REXX exec library in the logon procedure, you can allocate it dynamically using the ALLOCATE command. For example:

```
allocate da('rexx.proc.lib') f(sysexec) shr
```

SYSPROC
defines the current REXX exec or CLIST library concatenation to the EXEC command when users use the implicit form of the command. The data set described by this DD statement must be partitioned, and have a record format of V, VB, F or FB. Instead of allocating the REXX exec or CLIST library in the logon procedure, you can allocate it dynamically, by using the ALLOCATE command. For example:

```
allocate da('clist.proc.lib') f(sysproc) shr
```

To explicitly activate REXX execs or CLISTs, use the ALTLIB command. By using ALTLIB, you do not need to include all application-level REXX execs or CLISTs in the user's logon procedure. For information about the ALTLIB command, see *z/OS TSO/E Command Reference*. The search order for the user, application, and system libraries is shown and explained in Chapter 28, “Customizing the ALTLIB command,” on page 229.

By default, the system searches SYSEXEC first, followed by SYSPROC.

STEPLIB
defines a private step library, which can contain command processors. You should avoid using STEPLIB data sets in logon procedures, however, because they may adversely affect performance. Each time the user loads a module, the system searches the STEPLIB data set before searching any others. As an alternative, users can use the TSO/E TSOLIB command.

Storing logon procedures

You can store logon procedures in either SYS1.PROCLIB or in an installation data set. SYS1.PROCLIB cannot be shared among systems in a multiple system complex. If you want to share logon procedures among systems, you must place them in an installation data set that can be shared.

If you use an installation data set, the data set must be partitioned, and contain logon procedures only. Note that the name of a TSO/E logon procedure must not be the same as any subsystem.

If you use JES2, code a PROCxx DD statement to define the data set in the procedure for starting JES2. Use the TSUCLASS initialization statement to refer to the DD statement. For more information about specifying the TSUCLASS initialization statement, see *z/OS JES2 Initialization and Tuning Reference*.

If you use JES3, code an IATPLBxx DD statement to define the data set in the procedure for starting JES3. Use the STANDARDS initialization statement to refer to the DD statement. For more information about specifying the STANDARDS initialization statement, see *z/OS JES3 Initialization and Tuning Reference*.

Like other tasks, some of the JCL for a time shared task for a TSO/E logon is created automatically. The system generates a JOB and EXEC statement and gives control to the TSO exits that allow the installation to manipulate the values that are placed on the JOB and EXEC statements. In addition, TSO/E copies the characteristics specified in the logon panel to the JOB statement. The procedure name specified on a logon panel is placed in the EXEC PROC field to indicate which procedure to use. TSO/E procedures must be single-step procedures.

TSO/E logon procedures are started under the primary job entry subsystem (JES). A JES initialization statement determines the library that contains the TSO/E logon procedure. (Often, TSO/E logon procedures are separated from other procedures.)

In a JES3 environment, the JCL used to submit a TSO/E logon is limited to approximately 700 cards. For JES2, the limit is slightly more that 400.

Giving users access to logon procedures

To give a user access to a logon procedure, specify the name of the logon procedure using either the ACCOUNT command, or the RACF RDEFINE and PERMIT commands. Whether you use the ACCOUNT command or RACF commands depends on whether the information about the user is contained in the UADS or RACF data base. (If you have RACF installed, you can define user information in the RACF data base. For more information, see Part 4, “Maintaining the UADS, RACF data base, and broadcast data set,” on page 193.)

You can give users access to more than one logon procedure. Users can then specify a particular procedure when they log on.

In line-mode TSO/E, if a user has access to more than one logon procedure and does not enter a procedure name, TSO/E prompts the user for a procedure as follows. If the user is defined in the RACF data base, TSO/E selects one of the logon procedures from the RACF data base and displays a message with that procedure name. The procedure TSO/E selects is the first one in alphabetical order. To use the selected procedure, the user presses the Enter key. To use a different procedure, the user can enter the procedure name. If the user is defined in the UADS, TSO/E displays a message that prompts the user for a logon procedure name.

If the user uses full-screen logon, TSO/E automatically displays the logon procedure the user used for the last TSO/E session. If the user blanks out the procedure name field on the panel, TSO/E does the following depending on whether the user is defined in the UADS or in the RACF data base:

- If the user is defined in the UADS, TSO/E displays a message prompting the user to specify a procedure name.
- If the user is defined in the RACF data base, TSO/E selects one of the logon procedures from the RACF data base and displays the procedure name on the panel. The procedure TSO/E selects is the first one in alphabetical order. TSO/E also displays a message that prompts the user to change the procedure name, if desired.

Note: In the previous description of how TSO/E selects a logon procedure, the term “defined in the RACF data base” means the user is defined to RACF and has a TSO/E segment.

Your installation can change the logon procedure the user specifies by using one of the logon exits. For more information about the exits, see Chapter 8, “Customizing the logon and logoff process,” on page 83.

Giving users access to SECLABELs

With RACF installed, your installation may be using security labels. Security labels are associated with a TSO/E logon session, message, or data set. Security labels are checked and the result of that check determines if processing continues. Authority to use a security label is given using RACF. For more information about setting up security labels, see *z/OS Security Server RACF Security Administrator's Guide*.

Chapter 8. Customizing the logon and logoff process

This chapter describes how you can customize the logon and logoff process to suit the needs of your installation.

If you want to allow TSO/E users to logon using passwords that are greater than 8 characters in length, you can select an alternate TSO/E logon panel that allows users to enter longer passwords, which are named password phrases or pass phrases. For details on how to activate the alternate logon panel, see "Activating password phrase support" on page 84. Starting with z/OS Version 1 Release 12, TSO/E no longer syntax checks passwords. Instead they are passed directly to the security product for verification. This allows special characters to be entered that were not previously accepted by TSO/E. However, RACF does not allow special characters other than the national characters (@, #, and $) so RACF users are not impacted by this change. For more information about the syntax rules RACF enforces for passwords, see the *z/OS Security Server RACF Security Administrator's Guide*.

If you want to use the APPL class in RACF to prevent TSO/E users from logging on to certain systems in a sysplex, there is an option in TSO/E for verifying that users are authorized to log on to the specific system that they are attempting to access. See "Activating APPL verification" on page 84 for more information.

When users log on, they might receive a message notifying them that logon is in process. "Customizing logon messages" on page 85 describes how to change how often the message occurs.

If a user supplies incorrect information when attempting to log on, TSO/E requests that the user reenter the information. "Limiting the number of logon attempts" on page 86 explains how to change the number of times a user can unsuccessfully enter information before having to start over and reissue the LOGON command.

When a user disconnects from the system, the user's address space remains available for a certain period of time. While the address space is available, the user can reconnect without going through the logon process. "Customizing the reconnect option of the LOGON command" on page 87 describes how to change the length of time a user's address space remains available.

Each time a user logs off, the system writes several messages to a SYSOUT data set. Because you rarely need to refer to the messages, you might want to send the SYSOUT data set to a class you can hold and later purge. For more information, see "Suppressing the SYSOUT data set generated from the logon job" on page 87.

You might want to review the factors that affect the performance of the logon process. For an overview of those factors, see "Improving the performance of the logon process" on page 87.

The TSO segment, which includes the TSO profile, will only be written to the RACF database if it has changed during the TSO/E session where the user is issuing LOGOFF.

With RACF installed, your installation can use security labels (SECLABELs). Users can specify a SECLABEL during logon. For an overview of security labels, see "Using SECLABEL on the logon process" on page 88.

TSO/E provides several exits that enable you to customize the logon and logoff processes. Using the exits, you can modify the logon process:
- Before the logon prompt, using the pre-prompt exit IKJEFLD/IKJEFLD1
- Before the display of the logon panel, using the pre-display exit IKJEFLN1
- After the display of the logon panel, using the post-display exit IKJEFLN2
- After the prompting has completed, using the post-prompt exit IKJEFLD3.

You can perform several functions to customize the logon process:
- Verify, change, or supply various logon parameters
- Modify JCL associated with the logon procedure
- Reference or update various TSO/E control blocks.

For an overview of each logon exit and the different processing operations each exit performs, see "Overview of logon exit processing" on page 88.

TSO/E supports the authorized logoff exit, IKJEFLD2, to customize the logoff process. IKJEFLD2 allows your installation to perform clean-up operations and tasks. During logoff, IKJEFLD2 can:
- Change certain data areas and control blocks
- Control information written to the UADS data set
- Issue the LOGOFF or LOGON command to control re-logons.

For more information about the logoff exit, see "Writing a logoff exit (IKJEFLD2)" on page 137.

Also, you can customize the logon panels using the source provided. You can customize panels in a variety of ways, including adding or changing fields in a panel. See "Customizing logon panels and logon help panels" on page 140 for more information on the logon panel modules.

You can also customize the logon help panels by using the source provided. See "Logon help panel" on page 143 for more information on the help panel modules.

Activating password phrase support

To change the TSO/E logon panel and allow users to enter password phrases which can be up to 100 characters, follow these steps:
1. Change the PASSPHRASE parameter on the LOGON statement in the SYS1.PARMLIB member IKJTSOxx from the default of PASSPHRASE(OFF) to PASSPHRASE(ON).
2. Use the MVS SET IKJTSO=xx or TSO/E PARMLIB UPDATE(xx) command to update the system settings.

This activates the alternate IKJLQENU logon panel (mixed case U.S. English), as opposed to the default panel, IKJLPENU (mixed case U.S. English). For more information about the PASSPHRASE parameter, see *z/OS MVS Initialization and Tuning Reference*.

Activating APPL verification

By default, TSO/E users can log on to any system in a sysplex with a shared RACF database. To limit user's access to specific systems, follow these steps:

1. Activate the APPL class in RACF and define profiles for each of your systems or groups of systems.
2. Change the VERIFYAPPL parameter on the LOGON statement in the SYS1.PARMLIB member IKJTSOxx from the default of VERIFYAPPL(OFF) to VERIFYAPPL(ON).
3. Use the MVS SET IKJTSO=xx or TSO/E PARMLIB UPDATE(xx) command to update the system settings.

This tells TSO/E logon processing to pass an APPL to the RACROUTE VERIFY request for user authorization. For more information about the VERIFYAPPL parameter, see *z/OS MVS Initialization and Tuning Reference*.

TSO/E determines the APPL using the following method. To use this support, these profiles must be defined in RACF.

- If VTAM generic resources are used for TSO/E, define the application name using the TCASGNAM defined in the TSOKEYxx, SYS1.PARMLIB member.
- If VTAM generic resources are not used, define the application name using the format TSOxxxx, where xxxx is the SID (or SMF system ID) defined in the SMFPRMxx member of SYS1.PARMLIB. For example, if the SID is 3390, type TSO3390 in the profile. For more information, see *z/OS Security Server RACF Security Administrator's Guide*.

Deactivating LOGONHERE support

By default, LOGONHERE support for the LOGON command is on; TSO/E users can reconnect to their session even if no disconnection has been detected. LOGONHERE support makes it easier for users to reconnect from a new terminal or after renewing their IP address. To disable this support, follow these steps:

1. Change the LOGONHERE parameter on the LOGON statement in the SYS1.PARMLIB member IKJTSOxx from the default LOGONHERE(ON) to LOGONHERE(OFF).
2. Use the MVS SET IKJTSO=xx or TSO/E PARMLIB UPDATE(xx) command to update the system settings.

This deactivates the LOGONHERE support that allows users to reconnect to their session even if no disconnection has been detected. You can use the MVS DISPLAY IKJTSO,LOGON or TSO/E PARMLIB LIST(LOGON) command to check the current settings of the LOGONHERE parameter. For more information about the LOGONHERE parameter, see *z/OS MVS Initialization and Tuning Reference*.

Customizing logon messages

If logon processing takes more than the amount of time specified through the IKJTSO macro statement, the system issues a message to notify the user that the logon is still in process. You can change the number of seconds that elapse before users receive the `logon proceeding` message. By default, 300 seconds elapse before the message is issued. In most cases, the default is adequate. If users regularly receive the `logon proceeding` message, you may want to review the factors that affect logon performance, instead of changing how often the message is issued. For an overview of some of the factors you should consider when reviewing logon performance, see "Improving the performance of the logon process" on page 87.

To change the number of times the `logon proceeding` message is issued, use the IKJTSO macro instruction and submit a System Modification Program Extended

(SMP/E) job to make the change. For example, to specify that two minutes elapse before the message is issued, code IKJTSO as follows:

```
LIMITS IKJTSO LOGTIME=120
```

TSO/E provides a sample job in SYS1.SAMPLIB member IKJTSMPE that shows how to use IKJTSO.

Instructions for using the sample job are included in the SYS1.SAMPLIB member. “IKJTSO macro” on page 720 describes the syntax of IKJTSO.

If you used the TSO SYSGEN macro to change the default number of times the `logon proceeding` message is issued, when you install a new release of TSO/E, the default that TSO/E provides overlays your changes. After system generation, use IKJTSO to change the default attributes. By using IKJTSO, future SYSGENs or IOGENs do not replace your values.

You might also want to customize the text of error messages issued during logon processing. *z/OS Communications Server: SNA Messages* lists the messages you can customize. For information about how to customize the messages, see *z/OS Communications Server: SNA Customization*.

Limiting the number of logon attempts

The number of unsuccessful attempts users can make at entering information in response to logon prompts is limited to the value specified on the IKJTSO macro statement. If users exceed the limit, they must reissue the LOGON command. By default, a user can make 10 attempts at entering information in response to logon prompts. For example, a user can unsuccessfully respond to any of the logon prompts—password, account number, or procedure—and press the Enter key a total of 10 times before having to log on again. You might want to reduce the number of attempts a user can make to enter information, thus making it more difficult for unauthorized users to gain access to the system.

To change the number of unsuccessful attempts a user can make at entering information, use the IKJTSO macro instruction and submit a System Modification Program Extended (SMP/E) job to make the change. For example, to specify that a user can make 3 attempts before having to reenter the LOGON command, code IKJTSO as follows:

```
LIMITS IKJTSO LOGLINE=3
```

TSO/E provides a sample job in SYS1.SAMPLIB member IKJTSMPE that shows how to use IKJTSO. Instructions for using the sample job are included in the SYS1.SAMPLIB member. “IKJTSO macro” on page 720 describes the syntax of IKJTSO.

If you used the TSO SYSGEN macro to change the number of unsuccessful attempts a user can make at entering logon information, when you install a new release of TSO/E, the default that TSO/E provides overlays your changes. After system generation, use IKJTSO to change the default attributes. By using IKJTSO, future SYSGENs or IOGENs do not replace your values.

Customizing the reconnect option of the LOGON command

By default, after a user's terminal disconnects, the user's address space remains available for the period of time as defined by the RECONLIM parameter in SYS1.PARMLIB member TSOKEYxx. To reconnect while the address space is still available, users can use the RECONNECT operand of the LOGON command, and bypass the logon process. After the time limit expires, however, the address space is no longer available, and the user must reissue the LOGON command. You can change the amount of time during which a user can reconnect.

To prevent users from unnecessarily tying up available logons, you may want to reduce the amount of time users' address spaces remain available. Be careful, however, that you do not limit the time to a point where users do not have enough time available to reconnect. To limit the amount of time available for reconnecting, use the RECONLIM parameter in SYS1.PARMLIB member TSOKEYxx. For more information about the RECONLIM parameter, see *z/OS MVS Initialization and Tuning Reference*.

You can write a logon pre-prompt exit to make it easier for users to reconnect after their terminals have been disconnected. For more information, see “Writing a logon pre-prompt exit (IKJEFLD/IKJEFLD1)” on page 90.

Suppressing the SYSOUT data set generated from the logon job

Each time a user logs off, the system generates a SYSOUT data set containing messages from the processing of the logon job. By default, the data set is sent to output class A. Because you probably only rarely have to refer to this data set, you may want to send it to either a class that is held, so you can selectively purge the output, or to a purge class, so all output is purged. To change the output class to which the data set is sent, use the JES2 STCCLASS or TSUCLASS initialization statement or JES3 exit IATUX19. For more information about using the JES2 STCCLASS or TSUCLASS initialization statement, see *z/OS MVS Initialization and Tuning Reference*. For more information about writing JES3 exit IATUX19, see *z/OS JES3 Customization*.

One problem with sending the SYSOUT data set to a held or purge class is that messages from jobs submitted by TSO/E users are sent to the same class. So, you may want to direct messages from jobs submitted by TSO/E users to a separate class. To direct messages to a separate class, use either the ACCOUNT command, or the RACF ADDUSER or ALTUSER command. Which command you use depends on whether you store information about users in the UADS or the RACF data base. (If you have RACF installed, you can define user information in the RACF data base. For more information, see Part 4, “Maintaining the UADS, RACF data base, and broadcast data set,” on page 193.)

For more information about using the ACCOUNT command, see *z/OS TSO/E System Programming Command Reference*. For more information about using RACF commands, see *z/OS Security Server RACF Command Language Reference*.

Improving the performance of the logon process

You should review the performance of the logon process, and, if necessary, adjust the factors that affect logon performance. The following are some guidelines for improving the performance of the logon process:

- Shorten logon procedures by allocating as few data sets as possible. However, note that allocating data sets in a logon procedure may improve on-line performance, after a user has logged on. For more information, see "Defining data sets in a logon procedure" on page 80.
- Avoid the use of STEPLIBs in logon JCL.
- Reduce the amount of searching the system does to locate a logon procedure:
  - Rename or move logon procedures so they are at the beginning of the procedure library or at the beginning of the library concatenation.
  - If you are using JES3, specify commonly-used logon procedures in a BLDL list. To specify a BLDL list, use the JES3 PROC initialization statement. For more information about using the JES3 PROC initialization statement, see *z/OS JES3 Initialization and Tuning Guide*.
- Place logon procedures in a procedure library that is likely to be open when a user logs on. For example, if possible, place all your procedures—both those for batch and for on-line—in the same procedure library.
- Allow users to reconnect after being disconnected (thus bypassing the logon process). Use SYS1.PARMLIB member TSOKEYxx to extend the amount of time address spaces remain active after users have been disconnected. One problem with extending the amount of time is that it can allow users to tie up logons. For more information, see "Customizing the reconnect option of the LOGON command" on page 87.
- Bypass or reduce the use of the broadcast data set. You can:
  - Use individual user log data sets to store messages sent to individual users.
  - Limit the use of the OPERATOR SEND command at your installation.
  - Have users bypass the use of the broadcast data set by using the NONOTICES and NOMAIL parameters of the LOGON command. For more information about using the LOGON command, see *z/OS TSO/E Command Reference*.
  - Use the SYSPLEXSHR option of the SEND PARMLIB parameter to avoid I/O to the broadcast data set for the display of system notices. For more information, see "Broadcast data set in a sysplex" on page 207.

Using SECLABEL on the logon process

With RACF installed, you can use security labels to protect messages and data sets. If security labels are used, a SECLABEL is assigned to the user's session. This SECLABEL can be specified in the SECLABEL field in full-screen logon or as an operand on the logon command in line- mode logon. For more information on setting up SECLABELs and security label checking, see *z/OS Security Server RACF Security Administrator's Guide*.

Overview of logon exit processing

This section provides a brief overview of how logon exits let you customize the logon process.

When the TSO/E user enters the LOGON command, the command initiates a terminal session by supplying the system with information that the user specifies on the command, such as user ID, password, and account number. For more information about the LOGON command and its operands, see *z/OS TSO/E Command Reference*.

During logon processing, TSO/E invokes installed logon exit routines. These logon exit routines allow you to tailor various stages of the logon process to meet the needs of your installation. Figure 9 illustrates the flow of TSO/E logon exits. The figure shows the division of logon exit processing into four stages: pre-prompt, pre-display, post-display, and post-prompt. These stages are described in the following sections.

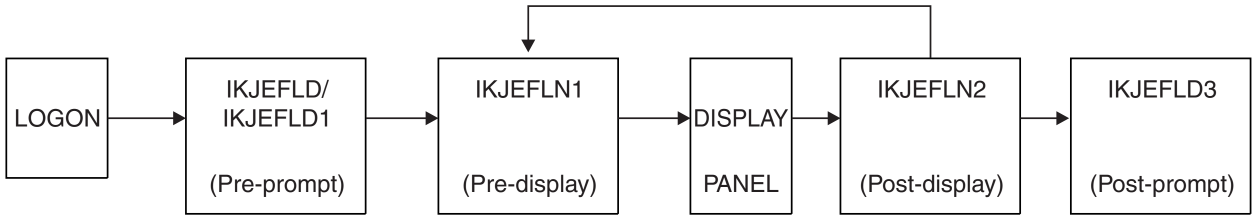


Figure 9. TSO/E logon exit flow

Logon pre-prompt exit IKJEFLD/IKJEFLD1

The logon pre-prompt exit IKJEFLD or IKJEFLD1 receives control:

- After the logon processor receives a LOGON command from the TSO/E user
- Before the logon processor accesses the UADS data set or the RACF data base to verify logon information
- Before the logon processor displays the full-screen logon panel for the first time. (If your installation does not use full-screen processing, the exit receives control before the user is prompted for information.)

Using exit IKJEFLD, you can verify, change, or supply various logon operands, or cancel a logon. You can use the authorized logon exit, IKJEFLD1, to perform the same functions as IKJEFLD, and additional authorized functions. For more information about the logon exits, see “Writing a logon pre-prompt exit (IKJEFLD/IKJEFLD1)” on page 90.

Logon pre-display exit IKJEFLN1

The pre-display exit, IKJEFLN1, receives control just before the logon panel is displayed to the TSO/E user. This exit receives control each time the logon panel is re-displayed, just before the panel is displayed to the TSO/E user.

You can use exit IKJEFLN1 to update information specified on the logon panel before the panel is displayed at the terminal and to provide information for the fields on the logon panel. IKJEFLN1 also can update fields in the PSCB and the UPT. For more information about IKJEFLN1, see “Writing a logon pre-display exit (IKJEFLN1)” on page 112.

After returning from the exit, the logon panel is displayed. The TSO/E user can then enter the required logon information and press the Enter key.

Logon post-display exit IKJEFLN2

The post-display exit, IKJEFLN2, receives control after the logon panel is displayed to the TSO/E user. This exit receives control when the TSO/E user presses the Enter key after a logon panel has been presented.

You can use exit IKJEFLN2 to validate and modify fields on the logon panel after the user has entered information on the panel. For more information about IKJEFLN2, see “Writing a logon post-display exit (IKJEFLN2)” on page 124.

Logon post-prompt exit IKJEFLD3

When prompting has completed and the logon processor has processed the LOGON command, the logon post-prompt exit, IKJEFLD3, receives control. Using IKJEFLD3, you can:
- Examine and modify the JCL statements built by logon processing
- Reference or update various TSO/E control blocks after they are built
- Provide additional JCL statements
- Terminate the logon process.

For more information, see "Writing a logon post-prompt exit (IKJEFLD3)" on page 134.

Writing a logon pre-prompt exit (IKJEFLD/IKJEFLD1)

Two exits enable your installation to modify the way logon operations are performed before the initial logon prompt:
- Logon pre-prompt exit IKJEFLD
- Authorized logon pre-prompt exit IKJEFLD1.

Exit IKJEFLD1 offers several advantages over IKJEFLD. IKJEFLD1 receives control in an authorized state and receives the standard TSO/E exit parameter list. With IKJEFLD1, you can perform the same functions as IKJEFLD, and:
- Specify the first TSO/E user command to be issued in the session.
- Specify job and SYSOUT classes.
- Specify that RACF processing is to be bypassed.
- Specify the relative block address (RBA) of the user's mail directory entry in the broadcast data set.
- Specify a 4-byte user word (value or address) of information to be used during the TSO/E session or at logoff.
- Specify a default security label (SECLABEL) to be used for the session. SECLABEL is recognized only when RACF is installed and security label checking has been activated. For information about setting up security labels, see *z/OS Security Server RACF Security Administrator's Guide*.
- Set up the console profile for the user. For information about the console profile, see Chapter 29, "Customizing the CONSOLE and CONSPROF commands," on page 233.
- Specify primary and secondary languages to be used for displaying translated information. For information about other tasks you need to perform to provide translated information, see Chapter 20, "Customizing TSO/E for different languages," on page 185.

IKJEFLD1 resides in its own load module, whereas IKJEFLD has to be link-edited with load module IKJEFLA in SYS1.LPALIB.

If both exits are installed, IKJEFLD1 receives control instead of IKJEFLD. If IKJEFLD1 is not installed but IKJEFLD is installed, logon processing passes control to IKJEFLD. The following is a functional description of logon processing and common elements in the logon pre-prompt exits. Exit-specific information follows the functional description.

Functional description

The logon pre-prompt exit routine initially receives control after the logon processor receives a LOGON command from a terminal user and before:

- The logon processor accesses the UADS or the RACF data base to verify logon information, and
- Displays the full-screen logon panel, if your installation uses full-screen logon processing, or
- Prompts the user for information in addition to the user ID, if your installation does not use full-screen logon.

Note: For more information about defining users in the RACF data base rather than in the UADS, see Part 4, “Maintaining the UADS, RACF data base, and broadcast data set,” on page 193.

You can write a logon pre-prompt exit to verify, change, or supply various logon operands and system characteristics. The exit can display a message at the user's terminal, request a response, and for full-screen logon, display your own logon panel instead of the default panel. You cannot change the TSO/E full-screen logon panel, but you can use the exit to display your own panel. The exit can also cancel a logon request to prevent a user from logging on.

The following highlights some ways you can use the logon pre-prompt exits. For more information, see “Possible uses” on page 109.

- Supply information for the LOGON command, such as user ID, password, procedure name, account number, region size, and performance group.
- Supply system attributes and user attributes for the protected step control block (PSCB), the generic unit name, and default SYSOUT destination. The user attributes can include authorization for the ACCOUNT, CONSOLE, and OPERATOR commands.
- Provide your own JCL statements instead of the standard JOB and EXEC statements that the logon processor builds.
- Validate the logon information that users specify.
- Display your own logon panel.
- Prevent users from logging on.
- Change logon processing depending on whether a user was disconnected or logged off from a previous TSO/E session.
- For IKJEFLD1 only:
  - Supply a TSO/E user command to be executed at the start of the user's session.
  - Supply job and SYSOUT classes.
  - Supply a security label (SECLABEL) for the session, if you have RACF installed and you use security label checking at your installation.
  - Set up the console profile for the user.
  - Specify primary and secondary languages to be used for displaying translated information.
  - Perform RACF processing and instruct logon processing not to issue a RACF RACINIT command.
  - Supply the relative block address (RBA) of the user's mail directory entry in the broadcast data set.
  - Supply a 4-byte user word for information to be used by the other logoff and logon exits.

TSO/E-supplied exits

TSO/E does not provide default logon pre-prompt exits.

Entry specifications

The contents of the registers on entry for the logon pre-prompt exits are:

Register 0
Unpredictable

Register 1
Pointer to a list of addresses, with one address for each parameter

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter list for IKJEFLD1

The logon pre-prompt exit IKJEFLD1 receives the standard exit parameter list. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32. Note that IKJEFLD1 does not use the following fields in the standard parameter list:

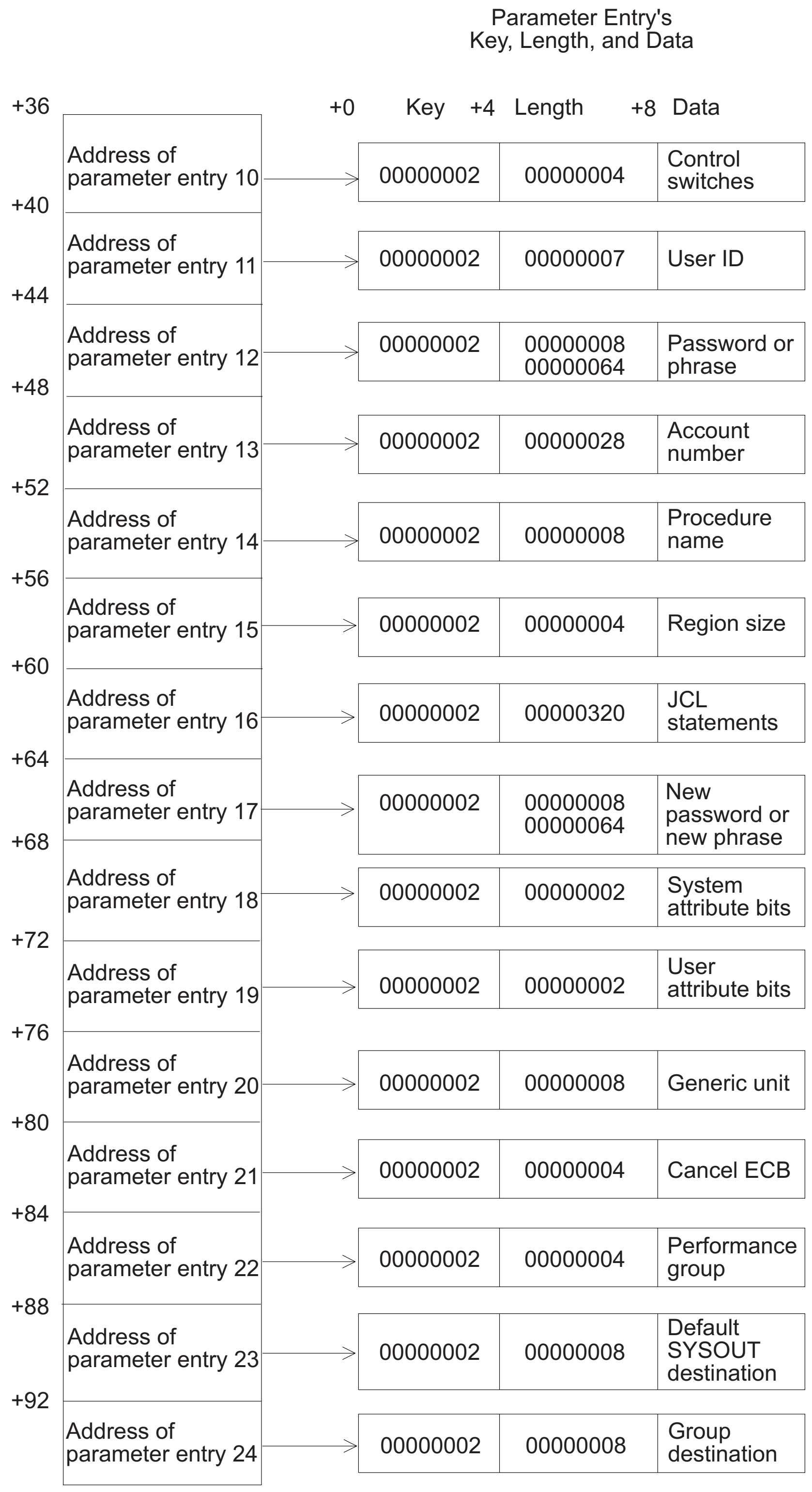
- The new command buffer field pointed to at offset +4
- The PSCB field pointed to at offset +16

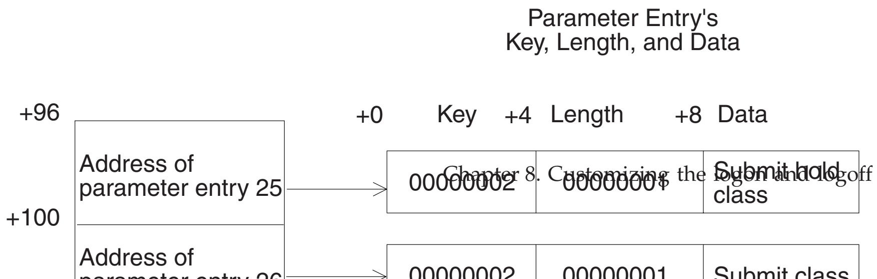
IKJEFLD1 cannot use the new command buffer parameter, parameter entry 2 in “TSO/E standard exit parameter list” on page 32, however the exit can use the command buffer parameter, parameter entry 1 in “TSO/E standard exit parameter list” on page 32. The exit changes the command buffer it receives and updates the length field (parameter entry 1) to specify the length of the updated command buffer.

The command buffer format for IKJEFLD1 differs from the regular command buffer format described in Figure 2 on page 35 in that the command buffer for IKJEFLD1 does not contain the four-byte header field. It contains only the text field. The command buffer for IKJEFLD1 may be altered as described in “Command and input buffer” on page 102.

Figure 10 on page 93 shows the exit-dependent data that IKJEFLD1 receives beginning at offset +36 (decimal) in the parameter list. Each parameter entry is described following the figure. You can give the parameters any names because their meanings are determined by their order. The names used in the figure are only for illustration.

Note: The value in the "Data" column contains actual data or the address of the actual data depending on the value in the "Key" column. For a definition of the Key values, see Table 1 on page 34.


Parameter Entry's
Key, Length, and Data
+36
+0 Key +4 Length +8 Data
Address of parameter entry 10
00000002 00000004 Control switches
+40
Address of parameter entry 11
00000002 00000007 User ID
+44
Address of parameter entry 12
00000002 00000008 00000064 Password or phrase
+48
Address of parameter entry 13
00000002 00000028 Account number
+52
Address of parameter entry 14
00000002 00000008 Procedure name
+56
Address of parameter entry 15
00000002 00000004 Region size
+60
Address of parameter entry 16
00000002 00000320 JCL statements
+64
Address of parameter entry 17
00000002 00000008 00000064 New password or new phrase
+68
Address of parameter entry 18
00000002 00000002 System attribute bits
+72
Address of parameter entry 19
00000002 00000002 User attribute bits
+76
Address of parameter entry 20
00000002 00000008 Generic unit
+80
Address of parameter entry 21
00000002 00000004 Cancel ECB
+84
Address of parameter entry 22
00000002 00000004 Performance group
+88
Address of parameter entry 23
00000002 00000008 Default SYSOUT destination
+92
Address of parameter entry 24
00000002 00000008 Group destination


Parameter Entry's
Key, Length, and Data
+96
+0 Key +4 Length +8 Data
Address of parameter entry 25
00000002 00000001 Submit hold class
+100
00000002 00000001 Submit class

Parameter list for IKJEFLD

Figure 11 on page 95 shows the format of the parameter list for IKJEFLD. You can give the parameters any names. Their meanings are determined by their order. The names used in the figure are only for illustration.

As Figure 11 on page 95 illustrates, the entries in the address list point either directly to the parameter or to a two-word descriptor. The four entries that point directly to the parameter are:
- Region size
- Cancel ECB
- Last step completion code
- Performance group

The remaining entries point to a two-word descriptor that consists of the:
- Address of the parameter (four bytes)
- Maximum length of the parameter (two bytes)
- Current length of the data in the parameter (two bytes)

For example, in the address list "Account Number Desc." points to a two-word descriptor that has the following information:
- The address of the parameter (pointer to account number)
- The maximum length of the parameter (40 bytes)
- The current length of the data in the parameter (0 bytes)

For each parameter, the maximum length allowed for the data and the current length of the data are either expressed in bits or in bytes. In Figure 11 on page 95, the lengths that are expressed in bits are indicated by an asterisk (*). The other lengths are expressed in bytes.

For example, in the address list "UPT Desc." points to a two-word descriptor where the maximum length of the parameter is 448 bits; whereas in the address list "User ID Desc." the maximum length of the parameter is 7 bytes.

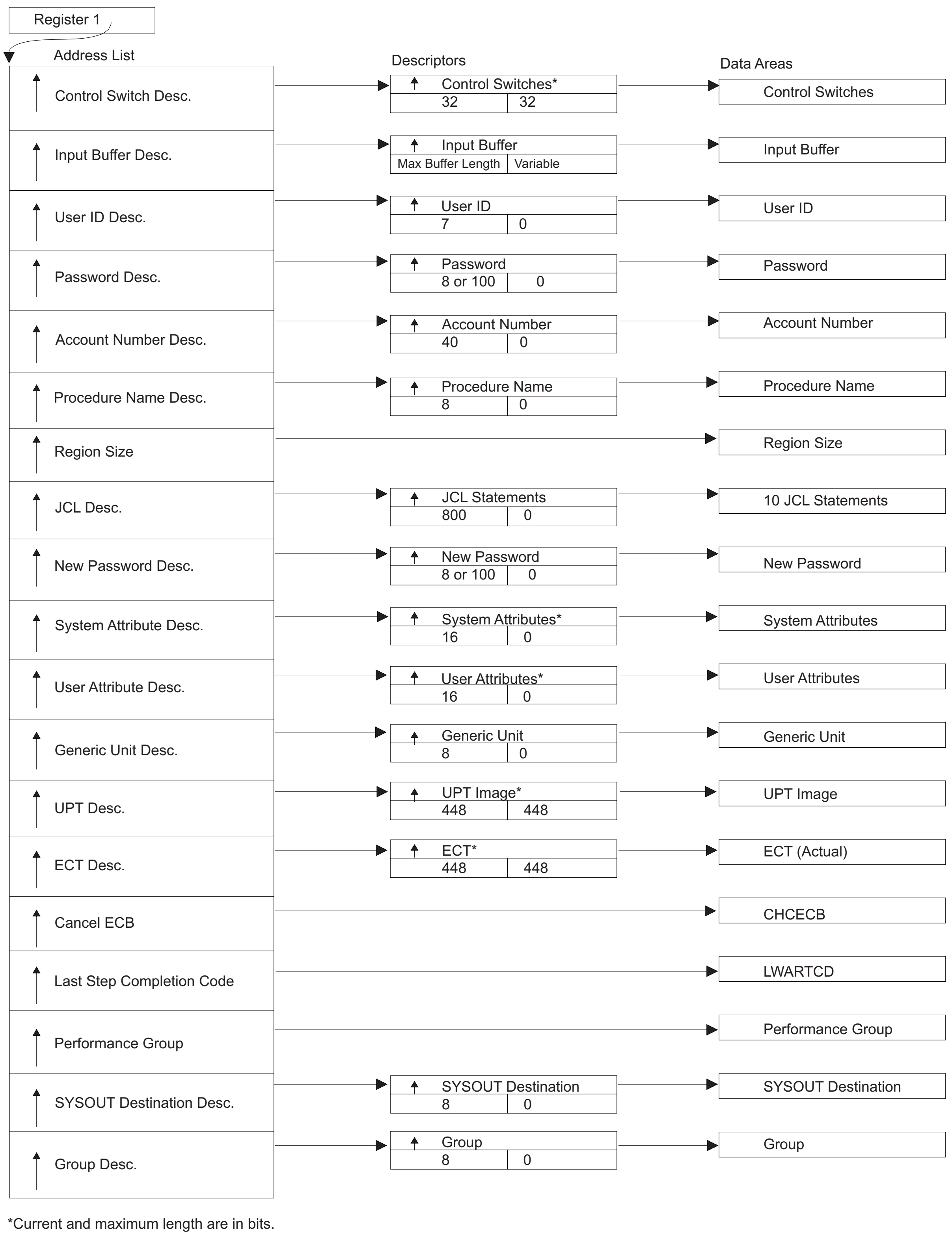


Figure 11. Parameter list for logon pre-prompt exit IKJEFLD

Parameter descriptions for IKJEFLD and IKJEFLD1

The following information describes the parameters for the logon pre-prompt exits IKJEFLD and IKJEFLD1. Parameters available to IKJEFLD1 only are described in "Parameters for IKJEFLD1 only" on page 105.

For IKJEFLD only: Note that, in the following parameter descriptions, the values for a) the maximum length of a parameter, and b) the current length of the data in a parameter, are expressed in bits or bytes. See "Parameter list for IKJEFLD" on page 94 and Figure 11 on page 95.

Control switches

The exit routine tailors logon processing by using the control switches and other parameters. Either the logon processor or the exit sets the different bits in the control switches parameter.

- The control switch bits that the logon processor sets indicate to the exit that certain system conditions exist.
- The bits that the exit sets indicate one of two things to the logon processor:
  - The exit is returning a value in a specific parameter
  - The logon processor needs to take some action.

Certain control switch bits that the exit can set relate directly to parameters in which the exit provides information. If the exit returns a value to the logon processor in a particular parameter, it must also set the corresponding control switch bit on. For example, if the exit routine provides its own JCL, it returns the JCL statements to the logon processor in the JCL parameter and sets the JCL control switch bit on.

Table 5 shows the control switch bits that the logon exits must set on if they return data in a particular parameter.

Table 5. Control switch bits the logon pre-prompt exit sets

Parameter	Control switch bit
Input buffer	No control switch bit is required
User ID	Don't Prompt bit
Password	Don't Prompt bit
Account number	Don't Prompt bit
Procedure name	Don't Prompt bit
Region size	Don't Prompt bit
JCL statements	JCL bit
New password	Don't Prompt bit
System attributes	• System Attributes bit, or • Don't Prompt and No UADS bits.
User attributes	• User Attributes bit, or • Don't Prompt and No UADS bits.
Generic unit	• Generic Unit bit, or • Don't Prompt and No UADS bits.
UPT	• UPT bit, or • Don't Prompt and No UADS bits.
ECT	No control switch bit is required
Performance group	Don't Prompt bit

Table 5. Control switch bits the logon pre-prompt exit sets (continued)

Parameter	Control switch bit
Default SYSOUT destination	Destination bit
Group	Don't Prompt bit

Table 6 shows additional parameters for IKJEFLD1 only, with the control switch bits that IKJEFLD1 must set on if it returns data in those parameters.

Table 6. Additional control switch bits that logon pre-prompt exit IKJEFLD1 sets

Parameter (IKJEFLD1 only)	Control switch bit (IKJEFLD1 only)
Submit Hold Class	Submit Hold Class bit
Submit Class	Submit Class bit
Submit Message Class	Submit Message Class bit
SYSOUT Class	SYSOUT Class bit
First Command	First Command bit
RBA	RBA bit
SECLABEL	SECLABEL bit
Console Profile	Console Profile bit
Primary Language	Primary Language bit
Secondary Language	Secondary Language bit

Table 7 shows the control switch bit configuration.

Table 7. IKJEFLD and IKJEFLD1 control switch bit configuration

Byte	Bit	Field name	Set by
0	0	User ID ENQ Failed	Logon processor
	1	Don't issue RACINIT	IKJEFLD1
	2	Resource Failure	Logon processor
	3	Disconnect	IKJEFLD, IKJEFLD1
	4	Don't Prompt	IKJEFLD, IKJEFLD1
	5	No UADS	IKJEFLD, IKJEFLD1
	6	JCL	IKJEFLD, IKJEFLD1
	7	No UADSE	IKJEFLD, IKJEFLD1
1	0	System Attributes	IKJEFLD, IKJEFLD1
	1	User Attributes	IKJEFLD, IKJEFLD1
	2	Generic Unit	IKJEFLD, IKJEFLD1
	3	UPT	IKJEFLD, IKJEFLD1
	4	Don't ENQ User ID	IKJEFLD, IKJEFLD1
	5	Destination	IKJEFLD, IKJEFLD1
	6	ABEND	Logon processor
	7	Reconnect	IKJEFLD, IKJEFLD1
2	0	Mail	IKJEFLD, IKJEFLD1
	1	Notices	IKJEFLD, IKJEFLD1

Table 7. IKJEFLD and IKJEFLD1 control switch bit configuration (continued)

Byte	Bit	Field name	Set by
	2	No Full-Screen Panel	IKJEFLD, IKJEFLD1
	3	Store Password in TSB	IKJEFLD, IKJEFLD1
	4	SUBMIT Hold Class	IKJEFLD1
	5	SUBMIT Class	IKJEFLD1
	6	SUBMIT Message Class	IKJEFLD1
	7	SYSOUT Class	IKJEFLD1
3	0	First Command	IKJEFLD1
	1	RBA	IKJEFLD1
	2	SECLABEL	IKJEFLD1
	3	Console Profile	IKJEFLD1
	4	Primary Language	IKJEFLD1
	5	Secondary Language	IKJEFLD1
	6	Don't Save	IKJEFLD1
	7	Reserved	Reserved

A description of the field names follows:

User ID ENQ Failed

If the ENQ on the user ID was unsuccessful, the logon processor sets this bit on to indicate that the user ID is in use.

The logon processor does not do an ENQ on the user ID before it initially invokes the exit. Therefore, when the exit first receives control, the User ID ENQ Failed bit is off.

If the exit sets the Don't ENQ User ID bit off, the logon processor performs an ENQ on the user ID after the exit returns control to the logon processor. If the user ID is in use (the ENQ failed), the logon processor sets the User ID ENQ Failed bit on and invokes the exit again.

Don't Issue RACINIT (IKJEFLD1 only)

If IKJEFLD1 performs RACF processing and you do not want the logon processor to issue a RACINIT, IKJEFLD1 must set the Don't Issue RACINIT and the Don't Prompt bits on. This bit will already be on in a z/OSMF ISPF environment. No RACINIT is needed as it was already done before the exit was called.

Resource Failure

If the logon processor was unable to obtain a resource other than the user ID, it sets this bit on.

Disconnect

The exit sets this bit on to tell the logon processor to terminate the session. The logon processor sends no further messages to the terminal. Before the exit returns control to the logon processor, it can issue an explanation to the user, for example, using PUTLINE.

Don't Prompt

If you do not want the logon processor to prompt the user for information, the exit must set the Don't Prompt bit on.

If the exit supplies any of the following information (not in the input buffer), it must set the Don't Prompt bit on:
- User ID
- Password
- Account number
- Procedure name
- Region size
- New password
- Performance group
- Group
- SECLABEL

If the exit sets the Don't Prompt bit on, it must return at least the following information to the logon processor:
- User ID
- Password
- Procedure name
- Account number (if your installation uses account numbers)

If the exit supplies any information in the input buffer, that information is ignored.

If you want the logon processor to prompt the TSO/E user for logon information, the exit must not set the Don't Prompt bit on (the bit must be off).

`No UADS`

The exit sets this bit on to tell the logon processor not to check either the UADS or the RACF data base to verify logon information. If the No UADS bit is on, no logon information is saved. If the user is defined to RACF, TSO/E always verifies the user ID and the password information with RACF even if the exit routine sets the No UADS bit on.

If the exit sets the No UADS bit on, it must supply the following information:
- System attributes
- User attributes (if your installation uses user attributes)
- Generic unit
- UPT

If the exit sets the Don't Prompt bit off, the logon processor ignores the No UADS bit. Here, the logon processor prompts the user for information. The logon processor also verifies the logon information with information either in the UADS or the RACF data base, depending on where the user is defined.

`JCL`

If the exit supplies JCL statements, it must set this bit on. The exit can supply a maximum of ten 80-byte JCL statements (in standard format) in the JCL parameter provided, or the exit can supply its own parameter for JCL. Logon processing will accept a JCL parameter containing up to 409 JCL statements, but your installation's JES capability determines the actual number of JCL statements allowed. Logon processing does not check that number. For information about the format of JCL statements, see *z/OS MVS JCL Reference*.

`No UADSE`

The exit sets this bit on to tell the logon processor to use the UADS or the RACF data base only for the LISTBC performance enhancement. If the exit sets the No UADSE bit on, it must also:
- Set the Don't Prompt and the No UADS bits on
- Set the Don't ENQ bit off

System Attributes
If the exit supplies the system attributes, it must set this bit on. If the exit sets this bit on, the system attributes are not saved. The exit supplies the attributes in the system attributes parameter.

User Attributes
If the exit supplies the user attributes, it must set this bit on. If the exit sets this bit on, the user attributes are not saved. The exit supplies the attributes in the user attributes parameter.

Generic Unit
If the exit supplies a generic unit name, it must set this bit on. The exit supplies the name in the generic unit parameter.

UPT
If the exit supplies the UPT, it must set this bit on. If the exit sets this bit on, the UPT is not saved. The exit supplies the UPT in the UPT parameter.

Don't ENQ User ID
The exit sets this bit on to tell the logon processor not to ENQ on the user ID. This bit will already be on in a z/OSMF ISPF environment, where no ENQ is needed. If the exit sets the Don't ENQ User ID bit on, it must also set the No UADS bit on and supply the following information:
- System attributes
- User attributes (if your installation uses user attributes)
- Generic unit
- UPT

Destination
If the exit supplies the default SYSOUT destination descriptor, it must set this bit on. The exit supplies the destination descriptor in the default SYSOUT destination parameter.

ABEND
The logon processor's ESTAI retry routine sets this bit on to indicate that an ABEND has occurred and that the logon processor is retrying.

Reconnect
If the user specified RECONNECT on the LOGON command, and the exit sets the “Don't Prompt” bit on, the exit must also set this bit on. The exit determines whether the user specified RECONNECT by checking the input buffer. When the Reconnect bit is on, a user's disconnected session is reconnected when the user logs on again.

If the Reconnect bit is on and there is no disconnected session, the user is automatically logged on normally when the Don't Prompt bit is off or when all of the following are true:
- The Don't Prompt bit is on
- The Don't ENQ User ID bit is off
- The necessary logon information (user ID, password, procedure, and so on) have been supplied

Mail
If the user is to receive mail, the exit must set this bit on. However, if the user specified MAIL on the LOGON command or the full-screen panel, the exit is overridden unless the Don't Prompt and No Full-Screen Panel bits are set. By default, the Mail bit is off.

Notices
If the user is to receive notices, the exit must set this bit on. However, if the user specified NOTICES on the LOGON command or the full-screen panel, the

exit is overridden unless the Don't Prompt and No Full-Screen Panel bits are set on. By default, the Notices bit is off. If the user specified NOTICES on the LOGON command, the exit must set this bit on. The exit determines whether the user specified NOTICES by checking the input buffer.

`No Full-Screen Panel`
If you do not want the logon processor to display the default full-screen logon panel, the exit must set this bit on.

`Store Password in TSB`
If you want the logon processor to store logon passwords in the terminal status block (TSB) for RACF users, the exit must set this bit on. By default, the passwords of users defined to RACF are not stored in the TSB. If you use the SUBMIT exit (IKJEFF10) to tailor job processing, you might want to retrieve passwords from the TSB. Passwords are not required on the JOB statement. However, passwords are required on the JOB statement if:

- Your RACF users are defined with the same user ID and password on more than one node in a network, and
- These users use the JES control statement `//*ROUTE XEQ` to route jobs to another node.

In this case, if you set the Store Password in TSB bit on to have the logon processor store passwords in the TSB, the SUBMIT exit can retrieve the passwords from the TSB and add them to the JOB statement. For more information about the SUBMIT exit, see “Writing an exit for the SUBMIT command” on page 296.

You should use the Store Passwords in TSB bit only if:

1. You use the SUBMIT exit, and
2. Your RACF users are defined with the same user ID and password on more than one node in a network, and these users use the JES control statement `//*ROUTE XEQ` to route jobs from one node to another.

`SUBMIT Hold Class (IKJEFLD1 only)`
If you want IKJEFLD1 to return a SUBMIT hold class, IKJEFLD1 must set the SUBMIT Hold Class bit on.

`SUBMIT Class (IKJEFLD1 only)`
If you want IKJEFLD1 to return a SUBMIT class, IKJEFLD1 must set the SUBMIT Class bit on.

`SUBMIT Message Class (IKJEFLD1 only)`
If you want IKJEFLD1 to return a SUBMIT message class, IKJEFLD1 must set the SUBMIT Message Class bit on.

`SYSOUT Class (IKJEFLD1 only)`
If you want IKJEFLD1 to return a SYSOUT class, IKJEFLD1 must set the SYSOUT Class bit on.

`First Command (IKJEFLD1 only)`
If you want IKJEFLD1 to return a TSO/E command to be issued at the beginning of the user's session, IKJEFLD1 must set the First Command bit on.

If the exit does not set the No Prompt bit on and allows logon to display the full-screen panel, the first command set by the exit will be displayed and the user can modify it.

`RBA (IKJEFLD1 only)`
If you want IKJEFLD1 to return the relative block address (RBA) within the broadcast data set, IKJEFLD1 must set the RBA bit on.

`SECLABEL bit (IKJEFLD1 only)`
If you want IKJEFLD1 to return a SECLABEL, IKJEFLD1 must set the SECLABEL bit on.

`Console Profile (IKJEFLD1 only)`
If you want IKJEFLD1 to return a console profile, IKJEFLD1 must set the Console Profile bit on.

`Primary Language (IKJEFLD1 only)`
If you want IKJEFLD1 to return a primary language to be used for displaying translated information, IKJEFLD1 must set the Primary Language bit on.

`Secondary Language (IKJEFLD1 only)`
If you want IKJEFLD1 to return a secondary language to be used for displaying translated information, IKJEFLD1 must set the Secondary Language bit on.

`Don't Save (IKJEFLD1 only)`
If you do not want the logon processor to save the values the user enters on the full-screen logon panel or values provided in the data areas in IKJEFLD1, IKJEFLD1 must set the Don't Save bit on. This bit is applicable only for users defined to RACF. It has no effect if user information is stored in the UADS instead. The fields that this bit controls are:
- Procedure name
- Account number
- Region size
- Performance group
- First command
- No Mail
- No Notices
- Reconnect
- Operator ID card
- Security Label

Command and input buffer

The command buffer for IKJEFLD1 or input buffer for IKJEFLD contains the information the user entered when logging on, including the LOGON command. For example, if the user entered `LOGON USER1`, the command or input buffer would contain `LOGON USER1`.

The logon processor obtains the buffer and passes it to the exit. The exit can alter the command or input buffer. If IKJEFLD alters the input buffer, it must update the current length of the data in the parameter. However, IKJEFLD must not change the maximum length of the parameter. See “Parameter list for IKJEFLD” on page 94.

IKJEFLD1 changes the command buffer it receives and updates the length field (parameter entry 1 in “TSO/E standard exit parameter list” on page 32) to specify the length of the updated command buffer.

Note: If the exit sets the Don't Prompt bit on, the logon processor ignores the altered input buffer.

If an ABEND occurred and the logon processor is retrying, the logon processor may pass a buffer length of zero to the exit. If there is a JCL error, the logoff processor may call the exit and pass a buffer length of zero.

User ID

The exit can return a user ID to the logon processor in the User ID parameter. The exit must also set the Don't Prompt bit on.

Password

The exit can return a password to the logon processor in the Password parameter. If the IKJTSO*xx* parmlib member indicates PASSPHRASE(ON) the exit can return a password phrase. Phrases can be 9 to 100 characters, while passwords can only be up to 8 characters. The exit must also set the Don't Prompt bit on.

Account number

The exit can return an accounting string to the logon processor in the Account Number parameter. The exit must also set the Don't Prompt bit on.

Procedure name

The exit can return the name of a cataloged procedure to the logon processor in the Procedure Name parameter. The procedure contains JCL that defines the resources the terminal job needs. The exit must also set the Don't Prompt bit on.

Region size

The exit can return a fullword to the logon processor in the Region Size parameter. The fullword contains the region size (Kbytes), in hexadecimal, for the terminal job. The exit must also set the Don't Prompt bit on.

If the user does not specify the SIZE operand on the LOGON command, the logon processor passes a region size of zero to the exit.

JCL

The exit can provide JCL statements to the logon processor in the JCL parameter. The JCL defines terminal job resources. The logon processor uses this JCL instead of the JOB and EXEC statements that it constructs. The default JCL parameter is a maximum of 800 bytes in length, in which the exit can return a maximum of ten JCL statements. To return more than ten JCL statements, the exit can provide its own parameter for JCL. Logon processing will accept a JCL parameter containing up to 409 JCL statements, but your installation's JES capability determines the actual number of JCL statements allowed. Logon processing does not check that number. To provide your own parameter for JCL, replace the JCL parameter address supplied by LOGON with the address of the alternative parameter. Then set the current length to the number of JCL statements multiplied by 80.

The logon processor expects that each 80 bytes of the parameter contains a full 80-byte JCL statement in standard format. It uses the current length field of the parameter to determine the number of statements that the exit is returning (current length/80 = n statements).

If the exit provides JCL, it must also set the JCL bit on.

If you issue the GETMAIN macro to obtain storage for an alternative JCL parameter, you must use a job-oriented subpool (such as subpools 0 to 127) whose storage will be automatically freed when the logon job ends, or you can use the user word to save the address so that logoff exit IKJEFLD2 can access it and issue the FREEMAIN macro to free storage in the subpool.

New password

For users who are defined to RACF, the exit can return a new password to the logon processor in the New Password parameter. If the IKJTSO*xx* parmlib member indicates PASSPHRASE(ON) the exit can return a new password phrase. The new password replaces the current password, or the new password phrase replaces the current password phrase. Phrases can be from 9 to 100 characters in length while passwords can only be up to 8 characters. The exit must also set the Don't Prompt bit on.

System attributes

The exit can return a value for the PSCBATR1 bit string in the PSCB to the logon processor in the System Attributes parameter. The system attributes include authorization for the ACCOUNT, CONSOLE, SUBMIT, and OPERATOR commands, and other system attributes for the user. The exit must also set either of the following bits on:
- The System Attributes bit, or
- The Don't Prompt and the No UADS bits

For a description of the PSCB, see *z/OS TSO/E System Diagnosis: Data Areas*.

User attributes

The exit can return a value for the PSCBATR2 bit string in the PSCB to the logon processor in the User Attributes parameter. The PSCBATR2 field is for use by the installation. The exit must also set either of the following bits on:
- The User Attributes bit, or
- The Don't Prompt and the No UADS bits

For a description of the PSCB, see *z/OS TSO/E System Diagnosis: Data Areas*.

Generic unit

The exit can return a value for the PSCBGPNM field of the PSCB to the logon processor in the Generic Unit parameter. The exit must also set either of the following bits on:
- The Generic Unit bit, or
- The Don't Prompt and the No UADS bits

If the exit does not provide a generic unit to the logon processor, the logon processor initializes the generic unit name from the UADS.

For a description of the PSCB, see *z/OS TSO/E System Diagnosis: Data Areas*.

UPT

The logon processor passes to the exit a UPT that contains binary zeros. The exit can update and return the UPT to the logon processor in the UPT parameter. If the exit returns an updated UPT, it must also set either of the following bits on:
- The UPT bit, or
- The Don't Prompt and the No UADS bits

Note: If the exit passes the UPT back but leaves the length or version fields zero, these fields are filled in by logon. If the primary or secondary language translation fields are left null or blank, the logon defaults them to ENU for mixed case U.S. English. If any of these fields are filled in by the exit, their validity is determined by the exit. For a description of the UPT, see *z/OS TSO/E System Diagnosis: Data Areas*.

ECT

The logon processor passes the ECT to the exit, which is the actual ECT that the logon processor builds. The exit can modify the ECT. If the exit modifies the ECT, it does not have to set any control switch bits.

Cancel ECB

The system uses the ECB for cancel processing. The exit can read the ECB, but it cannot alter it. If the operator has cancelled all TSO/E sessions, the Cancel ECB parameter is not zero and the logon processor ends the session of the user who is logging on.

Last step completion code

When the logon processor is retrying, it passes a fullword that contains the completion code for the user ID that it just logged off.

Performance group

The exit can return a fullword to the logon processor in the Performance Group parameter. The fullword contains the performance group number, in hexadecimal, for the terminal job. The exit must also set the Don't Prompt bit on.

If the exit does not return a performance group or returns a performance group of zero, the logon processor uses the system default performance group.

For more information about performance groups, see *z/OS MVS Initialization and Tuning Reference*.

Default SYSOUT destination

The exit can return the default SYSOUT destination for a user ID to the logon processor in the Default SYSOUT Destination parameter. The exit must also set the Destination bit on.

Group

For users who are defined to RACF, the exit can return a group name to the logon processor in the Group parameter. The group name is used as the user's current connect group. The exit must also set the Don't Prompt bit on.

Parameters for IKJEFLD1 only

The following are the parameters for IKJEFLD1 only:

Submit hold class

IKJEFLD1 can return a Submit hold class that is stored in the PSCB and serves as a default in the user's session. IKJEFLD1 must also set the Submit Hold bit on.

Submit class

IKJEFLD1 can return a Submit class that is stored in the PSCB and serves as a default in the user's session. IKJEFLD1 must also set the Submit Class bit on.

Submit message class

IKJEFLD1 can return a Submit message class that is stored in the PSCB and serves as a default in the user's session. IKJEFLD1 must also set the Submit Message Class bit on.

SYSOUT class

IKJEFLD1 can return a SYSOUT class that is stored in the PSCB and serves as a default in the user's session. IKJEFLD1 must also set the SYSOUT Class bit on.

First command

IKJEFLD1 can return the first TSO/E command to be executed. The command can be up to 80 bytes long. IKJEFLD1 must also set the First Command bit on. The first command can be returned regardless of the settings of the Don't Prompt and No UADS bits. However, if IKJEFLD1 does not set the Don't Prompt bit and allows logon to display the full-screen logon panel, the first command is displayed on the panel and the user can modify or delete it.

RBA

IKJEFLD1 can return the relative block address of the user's mail directory within the broadcast data set. IKJEFLD1 must set the RBA bit on.

SECLABEL

IKJEFLD1 can return a security label to the logon processor in the SECLABEL parameter. IKJEFLD1 must set the SECLABEL bit on. The exit must also set the Don't Prompt bit on.

Console profile

IKJEFLD1 can return a console profile that serves as the initial values for the user's session. IKJEFLD1 must also set the Console Profile bit on. For information about the console profile, see Chapter 29, “Customizing the CONSOLE and CONSPROF commands,” on page 233.

Primary language

IKJEFLD1 can return a primary language that is stored in the UPT and is used for displaying translated information. IKJEFLD1 must set the Primary Language bit on if it returns a primary language. The value specified must be a three-character language code defined to the MVS message service. Logon processing does not check that the three-character language code is defined.

The logon processor does not pass a value to the exit for this parameter. To provide a value, specify a key of 1, a length of 3, and the language code as immediate data.

For more information about language support and the tasks necessary to implement language support, see Chapter 20, “Customizing TSO/E for different languages,” on page 185.

Secondary language

IKJEFLD1 can return a secondary language that is stored in the UPT and is used for displaying translated information. If the primary language is not available or not supported by the user's terminal and a secondary language is specified, TSO/E uses the secondary language to display information. IKJEFLD1 must set the Secondary Language bit on if it returns a secondary language. The value specified must be a three-character language code defined to the MVS message service. Logon processing does not check that the 3-character language code is defined.

The logon processor does not pass a value to the exit for this parameter. To provide a value, specify a key of 1, a length of 3, and the language code as immediate data.

For more information about language support and the tasks necessary to implement language support, see Chapter 20, “Customizing TSO/E for different languages,” on page 185.

Return specifications

For IKJEFLD, logon processing set the contents of the registers on return to the same values they had on entry. The logon processor does not expect a return code from exit IKJEFLD.

For IKJEFLD1, the contents of the registers on return must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes

Table 8 shows the return codes that IKJEFLD1 supports.

Table 8. Return codes for the logon exit IKJEFLD1

Return code (decimal)	Description
0	Exit processing was successful. Logon processing continues.
12	Exit processing was unsuccessful. The logon processor issues an error message to the user and the console and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the LOGON command processor. For more information about the exit reason code, see “Exit reason code” on page 38.
16	Exit processing was unsuccessful. The logon processor terminates processing. The LOGON command processor does not display a message to the user or console if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the LOGON command processor terminates without displaying a message.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exits must be reentrant and reusable.

The exits can use the I/O service routines through the assembler macro instructions STACK, PUTLINE, GETLINE, and PUTGET. These macro instructions require the addresses of the ECT and UPT, which the logon processor passes to the exit. For information about the I/O service routines, see *z/OS TSO/E Programming Services*.

The exits must do the following for each parameter that they return:
- Move the data it wants to return to the logon processor into the appropriate parameter
- Update the current length of the data in the parameter
- Set the appropriate control switch bit on if a bit is required for the parameter

For more information about the parameters and the control switch bits, see “Entry specifications” on page 92.

For z/OSMF ISPF users, where a RACINIT has already been done by CEA before the exit is called, you may want to bypass your normal exit processing. The best way to determine that your exit is being called for a z/OSMF ISPF user is to examine the LWAWBSPF bit in the LWA control block pointed to by the ASXBLWA pointer. This bit will be set for z/OSMF ISPF users.

You use the control switch bit, Store Password in TSB, and the New Password and Group parameters for users who are defined to RACF. A user is defined to RACF if:

- The user ID information is stored in the UADS and the user is also defined to RACF, or
- The user ID information is stored in the RACF data base

For more information about defining TSO/E users in the RACF data base instead of in the UADS, see Part 4, "Maintaining the UADS, RACF data base, and broadcast data set," on page 193.

Environment

- IKJEFLD1
  - State: Supervisor
  - Key: 8
  - AMODE(24), RMODE(24) or AMODE(31), RMODE(ANY)
  - Not APF-authorized
- IKJEFLD
  - State: Problem program
  - Key: 8
  - AMODE(24), RMODE(24)
  - Not APF-authorized

Restrictions and limitations

- The exit must not change the:
  - Address of any parameter in the address list
  - Address of any parameter (in the two-word descriptor) other than the JCL parameter
  - Maximum length of any parameter (in the two-word descriptor) other than the JCL parameter

  If the exit changes any of the fields, the logon processor issues an error message to the console and an error message to the terminal, and then ends processing. For information about the address list and the two-word descriptor, see "Entry specifications" on page 92.
- The exit must not activate the prompt mode HELP function or the exit will ABEND with a 0B0 system completion code.
- TSO/E terminal users can change the characteristics of their user profile using the PROFILE command. Using PROFILE, users can define how they want to use the terminal. For information about the PROFILE command, see *z/OS TSO/E Command Reference*.

  Any profile changes that a user makes are stored in the user profile table (UPT). The new characteristics then remain valid from one session to another. However, profile changes are not saved if you use a logon pre-prompt exit and the exit does one of the following:
  - Supplies the UPT in the UPT parameter and sets the UPT bit on, or
  - Sets the No UADS bit on

  In either case, when a user logs off, the UPT is not updated with any PROFILE changes.

Installing the exits

You must name the exits as follows:

Authorized pre-prompt logon – IKJEFLD1
Unauthorized pre-prompt logon – IKJEFLD

Link-edit IKJEFLD1 as a separate load module. You can link-edit the exit in a separate authorized load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exit can reside in:
- The link pack area (LPA)
- LNKLST

For more information about using the LPA or LNKLST, see "Installing the standard-format exits" on page 39.

You must link-edit IKJEFLD with load module IKJEFLA in SYS1.LPALIB. Consult the output from stage 1 for correct linkage information. Use the system modification program (SMP) to make this update.

Possible uses

Some possible uses of the logon pre-prompt exits are described below. Additional uses for IKJEFLD1 only are described in "Possible uses for IKJEFLD1 only" on page 111.

- Supply or change values that users specify on the LOGON command.

  When users issue the LOGON command, they specify information such as user ID, password, account number, procedure name, region size, and performance group. When the exit receives control, it is passed an input buffer that contains the information that the user entered when logging on. The exit can change the logon information users specify and return the information to the logon processor in either an input buffer or a parameter. The exit can change the information in the input buffer, update the current length of the input buffer, and return control to the logon processor. The values that the exit can provide in the input buffer correspond to the operands of the LOGON command. For more information about the LOGON command, see *z/OS TSO/E Command Reference*.

  The exit can also return the following logon information to the logon processor in a parameter:
  - User ID
  - Password
  - Account number
  - Procedure name
  - Region size
  - New password (users defined to RACF only)
  - Performance group
  - Group (users defined to RACF only)

  If the exit returns the above logon information in the parameter, it must:
  - Supply the value in the parameter
  - Update the current length of the parameter
  - Set the Don't Prompt control switch bit on

  If the exit uses the parameters to supply this information, it must at least return the user ID, password, procedure name, and account number (if your installation uses account numbers) to the logon processor. These are required when the exit sets the Don't Prompt bit on.

  You must decide if the exit uses the input buffer or the specific parameters to return logon information. Your decision depends on whether you want the logon processor to prompt the user for information and validate the information in the UADS or the RACF data base (depending on where the user information is defined), and other processing the exit performs.

  As an example, suppose you want to use the same accounting number for a specific group of user IDs. The exit checks the input buffer for the user ID and corresponding account number, if any, that the user entered. If the user did not specify an account number or it is incorrect, the exit can return the account number to the logon processor using either the:

- Account Number parameter. The exit:
  - Provides the accounting string in the Account Number parameter
  - Updates the current length of the parameter
  - Sets the Don't Prompt bit on
  - Returns control to the logon processor

  The logon processor does not prompt the user for any information. The exit can prompt the user and return logon information to the logon processor. The exit must return at least the user ID, password, and procedure name to the logon processor because it set the Don't Prompt bit on.
- Input buffer. The exit:
  - Updates the input buffer with the ACCOUNT operand
  - Updates the current length of the input buffer
  - Returns control to the logon processor

- Supply system attributes and user attributes for the protected step control block (PSCB), the generic unit name, UPT, and default SYSOUT destination. The user attributes can include authorization for the ACCOUNT, CONSOLE, and OPERATOR commands.

  The exit can supply one or more, or all of these values in the appropriate parameters. If the exit supplies the default SYSOUT destination, it must set the Destination bit on. If it returns the system attributes, user attributes, generic unit, or the UPT it must:
  - Set both the Don't Prompt and No UADS bits on, or
  - Set the individual bits on and supply the following information:
    - System Attributes
    - User Attributes
    - Generic Unit
    - UPT

  Note: If the exit sets the Don't Prompt and No UADS bits on, then it must supply the system attributes, user attributes (if your installation uses user attributes), generic unit, and the UPT. These parameters are required whenever the exit sets the No UADS bit on. The system attribute bit for RECOVER/NORECOVER should be set to:
  - 0 to indicate RECOVER authority
  - 1 to indicate NORECOVER authority
- Provide your own JCL statements.

  The logon processor builds standard JOB and EXEC statements with logon information that the user provides, such as user ID and procedure name. You may want to provide your own JCL to add job parameters or supply DD statements. The exit provides the JCL in the JCL parameter and sets the JCL bit on. The logon processor then uses this JCL instead of the standard JOB and EXEC statements that it builds. For information about JCL statements, see *z/OS MVS JCL Reference*.

  The logon processor constructs a standard JOB statement. If your installation uses System Management Facilities (SMF) audit exits, the JOB statement is passed to SMF. If you want to include installation-dependent information, use the logon pre-prompt exit to provide your own JCL.
- Validate the logon information that users enter.

  The logon processor validates logon information with either the user information stored in the UADS or in the RACF data base, depending on where the user information is defined. The exit can check the information itself, for example, in your own data set, rather than using the UADS or the RACF data base. The exit can also verify logon information with information in the UADS or the RACF data base.

If the exit validates the logon information, it must set both the No UADS bit and the Don't Prompt bit on. In this case, the exit must also provide at least the following information:

– User ID
– Password
– Procedure name
– Account number (if your installation uses account numbers)
– System attributes
– User attributes (if your installation uses user attributes)
– Generic unit
– UPT

If the exit does not set the Don't Prompt bit on, the logon processor ignores the No UADS bit. The logon processor prompts the user and then validates the logon information with information in either the UADS or the RACF data base.

- Display your own logon panel.

  If your installation uses full-screen logon processing, the exit can display your own full-screen logon panel instead of the panel that TSO/E provides. After the exit displays your panel and verifies the input, it must set the No Full-Screen Panel bit on to prevent the logon processor from displaying the default full-screen panel. The exit must return the logon values it obtains from the user to the logon processor.

- Prevent a user from logging on to TSO/E.

  You can use the exit to prevent a user from logging on to TSO/E. The exit can perform various types of checking and, based on your own criteria, cancel logon processing by setting the Disconnect bit on. When the exit sets the Disconnect bit on, the logon processor terminates the logon session and sends no further messages to the user. Before the exit returns control to the logon processor, it can issue an explanation to the user, for example, using PUTLINE.

  Some reasons you may want to prevent a user from logging on are described below:

  – You may want to limit the number of logons according to groups of users. One way of doing this is to categorize your users into groups and define different naming conventions for user IDs for each group. Then the logon pre-prompt exit can check the user ID, determine how many users in that group are logged on, and then either cancel the logon session or allow the session to continue.

  – You may want to allow certain users or groups of users to log on only at certain times of the day.

Possible uses for IKJEFLD1 only

Some additional uses for IKJEFLD1 only are described below:

- Execute privileged instructions, such as changing the PSW key, and perform authorized functions, such as issuing RACF macros. For example, IKJEFLD1 can issue the RACINIT macro to perform RACF authority checking. IKJEFLD1 can then set the RACINIT control switch to prevent logon processing from issuing RACINIT.
- Supply the first TSO/E command to be executed in the user's session. IKJEFLD1 can supply an 80-byte TSO/E command to be executed by the TMP at the beginning of the session. This function is intended for installations that provide their own full-screen logon panel. If IKJEFLD1 does not set the Don't Prompt bit

and allows logon to display the full-screen logon panel, the first command is displayed on the panel and the user can modify or delete it. IKJEFLD1 must also set the First Command bit on.

For example, if IKJEFLD1 returned the MVSSERV command, an MVSSERV would begin when the user logged on to TSO/E, unless the user modified or deleted the MVSSERV command on the logon panel.

- Supply SYSOUT and job classes for the PSCB. IKJEFLD1 can specify the following classes to be used in the session:
  - Submit hold class
  - Submit class
  - Submit message class
  - SYSOUT class

  IKJEFLD1 must also set the appropriate control switch bits on. When IKJEFLD1 returns these job and SYSOUT classes, they are stored in the PSCB and remain in effect throughout the user's session.
- Supply the RBA of the user's mail directory within the broadcast data set. IKJEFLD1 can return the RBA, rather than request logon processing to obtain it from the UADS or RACF. IKJEFLD1 must set the No UADS bit on, and must also set the RBA bit on. The RBA is also available at logoff if you code logoff exit IKJEFLD2.
- Monitor how long the user's TSO/E session lasts.

  You can use logon exit IKJEFLD1 and logoff exit IKJEFLD2 to monitor the approximate duration of the user's TSO/E session. When IKJEFLD1 receives control, it can:
  - Invoke the TIME macro
  - Use the exit-to-exit communication word to return the time to the LOGON command processor. The exit updates the Key, Length, and Data fields for the exit-to-exit communication word as follows:

    Key X'01'

    Length
    Length of the data (time)

    Data Data (time).
  - Set a return code of 0 and return to the LOGON command processor.

  When the logoff exit IKJEFLD2 gets control, it receives the time from IKJEFLD1 in the exit-to-exit communication word. Before IKJEFLD2 returns control to LOGOFF, it can invoke the TIME macro. The exit can calculate the time difference between the time from the logon exit (in the exit-to-exit communication word) and the time it receives from issuing the TIME macro. The result is the approximate duration of the session. The logoff exit can include the processing time in a data set. You can then periodically print the data set and review the time calculations.
- Supply the primary and secondary language values for displaying translated information to users if your installation does not have RACF installed.
- Set up a console profile for the user.

Writing a logon pre-display exit (IKJEFLN1)

You can use the logon pre-display exit, IKJEFLN1, to further customize the logon process. This exit is invoked if:
- Logon language load modules are present in the LPA
- Exit IKJEFLD/IKJEFLD1 does not set the “Don't Prompt” control bit

The logon processor can invoke this exit several times during logon processing. If a re-prompt occurs for any data that is entered by the user, the logon processor invokes this exit.

You can write a pre-display exit to perform the following:
- Supply any data for IBM or installation-defined input fields
- Indicate that the logon processor should refresh fields in the 3270 outbound stream with data specified in the parameter entries defined for each field

TSO/E supplied exits

TSO/E does not provide a default logon pre-display exit routine.

Entry specifications

The contents of the registers on entry for the logon exit are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter list for IKJEFLN1

Exit IKJEFLN1 receives the address of the standard exit parameter list in register 1. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. IKJEFLN1 does not use the following standard parameter list entries:
- The command buffer pointed to at offset +0
- The new command buffer pointed to at offset +4
  - The values of these parameter entries are:

Key	X'00'
Length	X'04'
Data	X'00'

Figure 12 on page 115 shows the exit-dependent data that IKJEFLN1 receives starting at offset +36 (decimal) in the parameter list. The parameter entries are described following Figure 12 on page 115.

Key field meanings

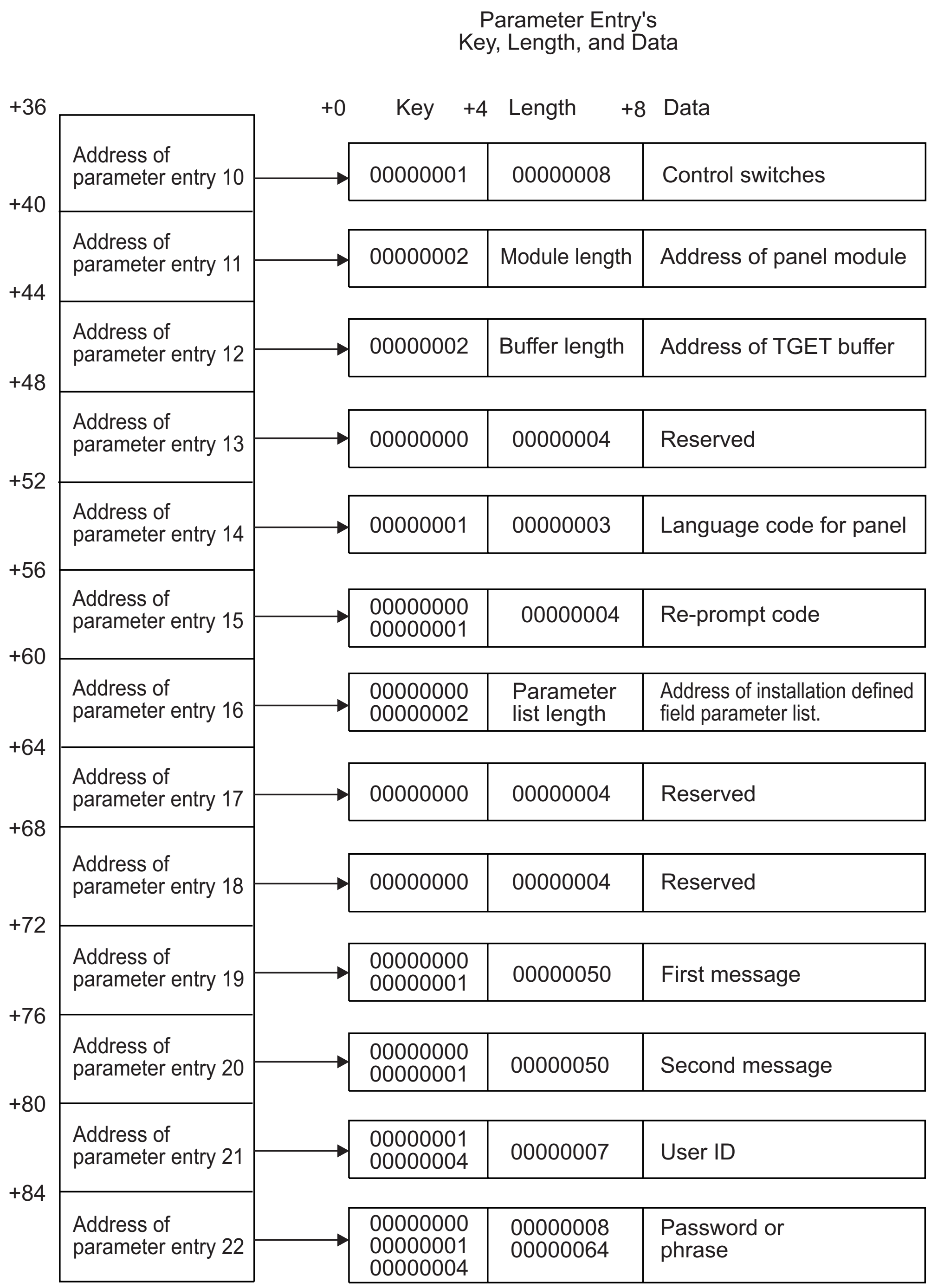
IKJEFLN1 uses the key field conventions described in Table 1 on page 34. Each parameter may or may not use all the key field settings depending on the field's function. Figure 12 on page 115 shows the acceptable values of the key field for each of the parameters.

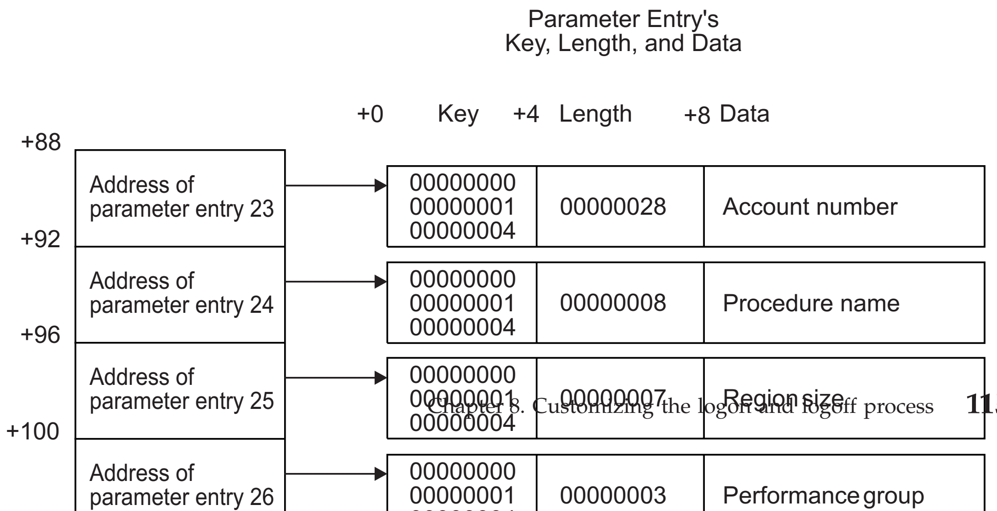
The IKJEFLN1 parameter list uses the key field value of X'04'. This value indicates that the field has been validated. After logon processing determines that the field is valid, it sets the key field to X'04' and locks the field to prohibit any additional prompts for data. If IKJEFLN1 validates a field and does not want logon processing to validate the field, it should set the key to X'04'.

Some fields must be validated by logon processing even if the exit sets the key to X'04'. The fields that logon processing must validate are:
- User ID
- Password
- New password
- RACF group ID
- SECLABEL

If this exit marks a field as valid that is not valid, logon processing will issue an abend 01A with a reason code of 36.

Note: The value in the "Data" column contains actual data or the address of the actual data depending on the value in the "Key" column. For a definition of the Key values, see Table 1 on page 34.


Parameter Entry's
Key, Length, and Data
+36
+40
+44
+48
+52
+56
+60
+64
+68
+72
+76
+80
+84
+0 Key +4 Length +8 Data
Address of parameter entry 10
00000001 | 00000008 | Control switches
Address of parameter entry 11
00000002 | Module length | Address of panel module
Address of parameter entry 12
00000002 | Buffer length | Address of TGET buffer
Address of parameter entry 13
00000000 | 00000004 | Reserved
Address of parameter entry 14
00000001 | 00000003 | Language code for panel
Address of parameter entry 15
00000000 00000001 | 00000004 | Re-prompt code
Address of parameter entry 16
00000000 00000002 | Parameter list length | Address of installation defined field parameter list.
Address of parameter entry 17
00000000 | 00000004 | Reserved
Address of parameter entry 18
00000000 | 00000004 | Reserved
Address of parameter entry 19
00000000 00000001 | 00000050 | First message
Address of parameter entry 20
00000000 00000001 | 00000050 | Second message
Address of parameter entry 21
00000001 00000004 | 00000007 | User ID
Address of parameter entry 22
00000000 00000001 00000004 | 00000008 00000064 | Password or phrase


Parameter Entry's
Key, Length, and Data
+88
+92
+96
+100
+0 Key +4 Length +8 Data
Address of parameter entry 23
00000000 00000001 00000004 | 00000028 | Account number
Address of parameter entry 24
00000000 00000001 00000004 | 00000008 | Procedure name
Address of parameter entry 25
00000000 00000001 00000004 | 00000007 | Region size
Address of parameter entry 26
00000000 00000001 | 00000003 | Performance group

Parameter descriptions

The following sections describe the parameters for the logon pre-display exit.

Control switches (parameter entry 10)

Control switches for logon exit IKJEFLN1 are described in Table 9.

Table 9. Control switches for logon exit IKJEFLN1

Byte	Bit	Field name	Set by
0	0	RACF/UADS	Logon processor
	1	DBCS	Logon processor
	2–7	Reserved	
1	0–7	Reserved	
2	0	IBM-defined field refresh	IKJEFLN1
	1	User-defined field refresh	IKJEFLN1
	2	Don't issue RACROUTE REQUEST=VERIFY	IKJEFLN1
	3–7	Reserved	
3	0–7	Reserved	
4	0	Don't Prompt	IKJEFLN1
	1	Invoke IKJEFLN2	IKJEFLN1
	2–7	Reserved	

RACF/UADS
If this bit is on, the source for the default information is from the RACF data base. If this bit is off, the source for the default information is the UADS data set.

DBCS
If this bit is on, the language code selected, either by or for the user, is a double-byte character set (DBCS) language. The 3270 data stream information can contain DBCS characters.

IBM-defined field refresh
You can set this bit on to request that the LOGON command refresh the IBM-defined fields in the 3270 outbound data stream with the values specified in the parameter entries.

User-defined field refresh
You can set this bit on to request that the LOGON command refresh the installation-defined fields in the 3270 outbound data stream with the values specified in the parameter entries.

Don't issue RACROUTE REQUEST=VERIFY
If IKJEFLN1 performs RACF processing and you do not want the logon processor to issue a RACROUTE REQUEST=VERIFY, IKJEFLN1 must set the “Don't issue RACROUTE REQUEST=VERIFY” bit on.

Don't Prompt
This bit should be set on if you want the logon processor to suppress display of the logon panel. When this bit is set, TSO/E will not prompt the TSO/E user for logon information.

`Invoke IKJEFLN2`
: This bit should be set on if you want the logon processor to invoke IKJEFLN2 even if the “Don't Prompt” bit is on. The “Don't Prompt” bit indicates that the logon processor should not prompt the TSO/E user.

Panel module address (parameter entry 11)

This parameter points to a local copy of the logon panel load module that resides below 16 MB in virtual storage. Any changes made to this field are ignored by the LOGON command.

TGET buffer address (parameter entry 12)

This parameter points to the input buffer returned from the TGET operation. You should not change this pointer. You can modify the data in the buffer, but any modifications must follow the format returned by TGET.

Reserved (parameter entry 13)

This parameter is reserved for TSO/E. Do not use this parameter entry.

Language code for the panel (parameter entry 14)

This parameter is the three-character code for the language used on the logon panel. This field is informational only. Any changes made to this field are ignored by the LOGON command.

Re-prompt code (parameter entry 15)

If the user enters data that is not valid and a re-prompt is necessary, this code identifies which field on the logon panel is in error.

On entry to the exit, this field indicates the reason for the prompt. A zero indicates that the prompting was for initial information. Other re-prompt codes are shown in Table 10.

When the logon processor issues a re-prompt, the field in error is highlighted. All fields except the highlighted field are locked and only the highlighted field can be changed.

The logon processor displays the appropriate error message for the field in error.

Table 10. Re-prompting codes for the logon processor

Code	Meaning
0	Initial prompt for data
1	User ID
2	Password
3	Account number
4	Procedure name
5	Region size
6	Performance group ID
7	New password
8	RACF group ID
9	No Mail option
10	No Notices option
11	Reconnect option
12	Operator ID card option
13	First command
14	SECLABEL

Table 10. Re-prompting codes for the logon processor (continued)

Code	Meaning
-1 to -n	Negative number of installation-defined parameters. If the exit sets the value to -1, the logon processor re-prompts for the first installation-defined field. If the exit sets the value to -2, the logon processor re-prompts for the second installation-defined field.

Installation panel parameters (parameter entry 16)

This parameter points to the installation-defined field parameter list. Each entry in the parameter list points to the parameter entry for an installation-defined field on the logon panel. The logon processor sets the key field of this parameter entry to X'02' if you have defined fields for your installation and to X'00' if you have not defined fields. If you have defined fields for your installation, the length field is set by the logon processor to the length of the installation panel field parameter list.

Installation-defined field parameter entries

On entry to the exit, the data field for each installation-defined field's parameter entry will be reset to what is in the field on the panel. This could be what was originally defined on the panel as default data, what the user entered at the LOGON prompt, or what a previous exit provided as data.

The length field for each installation-defined field's parameter entry will be set to the length of the data defined in the logon panel CSECT for the first invocation of IKJEFLN1. Should the user change the data for an installation-defined field on the LOGON panel, this length field will be updated to contain the length of the data entered on the LOGON panel, including trailing blanks or nulls if present. LOGON uses the length field for data movement from the exit's parameter list to the LOGON panel and vice-versa.

The example shown in Figure 13 on page 119 illustrates how the parameter list, pointed to by this parameter, is organized. You can use your defined parameter entries in exactly the same way as the IBM-defined parameter entries. You can edit and change information in these fields and request a re-prompt.

Note: The value in the "Data" column contains actual data or the address of the actual data depending on the value in the "Key" column. For a definition of the Key values, see Table 1 on page 34.

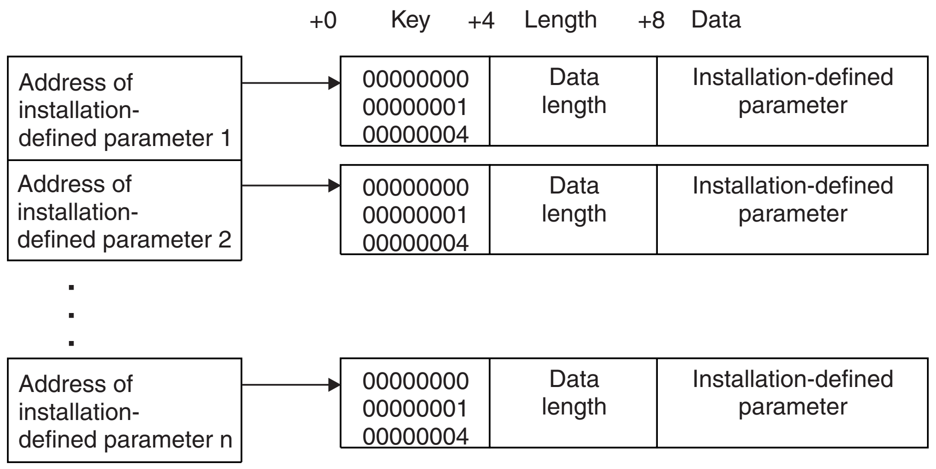

+0
Key
+4
Length
+8
Data
Address of installation-defined parameter 1
Address of installation-defined parameter 2
Address of installation-defined parameter n
00000000
00000001
00000004
Data length
Installation-defined parameter

Figure 13. Installation-defined parameter entries to IKJEFLN1 (pointed to by parameter entry 16)

Reserved

Parameter entries 17 and 18 are reserved for TSO/E. Do not use these two parameter entries.

First message

This parameter contains the text of the message that the logon processor will place in the first message area of the logon panel. You can change the message text in this exit using this parameter.

To use this parameter, the first message area must be present in the logon panel. If the first message area does not exist in the logon panel, the header for the first message is zero and this parameter is ignored.

A X'00' in the key field of this parameter entry signals the logon processor not to process the first message area parameter, even if the first message area exists in the logon panel.

If the key of this parameter entry is X'01', the logon processor will process the first message area. If there is text in this parameter, the text is placed in the first message area defined on the logon panel.

Second message

This parameter contains the text of the message that the logon processor will place in the second message area of the logon panel. You can change the message text using this parameter in this exit.

To use this parameter, the second message area must be present in the logon panel. If the second message area does not exist in the logon panel, the header for the second message is zero and this parameter is ignored.

A X'00' in the key field of this parameter entry also signals the logon processor not to process the second message area parameter, even if the second message area exists in the logon panel.

If the key of this parameter entry is X'01', the logon processor will process the second message area. If there is text in this parameter, the text is placed in the second message area defined on the logon panel.

User ID

This parameter contains the user ID that was specified by the user in the LOGON command. You can change the user ID parameter using this exit. The maximum length of the data is 7 bytes.

Password

Upon initial invocation of IKJEFLN1, this parameter does not contain any data. In subsequent invocations of IKJEFLN1, this parameter contains the password entered on the logon panel. If the IKJTSO*xx* parmlib member indicates PASSPHRASE(ON) it can contain a password phrase instead of a password. Phrases can be from 9 to 100 characters, while passwords can only be up to 8 characters. You can supply either a password or password phrase to the logon panel using this parameter. The maximum length of the field is 100 bytes.

Account number

Upon initial invocation of IKJEFLN1, this parameter contains either the value from the previous TSO/E session or the value specified on the LOGON command. Upon subsequent entry to IKJEFLN1, this parameter contains the account number entered on the logon panel. You can supply a different account number to the logon panel using this parameter. The maximum length of the field is 40 bytes.

Procedure name

Upon initial invocation of IKJEFLN1, this parameter contains either the value from the previous TSO/E session or the value specified on the LOGON command. Upon subsequent entry to IKJEFLN1, this parameter contains the logon procedure entered on the logon panel. You can supply a different logon procedure name to the logon panel using this parameter. The procedure contains JCL that defines the resources that the terminal job needs. The maximum length of the field is 8 bytes.

Region size

Upon initial invocation of IKJEFLN1, this parameter contains either the value from the previous T1O/E session or the value specified on the LOGON command. Upon subsequent entry to IKJEFLN1, this parameter contains the region size entered on the logon panel. You can supply a different region size to the logon panel using this parameter.

The region size is the number of kilobytes of main storage available for the TSO/E session. The maximum length of the field is 7 bytes. The maximum allowable region size is 2096128. The data contained in this parameter is in EBCDIC form.

Performance group

Upon initial invocation of IKJEFLN1, this parameter contains either the value from the previous TSO/E session or the value specified on the LOGON command. Upon subsequent entry to IKJEFLN1, this parameter contains the performance group entered on the logon panel. You can supply a different performance group to the logon panel using this parameter. The performance group associates the TSO/E session with a set of performance characteristics. The maximum length of the performance group is 3 bytes. The data contained in this parameter is in EBCDIC form.

New password

The new password field is used only if RACF is installed. Upon initial invocation of IKJEFLN1, this parameter does not contain any data. Upon subsequent entry to

IKJEFLN1, this parameter contains the new password entered on the logon panel. If the active IKJTSO*xx* parmlib member indicates PASSPHRASE(ON) it could contain a new password phrase, which can be from 9 to 100 characters. New passwords can only be up to 8 characters. You can supply either a new password or new password phrase to the logon panel using this parameter. The maximum length of the field is 100 bytes.

RACF group ID

The RACF group ID is used only if RACF is installed. Upon initial invocation of IKJEFLN1, this parameter contains either the value from the previous TSO/E session or the value specified on the LOGON command. Upon subsequent entry to IKJEFLN1, this parameter contains the RACF group ID entered on the logon panel. If your installation uses RACF, you can use this parameter to supply a different RACF group ID to the logon panel. The RACF group ID identifies the security group associated with this TSO/E session. The maximum length of the RACF group ID is 8 bytes.

No mail option

Upon initial invocation of IKJEFLN1, this parameter contains either the value from the previous TSO/E session or the value the user specified on the LOGON command. Upon subsequent entry to IKJEFLN1, this parameter contains the No Mail Option select character entered on the logon panel. You can use this parameter to supply a different No Mail Option select character to the logon panel. The maximum length of this field is one byte. The only allowable values are the character 'S' and ' '. The session receives previously saved mail from the SEND command during logon processing if this option is ' '. The session does not receive previously saved mail from the SEND command during logon processing if this option is 'S'.

No notices option

Upon initial invocation of IKJEFLN1, this parameter contains either the value from the previous TSO/E session or the value the user specified on the LOGON command. Upon subsequent entry to IKJEFLN1, this parameter contains the No Notices Option select character entered on the logon panel. You can use this parameter to supply a different No Notices Option select character to the logon panel. The maximum length of this field is one byte. The only allowable values are the character 'S' and ' '. The session receives broadcast messages from TSO/E during logon processing if this option is ' '. The session does not receive broadcast messages from TSO/E during logon processing if this option is 'S'.

Reconnect option

Upon initial invocation of IKJEFLN1, this parameter contains either the value from the previous TSO/E session or the value the user specified on the LOGON command. Upon subsequent entry to IKJEFLN1, this parameter contains the Reconnect Option select character entered on the logon panel. You can use this parameter to supply a different Reconnect Option select character to the logon panel. The maximum length of this field is one byte. The only allowable values are the character 'S' and ' '. If this option is 'S', TSO/E attempts to resume a logon session that was interrupted. If this option is 'S' and there is no disconnected session, the user is logged on normally. If this option is ' ', TSO/E starts a new logon session.

Operator ID card option

Upon initial invocation of IKJEFLN1, this parameter contains either the value from the previous TSO/E session or the value the user specified on the LOGON command. Upon subsequent entry to IKJEFLN1, this parameter contains the operator ID card select character entered on the logon panel. You can use this parameter to supply a different operator ID card select character to the logon panel. The maximum length of this field is one byte. The only allowable values are the character 'S' and ' '. If the field is ' ', the user is not prompted to enter the data using the Operator ID Card Facility. If the field is 'S', the user is prompted to enter the data using the Operator ID Card Facility.

First command

Upon initial invocation of IKJEFLN1, this parameter contains either the value from the previous TSO/E session or the value the user specified on the LOGON command. Upon subsequent entry to IKJEFLN1, this parameter contains the first command text that the user entered on the logon panel. You can supply first command text to the logon panel using this parameter. The maximum length of the first command is 80 bytes.

SECLABEL

SECLABEL is only used if RACF is installed and security label checking is active. Upon initial invocation of IKJEFLN1, this parameter contains either the value from the previous TSO/E session or the value the user specified on the LOGON command. Upon subsequent entry to IKJEFLN1, this parameter contains the RACF security label entered on the logon panel. If your installation uses RACF, you can use this parameter to supply a different RACF security label to the logon panel. SECLABEL is the RACF security label. The maximum length of the SECLABEL is 8 bytes.

Return specifications for IKJEFLN1

For IKJEFLN1, the contents of the registers on return must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for IKJEFLN1

Table 11 shows the return codes that IKJEFLN1 supports.

Table 11. Return codes for logon exit IKJEFLN1

Return code (decimal)	Description
0	Exit processing was successful. Logon processing continues.
12	Exit processing was unsuccessful. The logon processor issues an error message to the user and the console and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the LOGON command processor. For more information about the exit reason code, see “Exit reason code” on page 38.
16	Exit processing was unsuccessful. The logon processor terminates processing. The LOGON command processor does not display a message to the user or console if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

Note: If this exit issues return code 12 or 16, the post-display exit (IKJEFLN2) gets control. If you do not want the post-display exit (IKJEFLN2) to get control when IKJEFLN1 issues a return code of 12 or 16, then IKJEFLN1 should set the don't prompt bit on and the invoking of IKJEFLN2 should be turned off.

If the exit returns an undefined return code, the LOGON command processor terminates without displaying a message.

Programming considerations

The exit must follow standard linkage conventions. It must save the registers on entry and restore the registers when it returns. The exit must be reentrant, refreshable, and reusable.

The logon pre-display exit receives control only if the logon pre-prompt exit (IKJEFLD/IKJEFLD1) does not set the “Don't Prompt” control bit. TSO/E can invoke the logon pre-display exit several times on each invocation of the LOGON command.

Installing the exit

You must name the logon pre-display exit IKJEFLN1. Link-edit IKJEFLN1 as a separate load module. You can link-edit the exit in a separate, authorized load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exit can reside in:
- The link pack area (LPA)
- LNKLST

For more information about using the LPA or LNKLIST, see “Installing the standard-format exits” on page 39.

Environment

- State: Supervisor
- Key: 8
- AMODE(31), RMODE(ANY)
- Not APF-authorized
- Primary ASC mode

Possible uses

You can use this exit to supply information for the following fields on the logon panel:
- User ID
- Password
- Procedure name
- Account number (if used by the installation)
- Region size
- Performance group
- First command
- SECLABEL (defined for RACF users only)
- New password (defined for RACF users only)
- RACF group ID (defined for RACF users only)
- No Mail option
- No Notices option
- Reconnect
- Operator ID card
- Installation defined fields

Writing a logon post-display exit (IKJEFLN2)

You can use the post-display exit, IKJEFLN2, to further customize the logon process. This exit can receive control after the TSO/E user presses the Enter key on the logon panel display.

The logon processor invokes this exit if:
- The logon panel load modules are present in the LPA
- Exit IKJEFLD/IKJEFLD1 does not set the “Don't Prompt” control bit.

Note: If IKJEFLN1 sets the “Don't Prompt” bit, IKJEFLN2 will not be invoked unless IKJEFLN1 also sets the “Invoke IKJEFLN2” bit.

The logon processor can invoke this exit several times during logon processing. If a re-prompt occurs for any data that is entered by the user, the logon processor invokes this exit.

You can write a post-display exit to perform the following tasks:
- Update information contained on the logon panel
- Process any installation-defined fields
- Validate user-entered data
- Indicate that the LOGON command display a particular help panel
- Re-prompt for data

TSO/E supplied exits

TSO/E does not provide a default logon post-display exit routine.

Entry specifications

The contents of the registers on entry for the logoff exit are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address.

Parameter list for IKJEFLN2

Exit IKJEFLN2 receives the address of the standard exit parameter list in register 1. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32. Note that IKJEFLN2 does not use the following standard parameter list entries:
- The command buffer pointed to at offset +0
- The new command buffer pointed to at offset +4
  - The values of these parameter entries are:
    Key X'00'

Length	X'04'
Data	X'00'

Note: This exit should not be used to update the fields in the PSCB and UPT. You can update the fields of the PSCB and the UPT in the Logon Post-Prompt Exit (IKJEFLD3). Figure 14 on page 126 shows the exit-dependent data that IKJEFLN2 receives starting at offset +36 (decimal) in the parameter list. The parameter entries are described following Figure 14 on page 126.

Key field meanings

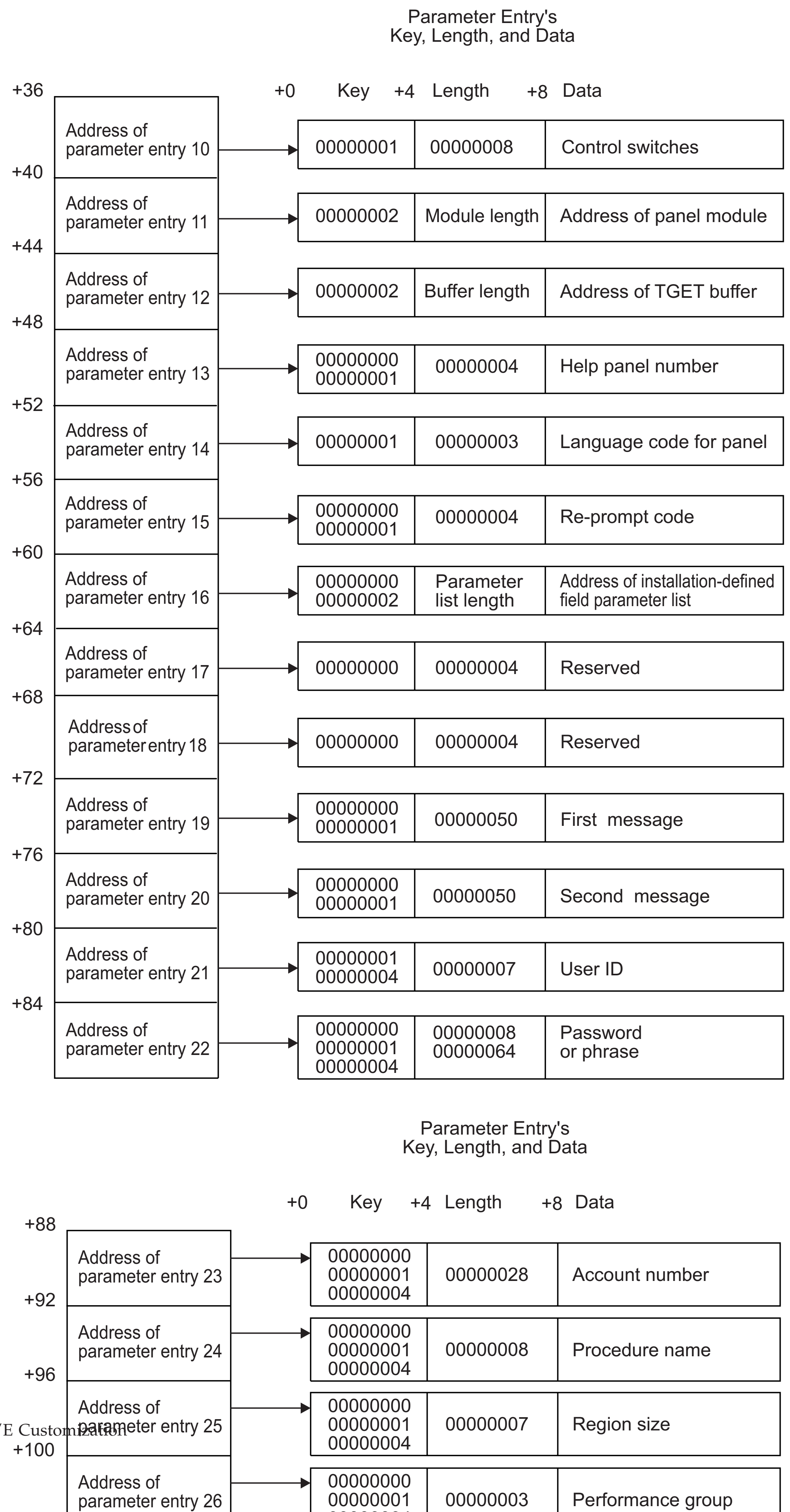
IKJEFLN2 uses the key field conventions described in Table 1 on page 34. Each parameter may or may not use all the key field settings depending on the field's function. Figure 12 on page 115 shows the acceptable values of the key field for each of the parameters.

The IKJEFLN2 parameter list uses the key field value of X'04'. This value indicates that the field has been validated. After logon processing determines that the field is valid, it sets the key field to X'04' and locks the field to prohibit any additional prompts for data. If IKJEFLN2 validates a field and does not want logon processing to validate the field, it should set the key to X'04'.

Some fields must be validated by logon processing even if this exit sets the key to X'04'. The fields that must be validated by logon processing are:
- User ID
- Password
- New password
- RACF group ID
- SECLABEL

If this exit marks a field as valid that is not valid, logon processing will issue an abend 01A with a reason code of 36.

Note: The value in the "Data" column contains actual data or the address of the actual data depending on the value in the "Key" column. For a definition of the Key values, see Table 1 on page 34.


Parameter Entry's
Key, Length, and Data
+0 Key +4 Length +8 Data
+36
Address of parameter entry 10
00000001 | 00000008 | Control switches
+40
Address of parameter entry 11
00000002 | Module length | Address of panel module
+44
Address of parameter entry 12
00000002 | Buffer length | Address of TGET buffer
+48
Address of parameter entry 13
00000000 00000001 | 00000004 | Help panel number
+52
Address of parameter entry 14
00000001 | 00000003 | Language code for panel
+56
Address of parameter entry 15
00000000 00000001 | 00000004 | Re-prompt code
+60
Address of parameter entry 16
00000000 00000002 | Parameter list length | Address of installation-defined field parameter list
+64
Address of parameter entry 17
00000000 | 00000004 | Reserved
+68
Address of parameter entry 18
00000000 | 00000004 | Reserved
+72
Address of parameter entry 19
00000000 00000001 | 00000050 | First message
+76
Address of parameter entry 20
00000000 00000001 | 00000050 | Second message
+80
Address of parameter entry 21
00000001 00000004 | 00000007 | User ID
+84
Address of parameter entry 22
00000000 00000001 00000004 | 00000008 00000064 | Password or phrase
Parameter Entry's
Key, Length, and Data
+0 Key +4 Length +8 Data
+88
Address of parameter entry 23
00000000 00000001 00000004 | 00000028 | Account number
+92
Address of parameter entry 24
00000000 00000001 00000004 | 00000008 | Procedure name
+96
Address of parameter entry 25
00000000 00000001 00000004 | 00000007 | Region size
+100
Address of parameter entry 26
00000000 00000001 | 00000003 | Performance group

Parameter descriptions

The following sections describe the parameters for the logon post-display exit.

Control switches (parameter entry 10)

Control switches for logon exit IKJEFLN2 are described in Table 12.

Table 12. Control switches for logon exit IKJEFLN2

Byte	Bit	Field name	Set by
0	0	RACF/UADS	Logon processor
	1	DBCS	Logon processor
	2–7	Reserved	
1	0–7	Reserved	
2	0	IBM-defined field refresh	IKJEFLN2
	1	User-defined field refresh	IKJEFLN2
	2	Don't issue RACROUTE REQUEST=VERIFY	IKJEFLN2
	3–7	Reserved	
3	0–7	Reserved	
4	0	Re-prompt	IKJEFLN2
	1	Help Text	IKJEFLN2
	2–7	Reserved	

RACF/UADS
If this bit is set on, the source for the default information is from the RACF data base. If this bit is off, the source for the default information is the UADS data set.

DBCS
If this bit is on, the language code selected, either by or for the user, is a double-byte character set (DBCS) language. The 3270 data stream information can contain DBCS characters.

IBM-defined field refresh
You can set this bit on to request that the LOGON command refresh the IBM-defined fields in the 3270 outbound data stream with the values specified in the parameter entries.

User-defined field refresh
You can set this bit on to request that the LOGON command refresh the installation-defined fields in the 3270 outbound data stream with the values specified in the parameter entries.

Don't issue RACROUTE REQUEST=VERIFY
If IKJEFLN2 performs RACF processing and you do not want the logon processor to issue a RACROUTE REQUEST=VERIFY, IKJEFLN2 must set the “Don't issue RACROUTE REQUEST=VERIFY” bit on.

Re-prompt
If you wish to re-prompt a TSO/E user for a field, you must set this bit on. The Re-prompt Code parameter (parameter 15) describes the field for which the re-prompt is necessary.

Help Text
If you wish to display help text, you must set this bit on. The logon processor then displays the help text as requested in the Help Panel Number parameter (parameter 13).

Panel module address (parameter entry 11)

This parameter points to a local copy of the logon language load module that resides below 16 MB in virtual storage. Any changes made to this field are ignored by the LOGON command.

TGET buffer address (parameter entry 12)

This parameter points to the input buffer returned from the TGET operation. You should not change this pointer. You can modify the data in the buffer, but any modifications must follow the format returned by TGET.

Help panel number (parameter entry 13)

This parameter entry allows you to request the display of a particular help panel. If the exit requests that a help panel be displayed, the exit should set the key field of the parameter entry to X'01' and the data field should specify the appropriate panel number (in binary). The numbers of the IBM-defined help panels for the LOGON command are listed in Table 17 on page 144. This exit must also set the help text control switch indicating that the LOGON command should display the help text (see Table 12 on page 127).

Language code for the panel (parameter entry 14)

The three-character code for the language used on the logon panel. This field is informational only. Any changes made to this field are ignored by the LOGON command.

Re-prompt code (parameter entry 15)

If the user enters data that is not valid and a re-prompt is necessary, this code identifies which field on the logon panel is in error.

On entry to the exit, this field indicates the reason for the prompt. A zero indicates that the prompting was for initial information. Other re-prompt codes are shown in Table 13.

You can set this code to indicate to the logon processor a field is in error. You must also set the re-prompt control switch. The logon processor will re-prompt the user for the specified field.

When the logon processor re-prompts the user for data, the field in error is highlighted. All fields except the highlighted field are locked and only the highlighted field can be changed.

The logon processor displays the appropriate error message for the field in error. If the code indicates a re-prompt for an installation-defined field, exit IKJEFLN2 can specify a message in parameter entry 19 or 20 that is displayed on the re-prompt.

Table 13. Re-prompting codes for the logon processor

Code	Meaning
0	Initial prompt for data
1	User ID
2	Password
3	Account number
4	Procedure name

Table 13. Re-prompting codes for the logon processor (continued)

Code	Meaning
5	Region size
6	Performance group ID
7	New password
8	RACF group ID
9	No Mail option
10	No Notices option
11	Reconnect option
12	Operator ID card option
13	First command
14	SECLABEL
-1 to -n	Negative number of installation-defined parameters. If the exit sets the value to -1, the logon processor re-prompts for the first installation-defined field. If the exit sets the value to -2, the logon processor re-prompts for the second installation-defined field.

Installation panel parameters (parameter entry 16)

This parameter points to the installation-defined field parameter list. Each entry in the parameter list points to the parameter entry for an installation-defined field on the logon panel. The logon processor sets the key field of this parameter entry to X'02' if you have defined fields for your installation and to X'00' if you have not defined fields. If you have defined fields for your installation, the length field is set by the logon processor to the length of the installation panel field parameter list.

Installation-defined field parameter entries

On entry to the exit, the data field for each installation-defined field's parameter entry will be reset to what is in the field on the panel. This could be what was originally defined on the panel as default data, what the user entered at the LOGON prompt, or what a previous exit provided as data.

The length field for each installation-defined field's parameter entry will be set to the length of the data defined in the logon panel CSECT for the first invocation of IKJEFLN1. Should the user change the data for an installation-defined field on the LOGON panel, this length field will be updated to contain the length of the data entered on the LOGON panel, including trailing blanks or nulls if present. LOGON uses the length field for data movement from the exit's parameter list to the LOGON panel and vice-versa.

The example shown in Figure 15 on page 130 illustrates how the parameter list, pointed to by this parameter, is organized. You can use your defined parameter entries in exactly the same way as the IBM-defined parameter entries. You can edit and change information in these fields and request a re-prompt.

Note: The value in the "Data" column contains actual data or the address of the actual data depending on the value in the "Key" column. For a definition of the Key values, see Table 1 on page 34.

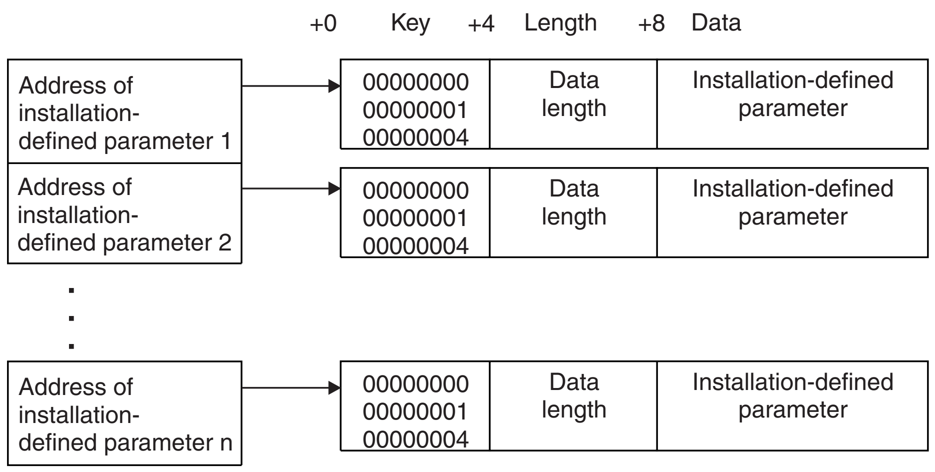

+0
Key
+4
Length
+8
Data
Address of installation-defined parameter 1
Address of installation-defined parameter 2
Address of installation-defined parameter n
00000000
00000001
00000004
Data length
Installation-defined parameter

Figure 15. Installation-defined parameter entries to IKJEFLN2 (pointed to by parameter entry 16)

Reserved

Parameter entries 17 and 18 are reserved for TSO/E. Do not use these two parameter entries.

First message

Using this parameter, you can provide message text that the logon processor will place in the first message area of the logon panel.

To use this parameter, the first message area must be present in the logon panel. If the first message area does not exist in the logon panel, the header for the first message is zero and this parameter is ignored.

A X'00' in the key field of this parameter entry signals the logon processor not to process the first message area parameter, even if the first message area exists in the logon panel.

If the key of this parameter entry is X'01', the logon processor will process the first message area. If there is text in this parameter, the text is placed in the first message area defined on the logon panel.

Second message

Using this parameter, you can provide message text that the logon processor will place in the second message area of the logon panel.

To use this parameter, the second message area must be present in the logon panel. If the second message area does not exist in the logon panel, the header for the second message is zero and this parameter is ignored.

A X'00' in the key field of this parameter entry signals the logon processor not to process the second message area parameter, even if the second message area exists in the logon panel.

If the key of this parameter entry is X'01', the logon processor will process the second message area. If there is text in this parameter, the text is placed in the second message area defined on the logon panel.

Note: If TSO/E needs to display a message, TSO/E chooses the first available message area. For example, if an exit has specified a message for the first message area, TSO/E will use the second message area to display a message.

User ID

This parameter contains the user ID that was specified by the user in the LOGON command. You can change the user ID using this parameter. The maximum length of the data is seven bytes.

Note: For all enterable fields on the logon panel, the values of the fields presented to this exit can be the values that the TSO/E user entered. These values can also be from the previous logon session.

Password

This parameter contains the password that is present on the logon panel. If the active IKJTSO*xx* parmlib member indicates PASSPHRASE(ON) it could contain a password phrase. Phrases can be 9 to 100 characters, while passwords can only be up to 8 characters. You can change the password or password phrase in the password field using this parameter. The maximum length of the field is 100 bytes.

Account number

This parameter contains the account number that is present on the logon panel. You can supply a different account number to the logon panel using this parameter. The maximum length of the field is 40 bytes.

Procedure name

This parameter contains the name of the logon procedure that is present on the logon panel. You can change the name of the logon procedure using this parameter. The procedure contains JCL that defines the resources that the terminal job needs. The maximum length of the field is 8 bytes.

Region size

This parameter contains the region size that is present on the logon panel. The region size is the number kilobytes of main storage available for the TSO/E session. You can change the region size using this parameter. The maximum length of the field is 7 bytes. The maximum allowable region size is 2096128. The data contained in this parameter is in EBCDIC form.

Performance group

This parameter contains the performance group name that is present on the logon panel. You can change the performance group name using this parameter. The performance group associates the TSO/E session with a set of performance characteristics. The maximum length of the performance group is 3 bytes. The data contained in this parameter is in EBCDIC form.

New password

For users who are defined to RACF, this parameter contains the new password specified on the logon panel. If the active IKJTSO*xx* parmlib member indicates PASSPHRASE(ON) it could contain a new password phrase. Phrases can be from 9 to 100 characters in length while passwords can only be up to 8 characters. You can change the new password or new password phrase in the new password field using this parameter. The maximum length of the filed is 100 bytes.

RACF group ID

For users who are defined to RACF, this parameter contains the RACF group identifier entered on the logon panel. You can change the RACF group name using this parameter. The RACF group ID identifies the security group associated with this TSO/E session. The maximum length of the RACF group ID is 8 bytes.

No mail option

This parameter contains the No Mail Option select character that is present on the logon panel. You can use this parameter to supply a different No Mail Option select parameter to the logon panel. The maximum length of this field is one byte. The only allowable values are the character 'S' and ' '. The session receives previously saved mail from the SEND command during logon processing if this option is ' '. The session does not receive previously saved mail from the SEND command during logon processing if this option is 'S'.

No notices option

This parameter contains the No Notices Option select character that is present on the logon panel. You can use this parameter to supply a different No Notices Option select parameter to the logon panel. The maximum length of this field is one byte. The only allowable values are the character 'S' and ' '. The session receives broadcast messages from TSO/E during logon processing if this option is ' '. The session does not receive broadcast messages from TSO/E during logon processing if this option is 'S'.

Reconnect option

This parameter contains the Reconnect Option select character that is present on the logon panel. You can change the value of the Reconnect Option select character using this parameter. The maximum length of this field is one byte. The only allowable values are the character 'S' and ' '. If this option is 'S', TSO/E attempts to resume a logon session that was interrupted. If this option is 'S' and there is no disconnected session, the user is logged on normally. If this option is ' ', TSO/E starts a new logon session.

Operator ID card option

This parameter contains the operator ID (OID) card value that is present on the logon panel. You can change the operator ID card value using this parameter. The maximum length of this field is one byte. The only allowable values are the character 'S' and ' '. If the field is ' ', the user is not prompted to enter the data using the Operator ID Card Facility. If the field is 'S', the user is prompted to enter the data using the Operator ID Card Facility.

First command

This parameter contains text present on the logon panel to indicate the first command is to be executed in the TSO/E session. The maximum length of the first command is 80 bytes.

SECLABEL

The parameter contains the SECLABEL present on the logon panel. SECLABEL is only used if RACF is installed and security label checking is active. This parameter contains the SECLABEL value that is present on the logon panel. You can change the RACF security label using this parameter. The maximum length of the SECLABEL is 8 bytes.

Return specifications for IKJEFLN2

For IKJEFLN2, the contents of the registers on return must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for IKJEFLN2

Table 14 shows the return codes that IKJEFLN2 supports.

Table 14. Return codes for logon exit IKJEFLN2

Return code (decimal)	Description
0	Exit processing was successful. Logon processing continues.
12	Exit processing was unsuccessful. The logon processor issues an error message to the user and console and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the LOGON command processor. For more information about the exit reason code, see “Exit reason code” on page 38.
16	Exit processing was unsuccessful. The logon processor terminates processing. The LOGON command processor does not display a message to the user or console if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the logon command processor terminates without displaying a message.

Programming considerations

The exit must follow standard linkage conventions. It must save the registers on entry and restore the registers when it returns. The exit must be reentrant, refreshable, and reusable.

The logon post-display exit receives control only if the logon pre-prompt exit (IKJEFLD/IKJEFLD1) does not set the “Don't Prompt” control bit. TSO/E can invoke the logon post-display exit several times on each invocation of the LOGON command.

Installing the exit

You must name the logon post-display exit IKJEFLN2. Link-edit IKJEFLN2 as a separate load module. You can link-edit the exit in a separate, authorized load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exit can reside in:
- The link pack area (LPA)
- LNKLST

For more information about using the LPA or LNKLST, see “Installing the standard-format exits” on page 39.

Environment

- State: Supervisor
- Key: 8
- AMODE(31), RMODE(ANY)
- Not APF-authorized
- Primary ASC mode

Possible uses

Some possible uses of this exit include the following:
- Processing fields on the logon panel
- Validating data on the logon panel
- Re-prompting for data
- Displaying help panels

Writing a logon post-prompt exit (IKJEFLD3)

The logon post-prompt exit (IKJEFLD3) allows an installation to modify logon processing. This exit receives control after the logon command processor has processed the LOGON command and full-screen panel information. This exit receives control in an authorized state and receives the standard TSO/E exit parameter list, including a 4-byte user word, if supplied at logon by a logon exit. The logon post-prompt exit routine can use the parameter list and user word to perform the following functions:
- Examine the JCL statements
- Modify the JCL statements
- Provide additional JCL statements
- Terminate the LOGON command
- Update fields in the PSCB and the UPT

TSO/E-supplied exits

TSO/E does not provide a default logon post-prompt exit routine.

Entry specifications

The contents of the registers on entry for the logon post-prompt exit are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter list for IKJEFLD3

Exit IKJEFLD3 receives the address of the standard exit parameter list. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32. IKJEFLD3 does not use the following standard parameter list entries:
- The command buffer pointed to at offset + 0
- The new command buffer pointed to at offset + 4

Figure 16 on page 135 shows the exit-dependent data that IKJEFLD3 receives starting at offset + 36 (decimal) in the parameter list. The parameter entries are described following Figure 16 on page 135.

Note: The value in the "Data" column contains actual data or the address of the actual data depending on the value in the "Key" column. For a definition of the Key values, see Table 1 on page 34.

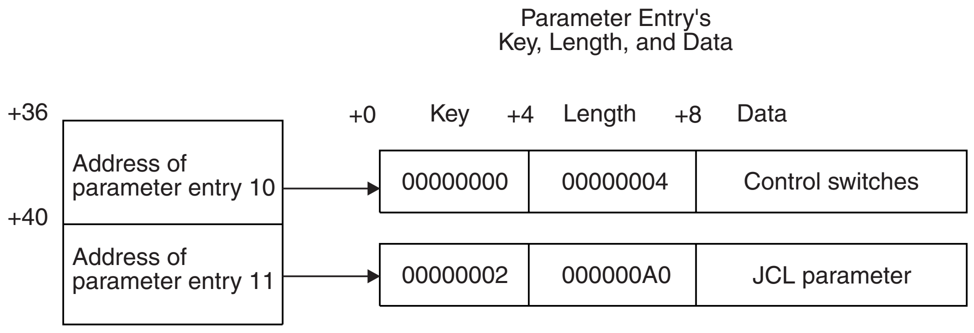


Figure 16. Exit-dependent data for the logon post-prompt exit IKJEFLD3

Parameter descriptions

The following sections describe the parameters for the logon post-prompt exit.

Control switches (parameter entry 10)

This parameter contains the control switches for logon post-prompt exit processing. Currently no control switches are used. You should not modify this parameter. The bit configuration is as follows.

Byte	Bit	Field name	Set by
0–3	0–7	Reserved	Reserved

JCL statements (parameter entry 11)

The exit receives the JCL statements built by the logon processor. The logon processor constructs JOB and EXEC statements. If the logon pre-prompt exit IKJEFLD or IKJEFLD1 changed the JCL parameter, this exit receives those JCL statements in place of the JCL statements the logon processor constructs. The JCL defines terminal job resources. The exit can modify the statements it receives, or it can add additional statements.

The JCL parameter is 800 bytes in length, in which the exit can return a maximum of ten JCL statements. To return more than ten JCL statements, the exit can provide its own JCL parameter. Your installation's JES capability determines the actual number of JCL statements you can return. Logon processing does not check that number. To provide your own JCL parameter, replace the parameter address supplied by the LOGON command with the address of the alternative parameter. Then set the current length to the number of JCL statements multiplied by 80.

The logon processor expects that each 80 bytes of the parameter contains a full 80-byte JCL statement in standard format. If your JCL statements are shorter than 80 bytes, pad them with blanks. The logon processor uses the current length field of the parameter to determine the number of statements that the exit is returning (current length/80 = n statements).

If you issue the GETMAIN macro to obtain storage for an alternative JCL parameter, you must use a job-oriented subpool (such as subpools 0–127). Storage is freed automatically when the logon job ends. Otherwise, you can use the user word to save the address so the logoff exit IKJEFLD2 can access it and issue the FREEMAIN macro to free storage in the subpool.

Return specifications

The contents of the registers on return from IKJEFLD3 must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for IKJEFLD3

Table 15 shows the return codes that IKJEFLD3 supports.

Table 15. Return codes for the logon exit IKJEFLD3

Return code (decimal)	Description
0	Exit processing was successful. Logon processing continues.
12	Exit processing was unsuccessful. The logon processor issues an error message to the user and console and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the LOGON command processor. For more information about the exit reason code, see “Exit reason code” on page 38.
16	Exit processing was unsuccessful. The logon processor terminates processing. The LOGON command processor does not display a message to the user or console if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the LOGON command processor terminates without displaying a message.

Programming considerations

The logon post-prompt exit must follow standard linkage conventions. It must save the registers on entry and restore the registers when it returns. The exit must be reentrant, refreshable, and reusable.

Environment

The attributes for the logon post-prompt exit are:
- State Supervisor
- Key: 8
- AMODE(24), RMODE(24) or AMODE(31), RMODE(ANY)
- Not APF-authorized

Installing the exit

You must name the logon post-prompt exit IKJEFLD3. Link-edit the exit as a separate load module. You can link-edit the exit in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exit can reside in:
- The link pack area (LPA)
- LNKLST

For more information about using the LPA or LNKLST, see "Installing the standard-format exits" on page 39.

Possible uses

Some possible uses of the logon post-prompt exit include the following:
- Examine the JCL statements
- Modify the JCL statements
- Provide additional JCL statements
- Terminate the LOGON command

Writing a logoff exit (IKJEFLD2)

Functional description

TSO/E users issue the LOGOFF command to end a session on TSO/E. This TSO/E installation exit lets you customize logoff processing for your users. The exit receives control before the LOGOFF command processor invokes the parse service routine to parse the command. The exit receives control in an authorized state, and receives the standard TSO/E exit parameter list, including a 4-byte user word, if any, supplied at logon by a logon exit. The logoff exit routine can use the parameter list and user word to perform the following functions:
- Free storage obtained by the logon exit
- Control re-logons
- Gather accounting information
- Control information written to the UADS data set

TSO/E-supplied exits

TSO/E does not provide a default logoff exit routine.

Entry specifications

The contents of the registers on entry for the logoff exit are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions

The logoff exit IKJEFLD2 receives the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. Note that IKJEFLD2 does not use the following standard parameter list fields:

- The new command buffer field pointed to at offset +4

Figure 17 shows the exit-dependent data that IKJEFLD2 receives beginning at offset +36 (decimal) in the parameter list. The exit-dependent parameter is reserved for future use and should not be modified.

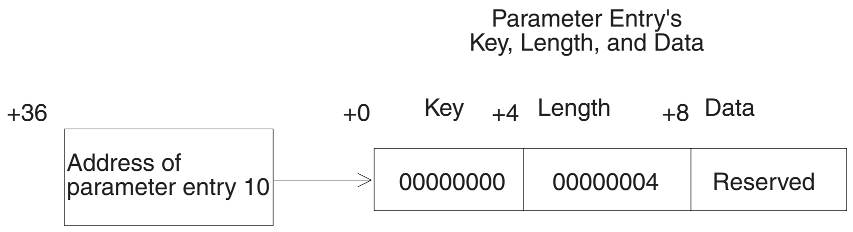


Figure 17. Exit-dependent data for the logoff exit IKJEFLD2

Reserved (Parameter Entry 10)
This parameter entry is reserved.

Return specifications

The contents of the registers on return from IKJEFLD2 must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes

Table 16 shows the return codes that the logoff exit supports.

Table 16. Return codes for the logoff exit IKJEFLD2

Return code (decimal)	Description
0	Exit processing was successful. Logoff processing continues.
12	Exit processing was unsuccessful. The logoff processor issues an error message to the user and the console, and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the logoff command processor. For more information about the exit reason code, see “Exit reason code” on page 38.
16	Exit processing was unsuccessful. The logoff processor terminates processing. The LOGOFF command processor does not display a message to the user or console if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the LOGOFF command processor terminates without displaying a message.

Either of the non-zero return codes or an undefined return code prevent the UADS data base from being updated.

Programming considerations

The logoff exit must follow standard linkage conventions. It must save the registers on entry and restore the registers when it returns. The exit must be reentrant, refreshable, and reusable.

Environment

The attributes for the logoff exit are:
- State: Supervisor
- Key: 8
- AMODE(24), RMODE(24) or AMODE(31), RMODE(ANY)
- Not APF-authorized

Installing the exit

You must name the logoff exit IKJEFLD2. Link-edit the exit as a separate load module. You can link-edit the exit in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exit can reside in:
- The link pack area (LPA)
- LNKLST

For more information about using the LPA or LNKLST, see "Installing the standard-format exits" on page 39.

Possible uses

Some possible uses of the logoff exit include the following:

- Clean up storage obtained by the logon exit

  Perform clean-up processing. For example, you can free storage that was obtained in the logon exit.
- Control re-logons

  The logoff exit receives the address of the command buffer containing the LOGOFF or LOGON command.

  To check the command buffer and change its contents, the logoff exit can:
  - Scan the command buffer and decide, based on your own criteria, to change the command that the user issued
  - Supply or change values that users specify on the command.

  The exit can change the information in the input buffer, update the current length of the input buffer, and return control to the logon processor. The values that the exit can provide in the input buffer correspond to the operands of the LOGOFF or LOGON command. For more information about the LOGOFF and LOGON commands, see *z/OS TSO/E Command Reference*.
- Gather accounting information

  The logoff exit can gather and report accounting information for the user's TSO/E session. For example, the logoff exit can obtain the account number and calculate the duration of the user's TSO/E session, based on information passed from logon exit IKJEFLD1 using the exit-to-exit communication word.

  To record the account number and the session's duration, the logon exit IKJEFLD1 can:
  - Obtain the account number from the LOGON command or panel.
  - Invoke the TIME macro.
  - Use the exit-to-exit communication word to return the time and account number to the LOGON command processor. The exit updates the Key, Length, and Data fields for the exit-to-exit communication word as follows:

Key X'01'

Length
Length of the data (time and account number)

Data Data (time and account number)

– Set a return code of 0 and return to the logon command processor.

When the logoff exit IKJEFLD2 receives control, it obtains the time and account number from IKJEFLD1 in the exit-to-exit communication word. Before IKJEFLD2 returns control to the logoff command processor, it can invoke the TIME macro. The exit can calculate the time difference between the time from the logon exit (in the exit-to-exit communication word) and the time it receives from issuing the TIME macro. The result is the approximate duration of the session. The logoff exit can record the account number and processing time in a data set for accounting purposes.

- Determine the userid

  The userid for the logoff exit is found at offset 0 in the protected step control block (PSCB). The PSCB contains the userid in PSCBUSER, a seven-character field which is padded on the right with blanks. The length of the userid is found in the following one byte field PSCBUSRL.

  For a description of the PSCB, see *z/OS TSO/E System Diagnosis: Data Areas*.
- Provide information to be written to the UADS or RACF data base.

  The logoff exit can update fields of the PSCB and UPT, whose addresses are passed in the exit parameter list. The logoff command processor then writes the contents of those fields back to the UADS or the RACF data base.

Customizing logon panels and logon help panels

You can customize logon panels and logon help panels in a variety of ways, including adding and removing fields.

You can use the source of logon panel modules and logon help panel modules to suit the needs of your installation. You can also create your own panels.

The source for the logon panel module (IKJLPENU), the logon panel module with password phrase support (IKJLQENU), and the logon help panel module (IKJLHENU) is included in SYS1.SAMPLIB. Logon panel load modules are installed in SYS1.LPALIB. Logon help panel load modules are installed in SYS1.LINKLIB.

To install a logon panel load module that you have customized, you must code and assemble the logon panel csect and link it into the LPA library.

A logon panel load module consists of a single csect divided into two parts. The first part is header information that contains offsets of the fields defined in the second section. The second part of the csect is the 3270 data stream instructions that comprise the logon panel.

The logon help panel load module consists of the same two parts, but the header does not contain address attributes.

If your installation has a language feature installed, logon and logon help panels are available in that language. You can customize logon panels and logon help panels for a particular language by using logon panel and logon help panel csects. The source for the logon panel csect is in SYS1.SAMPLIB. Chapter 20, "Customizing TSO/E for different languages," on page 185 describes how you can select a language for your installation or for users in your installation.

Functions activated by the presence of logon load modules

If the LPA contains the load modules for the logon panels, these panels are used and TSO/E invokes exits IKJEFLN1 and IKJEFLN2. If a logon panel load module is not present in the LPA, TSO/E does not invoke exits IKJEFLN1 and IKJEFLN2.

The presence of a logon panel load module in the LPA signals TSO/E to re-prompt the user for the new password when the user enters data in the new password field. The logon processor re-displays the logon panel requesting the user to enter the password again for verification. For further information on the re-prompt feature, see *z/OS TSO/E Command Reference*.

Logon panel

You can have any number of logon panel load modules in your LPA library. However, there must be one load module for each language used by the installation. The names of the load modules and csects containing the definitions of the logon panels are constructed as IKJLPXXX where XXX is the three-character code for the language. When password phrase support is active, the panels are named IKJLQxxx where XXX is the code for the language.

For example, if a TSO/E user has been assigned mixed case U.S. English as their selected language and invokes the LOGON command, the logon process searches for IKJLPENU in the LPA. If password phrase support is active, it searches for IKJLQENU instead.

Logon panel CSECT information

The following DSECT maps the start of the logon panel CSECT.

```
LOGONPAN  DSECT
PANID     DS     CL8                    Acronym of this csect
PANDATE   DS     CL8                    Date of assembly
PANTIME   DS     CL8                    Time of assembly
PANLEN    DS     AL2                    Length of this csect
PANVERS   DS     AL2                    Version of this panel
PANHEAD   DS     AL2                    Offset to header offset array
PANBEGIN  DS     AL2                    Offset to start of panel
PANMSG1   DS     AL2                    Offset to message 1
PANMSG2   DS     AL2                    Offset to message 2
PANICOFF  DS     AL2                    Offset to Insert Cursor Address
PANINSTH  DS     AL2                    Offset to first installation-  *
                                        defined field
PANLENGTH DS     AL2                    Length of panel
PANWCC    DS     AL2                    Offset to Write Cursor Control
PANRACFT  DS     AL2                    Offset to RACF Title
PANRACFL  DS     AL2                    Length of RACF Title
```

Logon panel CSECT header

Following the logon panel CSECT start information, the logon CSECT header contains an array of offsets to the 3270 data stream instructions. The offsets are contained in halfword addresses. Each field in the header contains the following offsets:

Highlighting Attribute Offset
This is the offset to the highlighting attribute instructions for the input field. For a re-prompt, the LOGON command processor highlights the input field and does so only on fields that have this attribute defined.

Highlighting Character Offset
This is the offset to a position on the panel to insert a highlighting character. The LOGON command processor highlights a field by inserting an asterisk in the byte that is pointed to by this offset. For a re-prompt, the LOGON command processor inserts an asterisk in this field if this offset is present.

Field Attribute Offset
This is the offset to the field attribute instructions for the input field. The LOGON command processor uses this field to protect and unprotect a field and does so only on fields that have this attribute defined.

Address of Start of Field
This is the 12-bit address that identifies the starting point of the text on the screen. The 12-bit addresses are described in *3270 Information Display System, Buffer Address Codes*. The LOGON command uses this address to identify the input fields that are returned in the TGET buffer.

Length of Input Field
A 2-byte length field of the input area.

Offset to Input Field
A 2-byte offset to the input field.

Reserved
Two reserved halfwords are included in each entry.

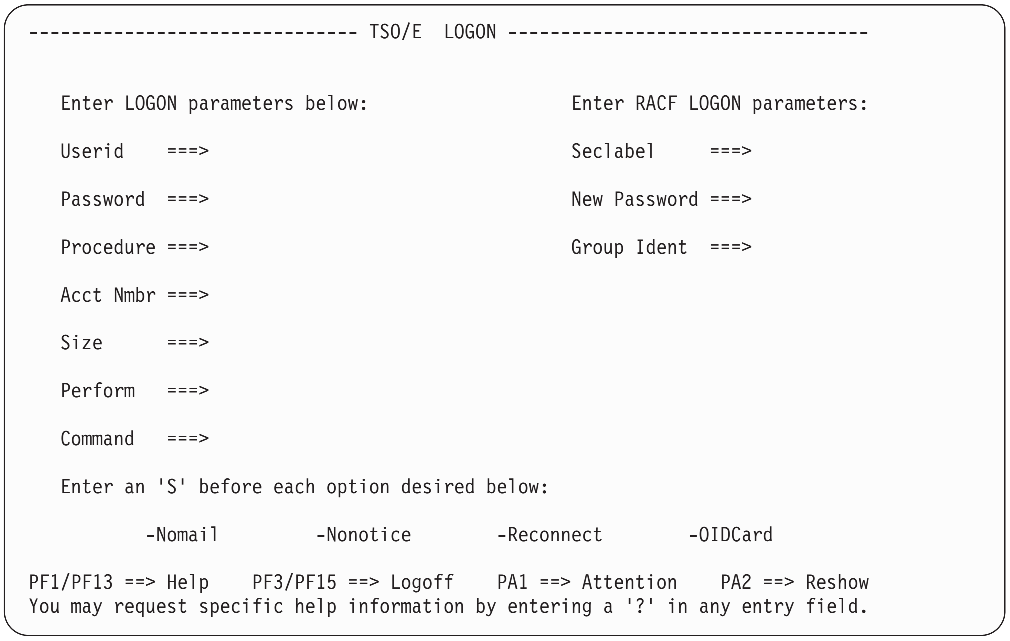
The size of the header for each header element of the array is 16 bytes. The IBM-defined fields have the following header order. They are also in the same order in the 3270 data stream instructions that follow the header:
- User ID
- Password
- Account number
- Procedure name
- Region size
- Performance group ID
- New password
- RACF group ID
- No Mail option
- No Notices option
- Reconnect option
- Operator ID card option
- First command
- SECLABEL
- Installation-defined fields (each entry in the header must be 16 bytes)
- End Header flag (16 bytes of X'FF')

Installations can insert as many installation-defined fields as needed. The fields should conform to the format described above.

Logon panel body

The logon panel body consists of the data stream instructions that are sent to a 3270 device. These data stream instructions contain the IBM-defined fields and the installation-defined fields. The data stream and the output screen fields can be in any order as long as the header entries are in the correct order.

The source for the default logon panel module for mixed case U.S. English is included in SYS1.SAMPLIB and is called IKJLPENU. The logon panel that results from this module is shown in Figure 18 on page 143.



```
------------------------------ TSO/E  LOGON ---------------------------------


   Enter LOGON parameters below:                  Enter RACF LOGON parameters:

   Userid    ===>                                 Seclabel     ===>

   Password  ===>                                 New Password ===>

   Procedure ===>                                 Group Ident  ===>

   Acct Nmbr ===>

   Size      ===>

   Perform   ===>

   Command   ===>

   Enter an 'S' before each option desired below:

           -Nomail         -Nonotice        -Reconnect        -OIDCard

PF1/PF13 ==> Help    PF3/PF15 ==> Logoff    PA1 ==> Attention    PA2 ==> Reshow
You may request specific help information by entering a '?' in any entry field.
```

Figure 18. Sample logon panel

Logon help panel

If you are using logon help panel load modules, you can use the help facility to specify help for the entire panel or for a particular field.

There is one load module per language. The load module name is IKJLH*XXX*, where *XXX* is the three-character code for the language. For more information on three-character codes, see *z/OS MVS Programming: Assembler Services Guide*.

Logon help panel csect information

The help panel CSECT is similar to the primary logon panel CSECT, but the size of the header is smaller. The following DSECT maps the start of the help panel CSECT.

```
HELPPAN  DSECT
HPANID    DS     CL8                    Acronym of this csect
HPANDATE  DS     CL8                    Date of assembly
HPANTIME  DS     CL8                    Time of assembly
HPANLEN   DS     AL2                    Length of this csect
HPANVERS  DS     AL2                    Version of this panel
HPANHEAD  DS     AL2                    Offset to header offset array
HPANINST  DS     AL2                    Offset to first installation - *
                                        defined field
```

Following the start information, the logon help panel csect header contains an array of offsets to the 3270 data stream instructions. The offsets are contained in halfword addresses. The size of the header is 12 bytes and consists of:

`Start of Help Text Offset`
: This is the offset to the start of the help text for the field for which help text is requested.

`Length of Help Text`
: This is the length of the help text associated with the field.

Four Reserved Halfwords
The next 4 halfwords are reserved.

The headers pointing to help text for IBM-defined fields have the same order as the headers defining the logon panel csects. The end header flag is 12 bytes of X'FF' instead of 16 bytes as in the logon panel csect.

If the user requests help text for a field and the corresponding help text header entry is 0, no help text is displayed. A message is also displayed on the logon panel.

The body of the help panel csect contains the 3270 data stream instructions necessary to display the help text for a selected field. The help text fields can be in any order as long as the header entries are in the correct order.

Invoking the help panel

When help text is selected, TSO/E uses a convention similar to logon panel processing to determine the presence or absence of help panel load modules. IKJLH is concatenated to the three-character language code.

Help panel number codes

Exit IKJEFLN2 can specify a help panel code to indicate which help text should be displayed. The help panel codes are described in Table 17.

Table 17. Help panel number codes for logon

Code	Meaning
0	Display All Help Text
1	User ID Help Text
2	Password Help Text
3	Account Number Help Text
4	Procedure Name Help Text
5	Region Size Help Text
6	Performance Group ID Help Text
7	New Password Help Text
8	RACF Group ID Help Text
9	No Mail Option Help Text
10	No Notices Option Help Text
11	Reconnect Option Help Text
12	Operator ID Card Option Help Text
13	First Command Help Text
14	SECLABEL Help Text
-1 to -n	Installation-defined Help Text

Programming considerations for logon and logon help panel csects

To use DBCS characters in a panel field, you should code the Start Field Extended Instruction (X'29') with the Shift-In Shift-Out Attribute pair. The pair is coded X'FE01'.

If you are sending panels to a Katakana device, note that lower case English characters become Katakana characters on Katakana devices. If you want to use English Language panels, those panels should contain uppercase English characters only.

Chapter 9. Defining TSO/E to ISPF and ISPF/PDF

TSO/E users can use Interactive System Productivity Facility/Program Development Facility (ISPF/PDF) panels to perform a variety of data processing tasks, such as managing data sets and developing programs. They can use TSO/E commands from ISPF/PDF panels to help perform some of those tasks.

This section is an overview of how to define TSO/E to ISPF and ISPF/PDF. You must write a logon procedure to give TSO/E users access to ISPF and ISPF/PDF. You can allow users to use TSO/E commands and Session Manager from ISPF/PDF panels.

Customizing the logon procedure for ISPF and ISPF/PDF

To use ISPF and ISPF/PDF, TSO/E users must have access to ISPF and ISPF/PDF data sets. You must modify the TSO/E logon procedure to allocate the required ISPF and ISPF/PDF data sets.

Before you modify the logon procedure, however, you must decide how users will access ISPF/PDF at your installation. To give users access, you can display the primary ISPF/PDF panel first when users log on, or display a panel that gives users access to the primary ISPF/PDF panel. For example, you could display the Information Center Facility panel as the first panel, giving users indirect access to ISPF/PDF. Instead of displaying a panel, you could simply allow users to invoke ISPF/PDF from TSO/E READY mode in response to the READY message.

In the logon procedure, use the EXEC statement to point to a CLIST or REXX exec that starts ISPF and displays either the primary ISPF/PDF panel, or a panel that gives users access to ISPF/PDF. For information about writing a logon procedure for TSO/E, see "Writing a logon procedure" on page 78. For more information about how to write a logon procedure to allocate ISPF data sets, see *z/OS ISPF Planning and Customizing*.

Specifying the TSO/E commands users can issue from ISPF/PDF

You can customize the set of TSO/E commands that users can issue from ISPF/PDF panels. Users can issue commands under ISPF/PDF:
- On the command line, preceded by "TSO"
- From ISPF/PDF option 6
- Using the ISPF command SELECT CMD (command-procedure)

You can customize the set of TSO/E commands that all users, or individual users, can issue from ISPF/PDF panels. By default, users can use all TSO/E commands that reside in LPA (link pack area) except the following:
- PRINTDS
- RACONVRT
- SYNC
- TEST
- LOGON
- LOGOFF

To allow users to use those commands, add them to the ISPF module ISPTCM. You can also restrict users from using commands from ISPF/PDF by deleting them from ISPTCM. In addition, you can use RACF to restrict these commands in all environments, including ISPF/PDF. For more information, see *z/OS Security Server RACF Command Language Reference*.

You can use the ISPMTCM macro statement to add or delete commands from ISPTCM.

Commands that do not reside in LPA can be used from ISPF/PDF panels, regardless of whether they are in ISPTCM.

If the following commands reside in LPA, you must indicate in ISPTCM that they be checked for authorization so that they receive control in an authorized state:
- RACONVRT
- SYNC
- SEND
- LISTBC
- SUBMIT

To override the defaults in ISPTCM for individual users, use ISPF/PDF command start and stop exits. Using these exits, you can monitor and restrict the commands individual users issue. For more information about the ISPMTCM macro statement and the ISPF exits, see *z/OS ISPF Planning and Customizing*.

Defining the TSO/E Session Manager to ISPF/PDF

There are two ways to use Session Manager: from TSO/E READY mode or from ISPF/PDF. By default, users who log on to TSO/E with the Session Manager procedure can keep a journal of the work they do from READY mode only.

There is a limitation with using Session Manager from READY mode. After a user has exited Session Manager, the user cannot reinvoke Session Manager without first logging off and logging back on. Therefore, users cannot easily alternate between Session Manager mode and non-Session Manager mode.

To overcome this limitation, you can allow users to use Session Manager from ISPF/PDF. A user can issue a TSO/E command, CLIST or REXX exec from an ISPF/PDF panel, log the output, then return to ISPF/PDF. To allow users to log output from commands, CLISTs or REXX execs, they issue, or from different language processors, you must:
- Install exit routines provided with ISPF for SVC 93 and 94
- Update and reassemble the ISRCONFG CSECT, setting the SESSMNGR value to YES

For more information about defining Session Manager to ISPF/PDF, see *z/OS ISPF User's Guide Vol I*.

Chapter 10. Considerations for using TSO/E with z/OSMF ISPF

This section is an overview of how to setup TSO/E address spaces for z/OS Management Facility (z/OSMF) ISPF.

ISPF task overview

The ISPF task allows you to access your host system ISPF applications from z/OSMF. For system administrators, the ISPF task provides a Web-based alternative to using traditional, 3270 based ISPF.

Through the ISPF task, you can:
- Access any applications that you usually access through z/OS ISPF on the host system, such as System Display and Search Facility (SDSF) and Hardware Configuration (HCD)
- Run TSO commands
- Use multiple sessions in parallel (split screen mode)
- Switch to TSO mode, for example, to check job output
- Customize the ISPF settings as you do with ISPF on the host system
- Use special keys. such as Attention (PA1)
- Use dynamic areas in ISPF and attributes such as color highlighting
- Use ISPF line mode applications (for example, ISPF option 3).

The ISPF task works with ISPF on your host z/OS system. User access to ISPF applications is controlled through the same authorizations that exist for your z/OS system.

For more information see *IBM z/OS Management Facility Configuration Guide*.

System prerequisites for the ISPF plug-in

This section describes the system setup actions required for using the ISPF task.

To use the ISPF task, a user should be an existing TSO/E user with a valid, non-expired password.

For each user of the ISPF task, you must ensure that the corresponding user ID:
- Is authorized to TSO/E
- Is authorized to the JES spool. This authorization allows the user to use various functions in TSO/E, such as the SUBMIT, STATUS, TRANSMIT, and RECEIVE commands, and to access the SYSOUT data sets through the command TSO/E OUTPUT command.
- Has an OMVS segment defined, which allows for access to z/OSMF
- Has a home directory defined, which is required for z/OSMF.

By default, the ISPF task is setup to use the logon procedure IKJACCNT, which is supplied by IBM. A user can select to use a different logon procedure, as long as the user's logon procedure is properly configured for ISPF.

Customizing for profile sharing

Some TSO users require the use of multiple ISPF sessions. For example, a user might need to:

- Log on simultaneously through a z/OSMF ISPF session and a telnet 3270 session, or
- Log on through multiple z/OSMF ISPF sessions (this is different than having split screens, which is also allowed).

To have multiple ISPF sessions, the user's logon procedure must be configured to allow profile sharing. With profile sharing enabled, the user's logon procedure is required to allocate ISPF profile data sets with the disposition SHARED, rather than NEW, OLD, or MOD, and the data sets must already exist. Or, these data sets must be temporary data sets. For more information, see the topic on profile sharing in *z/OS ISPF Planning and Customizing*.

Security setup for the ISPF plug-in

During the configuration process, z/OSMF creates a REXX exec with commands for your security administrator to use for controlling access to the ISPF task. For the commands that are created, see Commands for configuring the ISPF plug-in in the *IBM z/OS Management Facility Configuration Guide*. The z/OSMF configuration process also creates a commands exec for your security administrator to use for authorizing users to this and other z/OSMF tasks; see Authorizing users to z/OSMF in the *IBM z/OS Management Facility Configuration Guide*.

Also, you must assign the TRUSTED attribute to the common event adapter (CEA) started task if you have not done so already. Doing so will allow the CEA address space to bypass authorization checking and to access or create any resource it needs. For more information, see the topic on associating started procedures and jobs with user IDs in *z/OS Security Server RACF System Programmer's Guide*, and the topic on using started procedures in *z/OS Security Server RACF Security Administrator's Guide*. For information about how to configure CEA, see *z/OS Planning for Installation*.

Usage considerations for ISPF task users

Some TSO/E and ISPF functions are restricted or unavailable under z/OSMF ISPF. Users should be aware of the following usage considerations:

- z/OS creates an address space for each ISPF task session that is started. An individual z/OSMF user can have up to ten active ISPF task sessions. To conserve system resources, z/OSMF allows no more than 50 ISPF task sessions to be started on your system at any one time.
- In some situations, logon pre-prompt exits IKJEFLD and IKJEFLD1 that set the Don't Prompt control switch bit on can prevent z/OSMF ISPF users from logging on, or might not work with z/OSMF ISPF.
- An ISPF task user cannot:
  - Switch to TSO/E native mode from within a z/OSMF ISPF session.
  - Log in remotely to TSO/E on another z/OS system from z/OSMF ISPF.
  - Use full-screen applications that run outside of ISPF, such as OMVS, TELNET, or GDDM®.
  - Receive TSO/E messages, such as message from MVS operators or users in TSO/E native mode.
  - Use commands that are not allowed in traditional ISPF, such as TSOLIB and LOGON.
- Most VTAM terminal macros used by full screen applications, such as GTTERM or STFSMODE, are not supported under z/OSMF ISPF. However, you can use the GTSIZE macro or GETDEVSZ macro to obtain the screen size.

- Broadcast messages are not displayed at log on.
- Session Manager is not available; do not specify ADFMDF03 in your logon procedure. Your logon procedure should use the IBM-supplied terminal monitor program, IKJEFT01, which is specified on the PGM= operand of the EXEC statement.
- The REXX and CLIST system terminal ID (SYSTERMID) variable is blank for z/OSMF ISPF task sessions.

Chapter 11. Specifying authorized commands/programs, and commands not supported in the background

At your installation, you must maintain lists of authorized commands and programs, and commands not supported in the background. The lists allow users to execute authorized commands and programs, and restrict users from executing certain commands in background jobs. IBM provides copies of the lists, including all required entries. Table 18 shows which entries are required in each list.

Table 18. Required entries in the lists of commands and programs

Maintain in the list of:	These names:
Authorized commands	RECEIVE TRANSMIT XMIT SEND SE CONSPROF LISTBC LISTB RACONVRT SYNC TESTAUTH PARMLIB
Programs to be authorized when called through the TSO/E service facility	IKJEFF76
Commands not supported in the background	OPERATOR OPER TERMINAL TERM

You might want to add commands and programs to the lists. You can add authorized commands and programs, such as VLFNOTE, LISTDS, IEHMOVE, and user-written ones, to make them available to users at your installation. You can also add commands that you do not want used in the background, such as:

- User-written commands that do not work properly in the background (for example, command processors that issue TPUTs and TGETs)
- Commands whose use you plan to restrict in the foreground. The ACCOUNT command is one example of such a command. You might want to restrict its use as described in "Limiting the use of the ACCOUNT, OPERATOR, RACONVRT, and SYNC commands" on page 176.

For ISPF, you must ensure that the authorized commands receive control in an authorized state under ISPF. For more information, see *z/OS ISPF Planning and Customizing*.

You can maintain the lists of authorized commands and programs, and commands not supported in the background, in one of the following:

- SYS1.PARMLIB member IKJTSOxx
- CSECTs IKJEFTE2, IKJEFTE8, IKJEFTAP, and IKJEFTNS.

Using SYS1.PARMLIB member IKJTSOxx has certain advantages:

- It makes it easy to maintain the lists of commands and programs. You avoid having to update, assemble, and link-edit the CSECTs IKJEFTE2, IKJEFTE8, IKJEFTAP, and IKJEFTNS.
- You can use the PARMLIB command if you want to use a different list of authorized commands/programs, and commands not supported in the background, without re-IPL of the system.

  The PARMLIB command dynamically activates a SYS1.PARMLIB member IKJTSOxx of your choice. Its specifications are replaced by those in member IKJTSOxx of your choice. The chosen SYS1.PARMLIB member IKJTSOxx must at least contain the required entries shown in Table 18 on page 151.

  Use the PARMLIB command also to check the syntax of any IKJTSOxx member of SYS1.PARMLIB before you are going to use it, and to view the specifications in an active IKJTSOxx member. For more information about the PARMLIB command, see *z/OS TSO/E System Programming Command Reference*.

Using CSECTS IKJEFTE2, IKJEFTE8, IKJEFTAP, and IKJEFTNS does not allow for dynamic changes.

When you IPL the system, the commands and programs listed in either SYS1.PARMLIB member IKJTSOxx or in the individual CSECTs become available or restricted as specified.

If you maintain the lists of authorized commands and programs, and commands not supported in the background, in both the SYS1.PARMLIB member IKJTSOxx and the CSECTS the following applies:

- If IKJEFTE2, IKJEFTE8, IKJEFTNS, IKJEFTAP are link-edited into load module IKJTABLS in SYS1.LPALIB, IKJTABLS out of SYS1.LPALIB is used unless IKJTSOxx is found in SYS1.PARMLIB (or its logical concatenation).
- If IKJEFTE2, IKJEFTE8, IKJEFTNS, IKJEFTAP, IKJTABLS are in an authorized STEPLIB, then for the general users, SYS1.PARMLIB is used. For the user with the STEPLIB however, the CSECTs are used. This is supported through the following display PARMLIB LIST.

See "Using SYS1.PARMLIB member IKJTSOxx" and "Using CSECTs IKJEFTE2, IKJEFTE8, IKJEFTAP, and IKJEFTNS" on page 154 for details on using either method.

Using SYS1.PARMLIB member IKJTSOxx

To use IKJTSOxx, do the following:

- If you have not already done so, copy sample member IKJTSO00 from SYS1.SAMPLIB to SYS1.PARMLIB. You might have already copied IKJTSO00 to define other installation defaults.
- You can create alternative members in SYS1.PARMLIB using the IKJTSOxx naming convention.
- Edit the member in SYS1.PARMLIB to contain the names of appropriate commands and programs for your installation.

Entries in IKJTSOxx must be command or program names up to eight characters in length, separated from other entries by one or more blanks, and enclosed in parentheses after one of the following parameters:

AUTHCMD NAMES
to specify authorized TSO/E commands.

AUTHPGM NAMES
to specify programs that are authorized when invoked via the CALL command.

AUTHTSF NAMES
to specify programs that are authorized when invoked through the TSO/E service facility and in most cases are not in AUTHPGM. These programs are primarily those which expect more complex parameter lists than that of the CALL command and use parameter 7 of the IKJEFTSR parmlist to supply parameters for the invoked program. As a general rule, programs in this list should not accept parameters that are pointers to code what is to be executed (such as exit routines) as this might introduce an integrity exposure.

Note: Do not place programs from any IBM products in this table unless specifically instructed to do so by the owning product documentation. For example, do not put IDCAMS in AUTHTSF.

NOTBKGND NAMES
to specify commands not supported in the background.

The following example shows required entries in the list of authorized commands distributed in IKJTSOxx. In this and other lists, you can add your own entries. The entries can contain comments and must use continuation symbols when the list continues on the following line.

```
AUTHCMD NAMES(              /* AUTHORIZED COMMANDS    */      +
   RECEIVE                  /*                        */      +
   TRANSMIT XMIT            /*                        */      +
   LISTB    LISTBC          /*                        */      +
   SE       SEND            /*                        */      +
   RACONVRT                 /*                        */      +
   CONSPROF                 /*                        */      +
   SYNC                     /*                        */      +
   TESTAUTH TESTA           /*                        */      +
   PARMLIB)                 /*                        */
```

The following example shows required entries in the list of programs to be authorized when called through the TSO/E service facility.

```
AUTHTSF NAMES(   /* PROGRAMS TO BE AUTHORIZED         */      +
                 /* WHEN CALLED THROUGH THE TSO       */      +
                 /* SERVICE FACILITY.                 */      +
   IKJEFF76)     /*                                   */
```

The following example shows required entries in the list of commands not supported in the background.

```
NOTBKGND NAMES(        /* COMMANDS WHICH MAY NOT BE    */      +
                       /* ISSUED IN THE BACKGROUND.    */      +
   OPER     OPERATOR   /*                              */      +
   TERM     TERMINAL)  /*                              */
```

Using CSECTs IKJEFTE2, IKJEFTE8, IKJEFTAP, and IKJEFTNS

If you choose not to use IKJTSOxx, you can use CSECTs IKJEFTE2, IKJEFTE8, and IKJEFTAP to specify authorized commands and programs, and CSECT IKJEFTNS to specify commands that are not supported in the background. If you update these CSECTs, you must assemble them, link- edit them into load module IKJTABLS in SYS1.LPALIB, and IPL the system before the updates take effect. Table 19 indicates the contents of the CSECTs.

Table 19. Contents of CSECTs IKJEFTE2, IKJEFTE8, IKJEFTAP, and IKJEFTNS

CSECT	Table	Contents
IKJEFTE2	APFCTABL	Names of authorized command processors that the TMP executes
IKJEFTE8	APFPTABL	Names of programs that are authorized when invoked via the CALL command
IKJEFTAP	APFTTABL	Names of programs that are authorized when invoked through the TSO/E service facility
IKJEFTNS	NSCPTABL	Names of commands not supported in the background.

Note: The tables of authorized commands and programs must start 16 bytes (X'10') into the appropriate CSECT on a doubleword boundary.

Specifying authorized commands and programs

The lists in APFCTABL, APFPTABL, and APFTTABL must contain the names of commands or programs, up to eight characters in length. Entries of less than eight characters must be left-justified and padded to the right with blanks. You can reserve extra space by adding blank strings at the end of the list. The system ignores entries following a blank string.

You can replace the IBM-supplied modules IKJEFTE2, IKJEFTE8, and IKJEFTAP by link-editing installation-supplied modules with these names into load module IKJTABLS in SYS1.LPALIB. Specify RMODE(24) when link-editing IKJTABLS. Consult the output from stage 1 for correct link-edit information. Any program that depends on a job step environment such as the TMP should not be placed in the lists.

- The following example shows how to include the list of authorized commands in IKJEFTE2.

```
         ENTRY APFCTABL
IKJEFTE2 CSECT
         DC    C'IKJEFTE2'     MODULE NAME
         DC    C'76.152  '     RELEASE LEVEL
APFCTABL DS    0D              ALIGNMENT
         DC    C'RECEIVE '
         DC    C'TRANSMIT'
         DC    C'XMIT    '     NOTICE IT IS PADDED WITH BLANKS
         DC    C'SYNC    '
         DC    C'RACONVRT'
         DC    C'LISTB   '
         DC    C'LISTBC  '
         DC    C'CONSPROF'
         DC    C'SEND    '
         DC    C'SE      '
         DC    C'TESTAUTH'
         DC    C'TESTA   '
         DC    C'PARMLIB '
         DC    C'        '     TERMINATOR
        END
```

- CSECT IKJEFTE8 of load module IKJTABLS contains a list of programs invoked as authorized programs through the CALL command processor.

  If you wanted to allow authorized access to PROGRAMX through CALL, then the list might look as follows.

```
           ENTRY     APFPTABL
IKJEFTE8   CSECT
           DC        CL8'IKJEFTE8'     MODULE NAME
           DC        CL8'85.092  '     RELEASE LEVEL
APFPTABL   DS        0D                ALIGNMENT
           DC        CL8'PROGRAMX'
           DC        CL8'        '
           END
```

- CSECT IKJEFTAP of load module IKJTABLS contains a list of program names that the TSO/E service facility attaches as authorized programs.

  If you wanted to allow access to PROGRAMX through the TSO/E service facility, then the list might look like the following.

```
           ENTRY     APFTTABL
IKJEFTAP   CSECT
           DC        CL8'IKJEFTAP'     MODULE NAME
           DC        CL8'86.120  '     RELEASE LEVEL
APFTTABL   DS        0D                ALIGNMENT
           DC        CL8'IKJEFF76'     REQUIRED FOR FIB COMMANDS
           DC        CL8'PROGRAMX'
           DC        CL8'        '
           END
```

Note: Table IKJEFTAP must include the module name IKJEFF76 for the SUBMIT, OUTPUT, STATUS, and CANCEL commands to work properly.

Specifying commands not supported in the background

CSECT IKJEFTNS lists the commands not supported in the background. The list as distributed is shown in the following example. You can add any commands that you wish to restrict from background jobs.

```
          ENTRY    NSCPTABL
IKJEFTNS  CSECT
          DC       C'IKJEFTNS'      MODULE NAME
          DC       C'76.033  '      RELEASE LEVEL
NSCPTABL  DS       0D               ALIGNMENT
          DC       AL2(8)           LENGTH OF COMMAND NAME
          DC       C'OPERATOR'      COMMAND NAME
          DC       AL2(4)           LENGTH OF COMMAND NAME
          DC       C'OPER    '      COMMAND NAME
          DC       AL2(8)           LENGTH OF COMMAND NAME
          DC       C'TERMINAL'      COMMAND NAME
          DC       AL2(4)           LENGTH OF COMMAND NAME
          DC       C'TERM    '      COMMAND NAME
          DC       AL2(0)           TEN MORE BLANK ENTRIES
          DC       C'        '      FOR INSTALLATION USE
          DC       X'FFFF'          TABLE TERMINATOR
          END
```

Command Name List Format: (See the preceding example.) Each entry in the list consists of a 2-byte length field (the length of the command name) and an 8-byte command name field. The name is left-justified and padded with blanks, as required. An unused entry consists of a 2-byte field containing zeroes and an 8-byte field containing blanks. The end of the list is indicated by a 2-byte field containing ones (X'FFFF').

Modifying the Command Name List: CSECT IKJEFTNS contains 10 entries available for installation use. You can add command names and aliases to the list. If 10 entries are insufficient for your use, you can increase the length of the list. Each time you modify the list (add or delete an entry, or change the length of the list), re- assemble CSECT IKJEFTNS and link-edit it into IKJTABLS.

When modifying the list, make sure that TERMINAL, TERM, OPERATOR, and OPER are not deleted.

Chapter 12. Specifying commands and programs for the command/program invocation platform

An application program can run commands and programs in a command/program invocation environment. This environment eliminates the need for MVS task initialization and termination each time you invoke a command or program. This support provides a potential performance benefit to users by helping to reduce system overhead associated with the initialization and termination of commands and programs.

You can invoke unauthorized commands and programs on a command/program invocation platform using the platform initialization and termination routines of the TSO/E service facility. The TSO/E service facility is an interface that allows a TSO/E user to invoke functions such as commands, programs, CLISTs, and REXX execs from an application program. For more information on the TSO/E service facility, see *z/OS TSO/E Programming Services*. You can invoke authorized commands and programs on a command/program invocation platform, even if you are not using the TSO/E service facility.

In order for commands and programs to run in the command/program invocation platform environment, your installation must first specify the eligible commands and programs on the appropriate platform support statement in the IKJTSOxx member of SYS1.PARMLIB. The PLATCMD statement identifies commands that are eligible to run on a command/program invocation platform. Similarly, the PLATPGM statement identifies programs that are eligible to run on a command/program invocation platform. TSO/E determines authorization for commands and programs from the list of authorized commands and programs that you maintain. For more information on command and program authorization, see Chapter 11, "Specifying authorized commands/programs, and commands not supported in the background," on page 151.

Considerations for specifying commands and programs

Only commands and programs that do not require the services of MVS task termination can run on the command/program invocation platform.

The PLATCMD statement in SYS1.SAMPLIB member IKJTSO00 contains a list of recommended commands that you can run on the command/program invocation platform. Unauthorized commands are:
- ALLOCATE
- ALTLIB
- ATTRIB
- CALL
- EXEC
- FREE
- PRINTDS
- PROFILE
- STATUS
- SUBMIT

Authorized commands are:
- CONSPROF
- LISTBC

- PARMLIB
- RACONVRT
- RECEIVE
- SEND
- SYNC
- TRANSMIT

The PLATPGM statement in SYS1.SAMPLIB member IKJTSO00 contains a list of recommended programs you can run on the command/program invocation platform.

IEFBR14 is an unauthorized program you can run.

IKJEFF76 is an authorized program you can run.

You can also add installation-written commands and programs to platform support statements in SYS1.PARMLIB, member IKJTSOxx. Before adding commands or programs to a platform support statement, the following must be considered:

- Any task-related resources that the command or program obtains must be explicitly released during normal processing (for example, subpool 0 storage or ENQ serialization).

  When a command or program that is running on the command/program invocation platform abnormally terminates or terminates due to an attention interruption, the system releases any task-related resources.
- If an installation-supplied command or program invokes dynamic allocation, it is the responsibility of the command or program to provide its own remove in-use processing, which is normally provided by the terminal monitor program (TMP).

  To properly invoke remove in-use processing, you must be aware of the level of the task that is issuing the dynamic allocation. If the allocation occurs at the primary task level, you must invoke remove in-use processing (SVC 99 with verb code 05) using the IEFZB4D2 mnemonic DRITCBAD and specifying the current TCB address. The current TCB address is in the PSATOLD field of the PSA. The mnemonic DRITCBAD specifies that the in-use attribute is to be used from all resources associated with the specified TCB address.

  When the allocation occurs below the primary task level you invoke, remove in-use processing and the IEFZB4D2 mnemonic DRICURNT. DRICURNT specifies that the in-use attribute is to be removed from all resources except those associated with the current task and those associated with higher-level tasks.

  For more information about dynamic allocation and remove in-use processing, see *z/OS MVS Programming: Assembler Services Guide*.

Using the PLATCMD and PLATPGM statements

To use the PLATCMD or PLATPGM statements in IKJTSOxx, do the following:

- If you have not already done so, copy sample member IKJTSO00 from SYS1.SAMPLIB to SYS1.PARMLIB. You may have already copied IKJTSO00 to define other installation defaults.
- You can create alternative members in SYS1.PARMLIB using the IKJTSOxx naming convention.
- Edit the member in SYS1.PARMLIB and locate the PLATCMD or PLATPGM statement.
- Specify the eligible commands or programs for your installation.

Entries in the PLATCMD and PLATPGM statements in IKJTSOxx must be enclosed in parentheses after the PLATCMD or PLATPGM NAMES statement. Each entry must be a command name for PLATCMD or a program name for PLATPGM. Command and program names must be 8 characters or less and separated from other entries by at least one space. PLATCMD NONE and PLATPGM NONE are valid statements that specify that no commands or programs are to run on the command/program invocation platform. If the PLATCMD statement is not contained in the current IKJTSOxx member, PLATCMD NONE is the default. Similarly, if the PLATPGM statement is not contained in the current IKJTSOxx member, PLATPGM NONE is the default.

Figure 19 shows the PLATCMD and PLATPGM statements and the eligible commands and program names.

```
PLATCMD NAMES(                    /*  COMMANDS TO BE INVOKED       */+
                                  /*  VIA THE TSO COMMAND/PROGRAM  */+
                                  /*  INVOCATION PLATFORM          */+
                                  /*  (COMMANDS IN THIS            */+
                                  /*  LIST MUST NOT REQUIRE        */+
                                  /*  TASK TERMINATION             */+
                                  /*  CLEANUP PROCESSING)          */+
ALLOCATE  ALLOC                   /*                               */+
ALTLIB              IKJADLIB      /*                               */+
ATTRIB    ATTR                    /*                               */+
CALL                IKJEFG00      /*                               */+
EXEC      EX        IKJEXC2       /*                               */+
FREE      UNALLOC                 /*                               */+
PRINTDS   PR        IKJEFY50      /*                               */+
PROFILE   PROF                    /*                               */+
SUBMIT    SUB                     /*                               */+
STATUS    ST        IKJEFFCA      /*                               */+
LISTBC    LISTB                   /*  AUTHORIZED COMMANDS          */+
PARMLIB   IKJPRMLB                /*  (THESE COMMANDS MUST         */+
RECEIVE                           /*  BE LISTED IN AUTHCMD         */+
SEND      SE                      /*  STATEMENT OR IKJEFTE2        */+
TRANSMIT  XMIT                    /*  TABLE)                       */+
SYNC                              /*                               */+
RACONVRT                          /*                               */+
CONSPROF)                         /*                               */

PLATPGM NAMES(                    /*  PROGRAMS TO BE INVOKED       */+
IEFBR14                           /*  ON THE TSO COMMAND/PROGRAM   */+
                                  /*  INVOCATION PLATFORM          */+
                                  /*  (PROGRAMS IN THIS            */+
                                  /*  LIST MUST NOT REQUIRE        */+
                                  /*  TASK TERMINATION             */+
                                  /*  CLEANUP PROCESSING)          */+
                                  /*                               */+
IKJEFF76)                         /*  AUTHORIZED PROGRAMS.         */+
                                  /*  THESE PROGRAMS MUST          */+
                                  /*  BE LISTED IN AUTHTSF         */+
                                  /*  (OR THE IKJEFTAP TABLE)      */+
                                  /*                               */
```

Figure 19. PLATCMD and PLATPGM statements

Chapter 13. Setting up the TRANSMIT and RECEIVE environment

Before people at your installation can use the TRANSMIT and RECEIVE commands, you must:

- Specify installation defaults that control TRANSMIT and RECEIVE processing, and system ID-node name associations, in one of the following:
  - SYS1.PARMLIB member IKJTSOxx
  - CSECT INMXPARM

  "Setting TRANSMIT and RECEIVE defaults in IKJTSOxx" on page 162 describes how to use IKJTSOxx, and "Writing and installing the INMXPARM CSECT" on page 162 describes how to replace the IBM-supplied dummy INMXPARM CSECT with your version. Unless you use IKJTSOxx or replace the dummy CSECT, TRANSMIT and RECEIVE do not work.
- Add TRANSMIT, its alias XMIT, and RECEIVE to the table of authorized commands. Chapter 11, "Specifying authorized commands/programs, and commands not supported in the background," on page 151 describes how to maintain that table either in the IKJTSOxx member of SYS1.PARMLIB, or in the IKJEFTE2 CSECT in load module IKJTABLS, which is in SYS1.LPALIB.
- Check and, if necessary, modify JES initialization statements. The JES initialization statements must specify that SYSOUT data is to be written, that the output class is punched, and that the punched card output limit is large enough to transmit data sets. "Modifying JES initialization statements" on page 162 lists the initialization statements you need to check.
- Unless you are using RACF, to enable users to issue RECEIVE in the background, ensure that their user IDs are placed in the ASXBUSER field of the ASXB control block. (RACF stores the user ID for you.) If the user ID is not stored in that field, RECEIVE does not work in the background. If you allow users to issue RECEIVE in the background, you might also need to write a RECEIVE pre-processing exit (INMRZ11R or INMRZ11) to respond to the prompts that RECEIVE issues to users.

After making TRANSMIT and RECEIVE available, you can use one or more TSO/E or JES exits to monitor or modify TRANSMIT or RECEIVE processing. Chapter 39, "Customizing TRANSMIT and RECEIVE," on page 395 describes the exits.

Specifying installation defaults for TRANSMIT and RECEIVE

Before people at your installation can use TRANSMIT and RECEIVE, you must specify installation defaults that control TRANSMIT and RECEIVE processing, and system ID-node name associations. To do so, you can use the INMXPARM CSECT or member IKJTSOxx of SYS1.PARMLIB. With IKJTSOxx, you avoid having to update, reassemble and link-edit the CSECT, and you can update or list the defaults dynamically using the PARMLIB command. For information about using the PARMLIB command, see *z/OS TSO/E System Programming Command Reference*.

If TSO/E finds both, the INMXPARM CSECT and the TRANSREC statement in member IKJTSOxx of SYS1.PARMLIB, it uses either of it as shown:

Table 20. Usage of INMXPARM CSECT versus TRANSREC statement

	INMXPARM is IBM-supplied	**INMXPARM is customized**
No TRANSREC statement present	Error message INMR152I or INMX152I is issued	INMXPARM is used
TRANSREC statement present with operands	Defaults from IKJTSOxx are used	Defaults from IKJTSOxx are used

Note: If the TRANSREC statement is found without operands in IKJTSOxx it is treated as if it is not present.

Setting TRANSMIT and RECEIVE defaults in IKJTSOxx

To specify TRANSMIT and RECEIVE defaults in IKJTSOxx, do the following:

- If you have not already done so, copy sample member IKJTSO00 from SYS1.SAMPLIB to SYS1.PARMLIB. You may have already copied IKJTSO00 to define other installation defaults.
- You can create alternative members in SYS1.PARMLIB using the IKJTSOxx naming convention.
- Edit the member in SYS1.PARMLIB to contain the appropriate TSO/E options for your installation.

In IKJTSOxx, code the TRANSMIT and RECEIVE defaults on the TRANSREC statement. They include all the defaults you can specify in CSECT INMXPARM, and some more in addition.

For information about the TRANSREC syntax in IKJTSOxx, see *z/OS MVS Initialization and Tuning Reference*.

Writing and installing the INMXPARM CSECT

IBM supplies a dummy INMXPARM CSECT that you can replace by editing and executing the job stream in the INMINOPT member of SYS1.SAMPLIB. The job stream creates an SMP/E user-supplied system modification (a USERMOD) that replaces the IBM-supplied version of INMXPARM with your version. INMINOPT includes instructions for updating the job stream to create the required USERMOD.

The job stream includes the following INMXP, INMNODE, and INMEND macro instructions for specifying INMXPARM values. You must modify them:

```
INMXP
INMNODE NODENAME,SMFID
INMEND
```

Use INMXP to specify installation controls for TRANSMIT and RECEIVE processing, and defaults for parameters on the TRANSMIT and RECEIVE commands. Use INMNODE to associate a system ID with a network node name. Use INMEND to mark the end of the list of INMNODE statements. Chapter 51, “Macro syntax,” on page 707 gives the syntax and rules for specifying those macros.

Modifying JES initialization statements

If your installation has JES2 installed:

- Specify OUTPUT=YES on the TSUCLASS and STCCLASS JES2 initialization statements to specify that SYSOUT data is to be written for jobs executed in time sharing classes. JES2 treats a file submitted by TRANSMIT processing as an output data set. Unless the SYSOUT data set can be written to a spool device, TRANSMIT does not work.

  ```
  TSUCLASS OUTPUT=YES, ...
  STCCLASS OUTPUT=YES, ...
  ```
- Include a JES2 OUTCLASS initialization statement to specify the output class for punched output:

  ```
  OUTCLASS(v) OUTPUT=PUNCH, ...
  ```

  The value you specify for OUTCLASS(v) should be identical to the value specified on either:
  - The SPOOLCL parameter on the TRANSREC statement in PARMLIB member IKJTSOxx
  - The INMXPARM CSECT

  Refer to *z/OS JES2 Initialization and Tuning Guide* and *z/OS JES2 Initialization and Tuning Reference* for additional JES2 initialization statements and *z/OS JES2 Installation Exits* for installation exits that can affect punch card output limits.

If your installation has JES3 installed:
- Include a JES3 SYSOUT initialization statement to specify that the output class is to be punched:

  ```
  SYSOUT, CLASS=outclass,TYPE=PUNCH, ...
  ```

  The value you specify for the SYSOUT class should be identical to the value specified on either:
  - The SPOOLCL parameter on the TRANSREC statement in PARMLIB member IKJTSOxx
  - The INMXPARM CSECT

  Refer to *z/OS JES3 Initialization and Tuning Guide* and *z/OS JES3 Initialization and Tuning Reference* for additional JES3 initialization statements and *z/OS JES3 Customization* for installation exits than can affect punched card output limits.

Chapter 14. Customizing the HELP data set

The SYS1.HELP data set contains a member for each TSO/E command. When a user issues the HELP command, information from the HELP data set is displayed. Control characters within each member determine the information that is displayed.

This chapter explains how you can customize the HELP data set by adding new members or adding control characters to existing members.

You can also customize the HELP command to allocate different help data sets for particular languages. See Chapter 20, "Customizing TSO/E for different languages," on page 185 for information about providing help information in different languages.

Updating the HELP data set

Using the prompt mode function

Whenever a user issues a command and does not enter a positional operand or specifies the operand incorrectly, the parse service routine prompts the user. The user can then enter question marks to receive second-level messages that provide information about the operand. If second-level messages are unavailable or they have all been displayed, parse determines whether it can use the *prompt mode HELP function*. This function lets parse generate a HELP command and retrieve information about the operand from the HELP data set.

The prompt mode HELP function is available for the following TSO/E commands: ALTLIB, ATTRIB, CALL, CANCEL, EDIT, EXEC, HELP, OUTPUT, RUN, SEND, RACONVRT, SYNC, and TRANSMIT.

You can provide the prompt mode HELP function for other TSO/E commands and subcommands that have positional operands. The exceptions are the TEST and TESTAUTH commands, which do not support this function. To provide the prompt mode HELP function for these commands and subcommands, update the specific HELP data set members for the command in SYS1.HELP. The member name is the same name as the command. In the member, enter the positional control character `)P` on the first line of each positional operand description for the command and its subcommands. For more information about the prompt mode HELP function and the)P positional control character, see *z/OS TSO/E Programming Guide*.

Note: If you update a member for this support, you must ensure that you insert a)P for all of the positional operands for a command and its subcommands. If you do not do this and a positional operand does not have an)P, unpredictable results may occur when the parse service routine uses the prompt mode HELP function.

Using the include control character

In addition to displaying information from a single member of the HELP data set, you can include help information contained in other members by using the include control character followed by the member name. Using the include control character is useful when you plan to add information that is not yet available, and also when you must repeat information. For more information about the include control character and other control characters for the HELP data set, see *z/OS TSO/E Programming Guide*.

Chapter 15. Making host services available to PC users

The TSO/E command processor MVSSERV allows IBM host computers and properly-configured IBM personal computers to communicate. This communication enables personal computer (PC) programs to access a host computer's services, using IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities (ECF).

The host services are made available to the PC *requester* programs by corresponding *server* programs on MVS. IBM provides several server and requester programs, and you can write servers and requesters of your own.

Initializing MVSSERV for the TSO/E Enhanced Connectivity Facility

The TSO/E Enhanced Connectivity Facility (ECF) allows you to use IBM-supplied servers or write your own servers for use on TSO/E. The tasks you can perform using IBM-supplied server/requesters are:

- Use host disk space as if it were a personal computer fixed disk
- Print personal computer output on certain host printers as if they were personal computer printers
- Copy files from the personal computer to the host
- Execute TSO/E commands, CLISTs, and REXX execs

For more information about the servers and requesters IBM provides, and PC configurations supported, see *Enhanced Connectivity Facilities Introduction*.

You can write your own servers and requesters to provide additional host services to PC users. You can provide any service that is available to a problem program on MVS. Your servers are restricted only by MVS and PC resources, and by conventions used in the requester program. For information about writing and installing your own servers on TSO/E, see *z/OS TSO/E Guide to SRPI*. That document describes how to:

- Supply and initialize the input parameter data set for MVSSERV
- Supply diagnostic data sets for MVSSERV
- Write and package the servers and their initialization/termination programs
- Install the servers, initialization/termination programs, and access method drivers

With ECF, you can customize the way in which servers and requesters communicate. MVSSERV includes programs called access method drivers, which manage communications with PCs attached to the host through an IBM 3174 or 3274 control unit in Distributed Function Terminal (DFT) or Control Unit Terminal (CUT) mode. MVSSERV also provides an interface that lets you write and install other access method drivers to support other modes of host-to-PC attachment. For example, you could write an access method driver that supports a phone connection between the host and a PC, allowing a PC user to access host services by phone.

For information about writing and installing access method drivers, see *z/OS TSO/E Guide to SRPI*.

Chapter 16. Monitoring TSO/E resources

You can monitor certain TSO/E resources, such as TSO/E users and commands. This section will help you decide whether to monitor the:
- TSO/E commands users issue
- Performance of TSO/E transactions.

Monitoring TSO/E commands

You can monitor the TSO/E commands users issue and record the number of times a user issues a specific command or subcommand. System Management Facilities (SMF) records information about TSO/E commands in a type 32 SMF record. You can write an application to process and report on the information that SMF records. For example, you can:
- Keep track of and compare how frequently certain commands at your installation are used. You may want to provide better performance for the more commonly used commands by placing them in LPALIB.
- Keep track of the number of times users issue TSO/E commands so you can bill users for their computer use.
- Audit the commands users issue to ensure they do not violate security practices at your installation.

By default, SMF records contain certain statistics about most TSO/E commands. You can record information about additional TSO/E commands, such as MVSSERV, or keep information about certain TSO/E commands from being recorded. For information about how to add or delete commands, see *z/OS MVS System Management Facilities (SMF)*.

You may also want to record additional information about each TSO/E command, such as the number of TPUTs and TGETs that are issued for each command. Monitoring the number of TPUTs and TGETs allows you to keep track of terminal activity. To record additional information about each command, use the SYS parameter in SYS1.PARMLIB member SMFPRMxx. For more information about using SMFPRMxx, see *z/OS MVS System Management Facilities (SMF)*.

Note that you cannot use SMF to restrict users from using commands. For information about limiting commands, see Chapter 18, “Protecting the resources TSO/E users can access,” on page 175.

Monitoring the performance of TSO/E users

You can use SMF, System Resource Manager (SRM), and RMF to monitor the performance of TSO/E users. You can monitor the performance of TSO/E users on-line or in batch.

Monitoring the performance of TSO/E users on-line

You can use RMF Monitor III to monitor the performance of TSO/E users on-line, identify performance problems as they occur, and identify the TSO/E users who are delayed. You can customize RMF reports to include information about the relative speed of domains, performance groups, or individual TSO/E users. You can also report exceptional conditions, and request that RMF notify you when the performance of a TSO/E user or group of users falls below a certain level.

For information about using RMF Monitor III to monitor the performance of TSO/E users, see *z/OS RMF Report Analysis*.

Collecting statistics about transactions in batch

You can use RMF Monitor I to collect statistics about users and commands in batch. For each reporting performance group in your system, you can gather information about the:

- Average service rate
- Number of completed transactions
- Average response time of each transaction

RMF Monitor I helps you find out about system performance over time, so you can do capacity planning. RMF allows you to monitor the load a set of users places on the system, or find out whether a particular TSO/E command is consuming too many resources. For more information about using RMF Monitor I, see *z/OS RMF Report Analysis*.

Chapter 17. Defining performance objectives for TSO/E

You should review, and if necessary, adjust the factors that affect the performance of TSO/E. To develop objectives for TSO/E, you must:

- Decide what kind of on-line response time you want for TSO/E users
- Balance TSO/E against other work at your installation, such as batch, and decide the proportion of resources to be given TSO/E
- Decide whether to provide better performance for certain TSO/E users or commands
- Decide whether to provide more consistent response time for TSO/E commands

To put the objectives into effect, change System Resource Management (SRM) parameters in SYS1.PARMLIB. Before you change SRM parameters, review the information about SRM in *z/OS MVS Initialization and Tuning Reference*.

As you tune your system to provide better response time for TSO/E users, you may adversely affect the performance of other work at your installation, such as batch. There is usually a trade-off between the performance of your batch system and on-line response time.

Deciding what kind of response time TSO/E users will have

Before you can adjust TSO/E, you must determine what good performance is. The basic criterion of TSO/E performance is response time. When deciding what kind of response time you want, consider:

- The resources available at your installation, such as processor capacity
- What priority TSO/E users must have in relationship to other work at your installation, such as batch. After you have decided what priority TSO/E users have, you can decide the proportion of the resources to which TSO/E users have access.
- The expectations of the users and what they use the system for

Because TSO/E users work interactively with TSO/E, it is important that overall response time be fast enough so that users can work effectively. The optimum response time varies depending on how you use the system. In systems where users enter data rapidly and continuously, fast response time is more critical than in a system that carries out complex calculations.

Because users expect good response time from TSO/E commands, you should try to tune your system so that most TSO/E commands complete within one second. Categorize TSO/E commands according to the resources they require and provide better response time to commands that require fewer resources. To define your objectives, use SYS1.PARMLIB member IEAIPSxx. For more information about SYS1.PARMLIB member IEAIPSxx, see *z/OS MVS Initialization and Tuning Reference*.

Deciding about better performance for certain users and commands

You can allow individual users or groups of users to have different levels of performance than they receive by default. For example, you can give users who must accomplish high-priority work more resources and better response time. To associate TSO/E users with different levels of performance, use SYS1.PARMLIB member IEAICSxx. For more information about SYS1.PARMLIB member IEAICSxx, see *z/OS MVS Initialization and Tuning Reference*.

Although you cannot, through SYS1.PARMLIB member IEAICSxx, associate TSO/E commands with different levels of performance, you can provide different levels of performance for both commands and user-written exit routines, depending on where you place them. By default, most TSO/E commands reside in SYS1.CMDLIB, and exits IBM provides reside in the same place as the load modules with which they are associated. You can install commands and exit routines in PLPA (pageable link pack area), in FLPA (fixed link pack area), in the LNKLST concatenation, or in a private STEPLIB data set.

Using PLPA or FLPA provides the best performance. The commands reside in virtual storage, so the system does not have to do the I/O required for retrieving commands from SYS1.CMDLIB. However, placing commands in PLPA or FLPA:
- Uses more common storage.
- Makes it more difficult to make changes. To update commands or exits in PLPA or FLPA, you must re-IPL your system.
- Makes the process of installing TSO/E less straight-forward. When you install a new release of TSO/E, you must copy the new versions of the commands to PLPA.

You should move all reentrant TSO/E modules that can reside either above or below 16 MB in virtual storage to PLPA. If your system is storage-constrained, you may want to move just the most commonly-used commands and exit routines to PLPA. To identify the most commonly-used commands, use System Management Facilities (SMF). For more information about using SMF, see Chapter 16, “Monitoring TSO/E resources,” on page 169.

You may want to move commands that are infrequently used, but must provide good response time, to FLPA. For example, if users log on infrequently, you may want to page-fix the modules associated with the LOGON command. However, note that some of the modules associated with the LOGON command reside below 16 MB in virtual storage. Therefore, if your system is storage-constrained, you may not want to page-fix these modules. To page-fix modules associated with logon processing, update SYS1.PARMLIB member IEAFIXxx. For more information about IEAFIXxx, see *z/OS MVS Initialization and Tuning Reference*.

Although the performance of commands and exit routines in the LNKLST concatenation is generally not as good as PLPA, modules are easier to update. To update modules, you must build a new copy of the LNKLST directory. For information about how to build a new copy of the LNKLST directory, see *z/OS MVS System Commands*.

Note: If you move the following subcommands of the EDIT command from PLPA, do not delete them from the LNKLST concatenation: ALLOCATE, ATTRIB, EXEC, FREE, HELP, PROFILE, RUN, SEND, and SUBMIT.

You should avoid placing commands and programs in STEPLIB data sets, except when you are testing them. Using STEPLIB data sets could adversely affect the performance of your system because each time you execute a command or load a program, the system searches the STEPLIB data set before searching any others. If you must use STEPLIB data sets, you should not concatenate them. Response time is usually better if you use one large data set instead of concatenating several smaller data sets.

Making TSO/E response time more consistent

To make TSO/E response time more consistent, use the RTO parameter in SYS1.PARMLIB member IEAIPSxx. You may want to keep TSO/E response time from fluctuating if your system is used heavily during one part of the day, and used little during others. If you are installing TSO/E for the first time, you may want to reserve resources for the future to keep response time from declining noticeably as you add users. Note that you cannot use SYS1.PARMLIB member IEAIPSxx to improve response time in a heavily-loaded system, only to make response time more consistent. One possible drawback to using the RTO parameter is that it defines the minimum possible response time. The response time for TSO/E will never be faster than the value you specify. For more information about SYS1.PARMLIB member IEAIPSxx, see *z/OS MVS Initialization and Tuning Reference*.

Chapter 18. Protecting the resources TSO/E users can access

You can use several facilities to limit the use of TSO/E commands and protect TSO/E data sets. This chapter provides an overview of how to perform these tasks.

Limiting the use of TSO/E commands

You may want to limit the use of TSO/E commands at your installation. For example, you may want to limit certain commands to conserve resources, such as spool space. You can restrict the TSO/E commands users can use from:
- READY mode TSO/E
- The TSO/E Session Manager
- Background mode
- ISPF/PDF panels

Table 21 lists the commands you can limit from each environment and the functions you can use to limit the commands.

Table 21. Commands you can limit

Environment	Commands you can limit	Method
READY mode TSO/E	ACCOUNT, OPERATOR, RACONVRT, and SYNC	ACCOUNT and RACF commands, and the logon exit IKJEFLD or IKJEFLD1
	SUBMIT, OUTPUT, STATUS, and CANCEL	ACCOUNT and RACF commands, and exit routines
	CONSOLE and CONSPROF	RACF commands, logon exit IKJEFLD or IKJEFLD1, and the CONSOLE and CONSPROF exit routines
	PARMLIB and TESTAUTH	RACF commands and exit routines
	TRANSMIT, RECEIVE, SEND, LISTBC, LOGON, LOGOFF, ALLOCATE, ALTLIB, TEST, FREE, OUTDES, PRINTDS, EXEC, and OPERATOR SEND	Exit routines
TSO/E Session Manager	Any TSO/E command	Session Manager exit routines
Background mode	Any TSO/E command	SYS1.PARMLIB member IKJTSOxx or CSECT IKJEFTNS
ISPF/PDF panels	Any TSO/E command	ISPMTCM macro statement

Limiting the TSO/E commands users can use from READY mode

The following topics describe how you can limit the use of commands from READY mode. Limiting the use of these commands from READY mode also limits their use from ISPF/PDF and the TSO/E Session Manager, but not from the background.

Limiting the use of the ACCOUNT, OPERATOR, RACONVRT, and SYNC commands

By default, users cannot use the ACCOUNT, OPERATOR, RACONVRT, or SYNC commands. You should grant only a few users at your installation the authority to use these commands. You should grant ACCOUNT command authority only to users who must add and maintain user profiles. Granting users authority to use the ACCOUNT command automatically allows them to use the SYNC command. Because limiting the use of the broadcast data set may improve the performance of the LOGON command, you may want to restrict the use of the OPERATOR command to users who must send notices and cancel other users' sessions. You should restrict the use of the RACONVRT command to users responsible for converting from SYS1.UADS to the RACF data base.

You grant users access to the ACCOUNT, OPERATOR, SYNC, and RACONVRT commands when you add them to TSO/E. To specify whether a user can use these commands, use either the ACCOUNT command or the RACF RDEFINE and PERMIT commands depending on whether the user is defined in the UADS or the RACF data base. You can also authorize users to the ACCOUNT, SYNC, and OPERATOR commands using the logon pre-prompt exit IKJEFLD or IKJEFLD1.

For more information about using the ACCOUNT command, see *z/OS TSO/E System Programming Command Reference*. For more information about using RACF commands, see *z/OS Security Server RACF Command Language Reference*.

Using RACF commands, the ACCOUNT command, or the logon exits to limit the use of the ACCOUNT command limits its use in the foreground, but not in the background. Users who have not been granted authority to use the ACCOUNT command can still use it in background mode. To restrict the use of the ACCOUNT command in the background:

- Restrict access to the user attributes data set (UADS). For example, you can use RACF to limit users who have access to the data set.
- Use SYS1.PARMLIB member IKJTSOxx or CSECT IKJEFTNS to specify that the ACCOUNT command cannot be used in the background.

It is probably better to restrict access to the UADS, rather than restricting the use of the ACCOUNT command in the background. Restricting the use of the ACCOUNT command in the background restricts all users from using it, and you may want administrators at your installation to be able to run the command in the background.

Limiting the use of the SUBMIT, OUTPUT, STATUS, and CANCEL commands

You can allow users to submit and process batch jobs with the TSO/E SUBMIT, OUTPUT, STATUS, and CANCEL commands. By default, users cannot use these commands. Because TSO/E jobs are submitted randomly, they may interfere with other batch jobs. For this reason, you may want to grant only the users who must submit and process batch jobs access to the SUBMIT, OUTPUT, STATUS, and CANCEL commands. For example, you may want to keep users whose primary job involves on-line data entry from submitting batch jobs and processing the output.

You grant users access to the SUBMIT, OUTPUT, STATUS, and CANCEL commands when you add them to TSO/E by using either the ACCOUNT command or the RACF RDEFINE and PERMIT commands. You can only allow a user to use all, or none, of these commands.

For more information about using the ACCOUNT command to restrict the use of the SUBMIT, OUTPUT, STATUS, and CANCEL commands, see *z/OS TSO/E Administration*. For more information about using RACF commands, see *z/OS Security Server RACF Command Language Reference*.

To further control the use of these commands, use TSO/E, JES2, and JES3 installation exits as described in Chapter 34, "Customizing the SUBMIT command and job output processing," on page 293. For example, you can use the TSO/E OUTPUT, STATUS, and CANCEL command exit to restrict the jobs for which a user can obtain status.

Limiting the use of the CONSOLE and CONSPROF commands

The CONSOLE command allows users to issue MVS system and subsystem commands and obtain responses to those commands. By default, users cannot use the CONSOLE command.

You can grant users access to the CONSOLE command using one of the following methods:

- RACF RDEFINE and PERMIT commands.

  Use the RACF RDEFINE command to define CONSOLE as a RACF resource belonging to the TSOAUTH RACF class. Then grant selected users access to the CONSOLE resource using the RACF PERMIT command.
- Logon pre-prompt exit IKJEFLD or IKJEFLD1.
- CONSOLE exit IKJCNXAC and CONSPROF exit IKJCNXCI. IKJCNXAC grants users CONSOLE command authority for the duration of the console session. IKJCNXCI grants users CONSOLE command authority for the duration of the CONSPROF command.

  Note: Because each of these exits can give users only temporary CONSOLE command authority, it is suggested that you use both exits to authorize users.

The CONSPROF command allows users to set up a console profile to tailor the processing of the CONSOLE command. Users require CONSOLE command authority to use the CONSPROF command. Granting users CONSOLE command authority using RACF commands or the LOGON pre-prompt exit automatically allows them to use the CONSPROF command.

For information about writing CONSOLE and CONSPROF exit routines, see Chapter 29, "Customizing the CONSOLE and CONSPROF commands," on page 233.

Limiting the use of the PARMLIB command

You can list and update TSO/E specifications that are in effect on the system using the PARMLIB command. By default, users cannot use this command.

You grant users access to the PARMLIB command using the RACF RDEFINE and PERMIT commands, or by writing an exit routine for PARMLIB.

You can use the RACF RDEFINE command to define PARMLIB as a RACF resource belonging to the TSOAUTH RACF class. Then grant selected users access to the PARMLIB resource using the RACF PERMIT command.

For information about writing an exit routine to control access to PARMLIB, see Chapter 33, "Customizing the PARMLIB command," on page 287.

Limiting the use of the TESTAUTH command

You can issue the TESTAUTH command to test authorized assembler programs. By default, users cannot use this command. The users of the TESTAUTH command should be limited to system programmers who need to debug authorized exits, commands, or programs for your installation.

You grant users access to the TESTAUTH command using the RACF RDEFINE and PERMIT commands, or by writing an exit routine for TESTAUTH.

You can use the RACF RDEFINE command to define TESTAUTH as a RACF resource belonging to the TSOAUTH RACF class. Then give selected users access to the TESTAUTH resource using the RACF PERMIT command.

For information about writing an exit routine to control access to TESTAUTH, see Chapter 38, "Customizing the TESTAUTH command," on page 383.

Limiting the use of other commands using exits

By default, users can issue the TRANSMIT, RECEIVE, SEND, LISTBC, LOGON, LOGOFF, ALLOCATE, ALTLIB, TEST, FREE, OUTDES, PRINTDS, and EXEC commands from READY mode. To limit these commands to certain users, you can use the exit routines provided for these commands.

Users who are authorized to use the OPERATOR command can use the SEND subcommand. You can restrict authorized users of the OPERATOR command from using the SEND subcommand by writing an exit routine. For more information, see the individual descriptions of the exit routines in this document.

Limiting the commands users can use from Session Manager

By default, users can use all TSO/E commands from the TSO/E Session Manager. You can use Session Manager exits to restrict all users, or specific users, from using any TSO/E command.

If users at your installation primarily use the TSO/E Session Manager to interact with TSO/E, you may want to restrict the commands they can use under Session Manager. For example, you could restrict the use of TRANSMIT and RECEIVE if spool space at your installation is limited.

One problem with using Session Manager to restrict the use of commands is that the commands are restricted only when users use Session Manager. Because users can issue TSO/E commands from other environments, such as ISPF/PDF and TSO/E READY mode, using Session Manager to restrict the commands users can issue is effective only if users do all of their work under Session Manager.

For more information, see "Writing Session Manager exits" on page 571.

Limiting the commands users can use in the background

By default, users can issue all TSO/E commands in background mode, except for the OPERATOR and TERMINAL commands and their aliases OPER and TERM. You may want to restrict the use of certain TSO/E commands to, for example, keep users from:
- Using user-written commands that do not work in the background
- Using commands whose use you plan on restricting in foreground mode.

To limit the use of commands in the background, you can either use SYS1.PARMLIB member IKJTSOxx, or update the CSECT IKJEFTNS. One advantage of using SYS1.PARMLIB member IKJTSOxx is that it is easier to update. You must reassemble and link-edit CSECT IKJEFTNS each time you update it.

Limiting the commands users can issue from ISPF/PDF panels

By default, users can issue most TSO/E commands from ISPF/PDF panels. For a list of the commands users can issue by default, see *z/OS ISPF Planning and Customizing*. To restrict users from using TSO/E commands from ISPF/PDF panels, modify the ISPTCM module. For example, if you wanted to keep users from using any TSO/E commands, and restrict them to just ISPF/PDF panels, you would have to remove the TSO/E commands from the module. To update the ISPTCM module, use the ISPF ISPMTCM macro. For more information about the ISPMTCM macro, see *z/OS ISPF Planning and Customizing*.

Limiting access to data sets

You may want to control the access TSO/E users have to data sets. For example, you may want to allow users to allocate only data sets having a high-level qualifier of the user's user ID, giving you more control over the use of DASD.

You can use several functions to control access to data sets and DASD, including:

- The MVS allocation input validation routine (IEFDB401). For example, you can restrict the data sets a user can allocate. For more information, see *z/OS MVS Programming: Authorized Assembler Services Guide*.
- RACF. You can use RACF to specify which users can access:
  - Non-VSAM and VSAM data sets
  - Generation data groups.

  You can RACF-protect one data set at a time, or many. For example, you can specify that all of the data sets beginning with the high-level qualifier ‘SYSTEM’ have the same protection.

  To RACF-protect a data set, use either RACF commands or ISPF RACF panels to build a profile for the data set. The profile contains information about the users who can access the data set.

  With RACF installed, security label checking can be activated. In this case, each data set and each user have security labels associated with them. The security label of the data set and the security label of the user's current session are checked. Access to the data set is determined by the result of that check.

For more information about using RACF to protect data sets, see *z/OS Security Server RACF Security Administrator’s Guide*.

Users can password-protect their data sets by using the TSO/E PROTECT command. You may want to limit the use of the PROTECT command at your installation because it may be difficult to centrally manage data sets that are individually password-protected. For more information about using the PROTECT command, see *z/OS TSO/E Command Reference*.

With RACF installed, messages contained in individual user log data sets can be protected from the user's view while the user is logged on at an insufficient security label. For more information, see Chapter 36, “Customizing how users send and retrieve messages,” on page 333.

Summary of resources protected using RACF

Table 22 summarizes the RACF profiles needed to protect various TSO/E resources. Descriptions of these TSO/E resources and more details on protecting them can be found in the previous sections of this chapter.

Table 22. Summary of resources protected using RACF

RACF class name	Profile name	Resource protected
TSOPROC	procedure name	Logon procedure
ACCTNUM	account number	Account number for TSO/E session
PERFGRP	performance group	Performance group for TSO/E session
TSOAUTH	ACCT	ACCOUNT, SYNC, & RACONVRT commands
	JCL	SUBMIT, CANCEL, OUTPUT, & STATUS commands
	MOUNT	Allow this user to issue dynamic allocation requests that result in the need for volume mounting
	OPER	OPERATOR command
	RECOVER	EDIT command recovery facility
	PARMLIB	PARMLIB command (read access for LIST, UPDATE access for UPDATE
	TESTAUTH	TESTAUTH command
	CONSOLE	CONSOLE & CONSPROF commands

Chapter 19. Customizing PUTGET and GETLINE processing

Commands and programs can invoke the PUTGET and GETLINE service routines to obtain lines of input. You can use the pre-PUTGET and pre-GETLINE exit (IKJEFXG1) to customize PUTGET and GETLINE processing.

Customizing PUTGET and GETLINE

Functional description

The PUTGET macro instruction puts messages out to the terminal and obtains a response to those messages. For example, the responses can be commands, subcommands, and prompt messages. The GETLINE macro instruction obtains lines of input. Both PUTGET and GETLINE obtain a line of input from the input stack, the REXX data stack, or the terminal. For more information about PUTGET and GETLINE, see *z/OS TSO/E Programming Services*.

To customize PUTGET and GETLINE processing, use the IKJEFXG1 exit. TSO/E gives control to this exit when either PUTGET or GETLINE obtains a line of input.

TSO/E-supplied exits

TSO/E does not provide a default exit routine for IKJEFXG1.

Entry specifications

For IKJEFXG1, the contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Entry point address

Parameter descriptions for IKJEFXG1

IKJEFXG1 receives the standard exit parameter list with the following exceptions:
- the command buffer field is not used
- the new command buffer is not used
- the exit reason code is not used

For a description of the standard exit parameter list, see “TSO/E standard exit parameter list” on page 32.

Figure 20 shows the exit-dependent data that IKJEFXG1 receives beginning at offset +36 (decimal) in the parameter list.

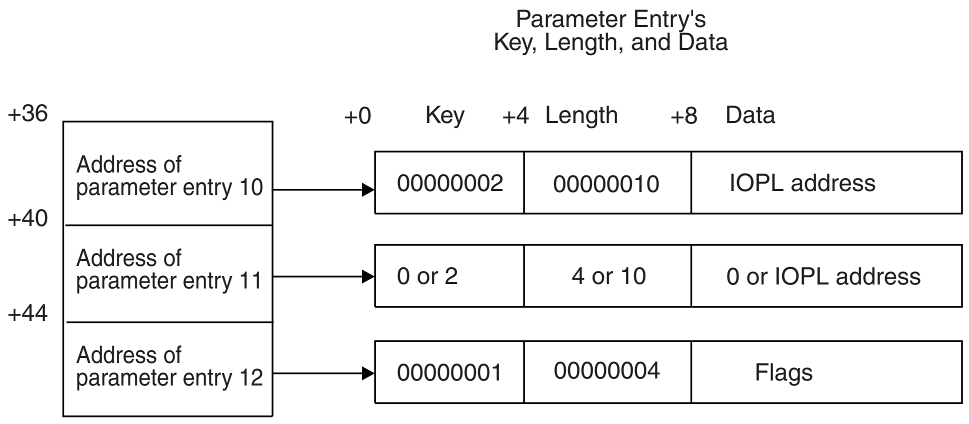


Figure 20. Exit-dependent data for exit IKJEFXG1

IOPL Address to GETLINE (Parameter Entry 10)
This parameter is an address of a pointer to the input/output parameter list (IOPL) received on invocation of the GETLINE service routine. The IOPL is passed to the GETLINE service routine. Any modifications made to the IOPL address are ignored by the GETLINE service routine. However, modifications can be made to the IOPL itself. Parameter entry 10 is always passed to IKJEFXG1 by the PUTGET and GETLINE service routines. For more information about the IOPL, see *z/OS TSO/E Programming Services*.

IOPL Address to PUTGET (Parameter Entry 11)
This parameter is an address of a pointer to the input/output parameter list (IOPL) received on invocation of the PUTGET service routine. The IOPL is passed to the PUTGET service routine. Any modifications made to the IOPL address are ignored by the PUTGET service routine. However, modifications can be made to the IOPL itself. Parameter entry 11 is always passed to IKJEFXG1 by the PUTGET service routine. For more information about the IOPL, see *z/OS TSO/E Programming Services*.

Flags (Parameter Entry 12)
This parameter contains a word consisting of flags. The flags for this exit's processing are described in Table 23.

Table 23. Flags for IKJEFXG1

Number of bytes	Field name	Contents or meaning
1		**10.. ....** The application program invoked the GETLINE service routine. Parameter entry 10 is always passed to IKJEFXG1.
		01.. .... The routine is invoked for PUTGET processing. Parameter entry 11 contains a pointer to the PUTGET's IOPL.
		..xx xxxx Reserved.
3		Reserved.

Return specifications

On return from IKJEFXG1, the contents of the registers must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for IKJEFXG1

IKJEFXG1 always sets a return code of zero.

Programming considerations

IKJEFXG1 receives control whenever TSO/E is obtaining a line of input from the input stack, the REXX data stack, or the terminal through the PUTGET or GETLINE service routine. Recursive calls are possible if the exit attempts to invoke the PUTGET or GETLINE service routine or any program (for example, the parse service routine) that invokes PUTGET or GETLINE.

The exit must be reentrant, refreshable, reusable, and reside in an APF-authorized library. It is inadvisable for the exit to have APF authorization unless it is designed also to be called as the entry point for a job step program.

Environment

IKJEFXG1 requires the following environment:
- State: Problem Program
- Key: 8
- AMODE(31), RMODE(ANY)
- Reside in an APF-authorized library

Installing the exit

You must name the exit IKJEFXG1. Link-edit the exit as a separate load module. The exit must reside in an authorized library, preferably LPA.

Chapter 20. Customizing TSO/E for different languages

TSO/E takes advantage of the MVS message service to let you provide TSO/E information to users in their national language. TSO/E information includes:

- TSO/E messages
- Help text for TSO/E commands
- The TRANSMIT command full-screen panel header information and PF key definitions

In addition, the TSO/E CONSOLE command supports the display of translated system messages issued during an extended MCS console session.

Note: The Session Manager does not support the display of translated information.

To provide translated messages and information (HELP and TRANSMIT), you need to initialize and activate the MVS message service and set up languages for users. For help information, you also need to specify the help data sets to be used for each language in the IKJTSOxx member of SYS1.PARMLIB.

This section describes how you can:

- Initialize and activate the MVS message service
- Specify help data sets for different languages
- Set up languages for users

Providing translated messages

The MVS message service (MMS) enables your installation to use message files to display translated messages. MMS substitutes a message translated into another language for the equivalent U.S. English message.

For TSO/E to display translated messages, your installation needs to format install message files that contain U.S. English message skeletons and the translated language message skeletons using the MVS message compiler.

For MVS messages, IBM provides the install message file for U.S. English messages and the install message file for Japanese translations. IBM also provides TSO/E install message files in upper and mixed case U.S. English and Japanese. For language translations other than Japanese, your installation must supply its own version of the install message file with the appropriate translated message skeletons. For more information, see *z/OS MVS Planning: Operations*.

Initializing and activating the MVS message service

The following steps describe what your installation must do so TSO/E can display translated messages. For complete information about these tasks, see *z/OS MVS Planning: Operations*.

1. Ensure that the appropriate install message files have been installed on your MVS/ESA System Product (MVS/ESA SP) system.

   If your installation uses installation-written languages or symbolic language names other then the standard three-character codes that are provided, these languages and symbolic names must be defined to the MVS message service.

2. Allocate space for each run-time message file.

   This run-time message file must be a VSAM linear data set. First, allocate a VSAM linear data set for each run-time message file, then use the MVS message compiler to format each install message file to a run-time message file.
3. Use the MVS message compiler to format the install message file into a run-time message file.

   The input to the compiler is the install message file PDS. The output from the compiler is the run-time message file allocated in step 2.
4. Create installation exits if you want to tailor MMS processing.

   IBM provides two installation exits that an installation can use to customize MMS processing. Specify the exit names in the MMSLSTxx member of SYS1.PARMLIB.
5. Update the following SYS1.PARMLIB members to initialize values for MMS:
   - MMSLSTxx, to define the available languages for message translation and other message translation processing
   - CNLcccxx, to define the date and time formats used in the display of translated messages
   - CONSOLxx, to specify the MMSLSTxx member in effect for the system
6. Activate MMS.

   You can use the INIT statement in CONSOLxx to activate MMS at initialization. The operator can activate MMS using the SET MMS command.

Specifying help data sets

You can use the HELP statement in SYS1.PARMLIB member IKJTSOxx to define help data sets for different languages. For the list of languages that IBM provides help data sets for, see the appropriate installation manual.

On the HELP statement you can:
- Specify help for any number of languages
- Specify up to 255 data sets to be searched for help text in a particular language

To define help data sets in IKJTSOxx, do the following:
1. If you have not already done so, copy the sample member IKJTSO00 from SYS1.SAMPLIB to SYS1.PARMLIB. You may have already copied IKJTSO00 to specify other installation defaults.
2. You can create alternate members using the IKJTSOxx naming convention.
3. Edit your IKJTSOxx member in SYS1.PARMLIB and locate the HELP statement.
4. Specify the operands for your installation requirements.

For more information about SYS1.PARMLIB, see *z/OS MVS Initialization and Tuning Reference*.

```
                         ┌─,─────────┐
                         ▼           │
►►──HELP──language──(────'dsname'────┴──)──────────────────────────────►◄
```

language
: specifies the language of the associated help data sets. You must specify the three-character language code for the language. MMSLSTxx contains a list of valid language codes.

dsname
specifies the name of the data set containing the help text for the associated language. You may specify from 1 to 255 data sets, separating each with a comma. The list of data set names must be enclosed in parentheses.

Note: When setting up the HELP parmlib statement, consider the DYNAMNBR parameter on the JCL procedure used by LOGON. Each data set added increases the number of the data sets dynamically allocated during the TSO/E session.

A sample SYS1.PARMLIB HELP statement is shown in the following example.

```
HELP ENU('SYS1.HELP','SYS1.HELP.MIGLIB')
     ESP('SYS1.ESP.HELP')
     JPN('SYS1.JPN.HELP','ANYUSER.TEST.JPN.HELP')
```

When the HELP command locates information, it first checks the SYSHELP ddname for allocation. If the file is allocated, it searches only those data sets. If the file is not allocated, the HELP command checks the user profile table (UPT) for the primary and secondary languages, allocates the corresponding data sets to a file, and searches that file for the help information. If no entry is found for the primary or secondary language, U.S. English is used as the default.

In all cases, if a user's primary language is not supported at the user's terminal or if allocation problems occur, the HELP command attempts to use the secondary language. If both attempts fail, U.S. English is used.

You can use the PARMLIB command to list the current HELP defaults in SYS1.PARMLIB and dynamically update them (with the contents of IKJTSOxx) without having to re-IPL the system. You can also use the PARMLIB command to check the syntax of the HELP statement. For more information about using SYS1.PARMLIB and the PARMLIB command, see *z/OS TSO/E System Programming Command Reference*.

Setting up languages for users

You can specify a primary language and a secondary language to be used for displaying translated information. The primary and secondary languages you specify need to be defined to the MVS message service. If the primary language is not available or not supported by the user's terminal and a secondary language has been specified, information is displayed in the secondary language. If both the primary and secondary languages fail, the default of U.S. English is used.

Languages can be specified in one of the following ways:

- If your installation has RACF, you can use RACF commands to set up languages for users. Installations can set up language segments for individual users with the RACF ADDUSER or ALTUSER command. Users authorized to use the RACF ALTUSER command can establish or change a language segment for themselves. If a user has a language segment, the languages will be saved from session to session.

  Your installation can also use the SETROPTS command to establish a default language for all users on the system. For more information about using the SETROPTS command, see *z/OS Security Server RACF Security Administrator's Guide*.

- Your installation can use the logon installation exit IKJEFLD1 to set up language values for users. For more information about this exit, see “Writing a logon pre-prompt exit (IKJEFLD/IKJEFLD1)” on page 90.
- An individual user can use the TSO/E PROFILE command to set up or change languages. For more information about this command, see *z/OS TSO/E Command Reference*.

Considerations for setting up languages

In deciding how to set up languages for the user, consider how TSO/E locates the language it uses to display information. Figure 21 shows the language search order that TSO/E uses.

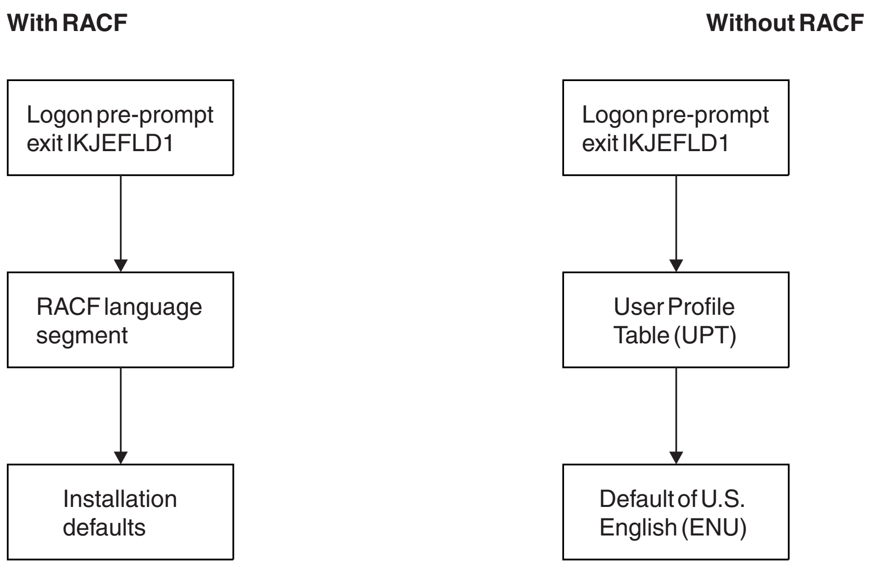

With RACF
Without RACF
Logon pre-prompt exit IKJEFLD1
RACF language segment
Installation defaults
Logon pre-prompt exit IKJEFLD1
User Profile Table (UPT)
Default of U.S. English (ENU)

Figure 21. TSO/E language search order

TSO/E first checks the logon pre-prompt exit, IKJEFLD1, if it exists. If a language is not defined in IKJEFLD1, then TSO/E takes one of two paths.

If you have RACF installed, TSO/E uses the primary and secondary languages from the user's RACF language segment if one has been defined. The language segment reflects any changes made by the user with the TSO/E PROFILE command. If a language segment has not been defined for the user, TSO/E uses the installation defaults established using the RACF SETROPTS command. If you have not specified language defaults, mixed case U.S. English (ENU) is used as the primary and secondary languages.

If you do not have RACF installed and your version of the user profile table (UPT) supports languages, TSO/E uses the primary and secondary languages from the UPT. The default values in the UPT for the primary and secondary languages is mixed case U.S. English (ENU), unless the user changes these using the TSO/E PROFILE command.

Chapter 21. Security considerations for customizing TSO/E

With RACF installed, your installation can use security enhancements. This chapter briefly explains how the security enhancements affect customizing TSO/E, and gives references to additional information.

TSO/E user identification

If your installation uses security labels, and your installation uses the UADS and broadcast data sets to maintain users, these data sets should be specifically defined to RACF. For information about security labels and the UADS and broadcast data sets, see Chapter 24, "Working with the UADS and broadcast data set," on page 199.

Your installation can maintain TSO/E users using the RACF data base by converting your existing UADS data base to RACF. For more information about converting from a UADS data base to a RACF data base, see Chapter 25, "Using the RACF data base to maintain TSO/E users," on page 209.

Security label (SECLABEL) at logon

If your installation is using security labels, the SECLABEL field is available on both the LOGON command and the LOGON panel. The SECLABEL set at LOGON time is valid for the length of the TSO/E session.

If you write a logon pre-prompt exit (IKJEFLD1) for your installation, you can specify a security label to be used for the session. For information about security labels and customizing the logon process, see Chapter 8, "Customizing the logon and logoff process," on page 83.

Protecting user's messages

If your installation plans to use security labels to protect user's messages, each user should have an individual user log. For information about setting up individual user logs, see "Converting from using the broadcast data set to user logs" on page 339 and "Security protected user logs" on page 340.

To ensure that the user's messages are protected in an individual user log and access to the messages is based on security label, the operand values in the SEND PARMLIB parameter of the IKJTSOxx member of SYS1.PARMLIB should be changed as follows:
- LOGNAME to *logname*
- USEBROD to `OFF`
- MSGPROTECT to `ON`

For more information about specifying the operands in the SEND PARMLIB parameter, see Chapter 36, "Customizing how users send and retrieve messages," on page 333.

Accesses to spool data sets

You can control what job names a TSO/E user is allowed to submit by creating a profile for each jobname in the RACF resource class JESJOBS. You can also control what jobs a TSO/E user is allowed to cancel by creating a profile for CANCEL in the RACF resource class JESJOBS.

You can control spool access of the SYSOUT data sets for the TSO/E OUTPUT command by using the RACF resource class JESSPOOL.

If your installation uses the RACF resource classes JESJOBS and JESSPOOL, a sample exit is supplied in member IKJEFF5X of SYS1.SAMPLIB that may be used to replace the IBM supplied exit, IKJEFF53. This sample exit allows the JESJOBS and JESSPOOL classes to control the jobname restrictions when these classes are active. If you are using an installation-written exit, you may wish to look at the sample exit in SYS1.SAMPLIB to determine if you need the same checking of the JES classes in your exit.

For information about setting up the RACF resource classes JESJOBS and JESSPOOL, see *z/OS Security Server RACF Security Administrator's Guide*. For information about the sample exit in SYS1.SAMPLIB, see "TSO/E sample exit" on page 308.

TSO/E TRANSMIT and RECEIVE commands

To allow TRANSMIT and RECEIVE access to the NAMES data set from any security label, the installation-defined NAMES data sets must be allocated with a SECLABEL of SYSLOW. For more information, see "Allocating NAMES and log data sets" on page 398.

If security label checking determines a receiving user is never authorized for the proper security label, the data set may be deleted. The two JES exits (Exit 38 for JES2, IATUX60 for JES3) can be written to hold or reroute data sets that would otherwise be deleted.

Chapter 22. Customizing the TSO/E health checks

TSO/E provides several health checks that are registered with the Health Check facility during TSO/E initialization. These checks are programs or routines that identify potential problems before they impact your availability or cause an outage. The health checks supported by TSO/E take advantage of the IBM Health Checker for z/OS framework. For more detailed information, see *IBM Health Checker for z/OS: User's Guide*.

TSO/E provides the following health checks:

- **TSOE_USERLOGS** - This check determines if user logs have been implemented and issues an exception message if they are not in use. When user logs have not been implemented, job notifications and other messages sent to individual users are saved in the system broadcast data set, instead. Utilizing user logs reduces contention on the broadcast data set and reduces the risk of problems with the broadcast data set that could slow or prevent TSO/E logons.
- **TSOE_PARMLIB_ERROR** - This check detects if serious errors occurred during PARMLIB initialization for TSO/E. An exception message is issued if an unrecognized operand was found in the IKJTSOxx member, a syntax error was encountered during the processing of a specific option, or the pointers to key TSO/E control blocks were left uninitialized. TSO/E processing might not work as desired due to the uninitialized or default settings used a result.

Each check includes a set of pre-defined values, such as:

- Interval, or how often the check will run
- Severity of the check, which influences how check output is issued
- Routing and descriptor codes for the check

You can update or override some check values using either SDSF or statements in the HZSPRMxx parmlib member or the MODIFY command. These are called installation updates. You might do this if some check values are not suitable for your environment or configuration.

For example, to delete the TSOE_USERLOGS health check temporarily, issue the following command from the MVS console:

```
F       hzsproc,DELETE,CHECK=(IBMTSOE,TSOE_USERLOGS)
```

A check that has been deleted can be undeleted later.

```
F       hzsproc,ADDNEW,CHECK=(IBMTSOE,TSOE_USERLOGS)
```

However, deleted health checks can also be reactivated if the Health Checker facility is restarted. To make permanent changes, a policy statement in the HZSPRMxx member is needed. For example, to delete the TSOE_USERLOGS health check permanently the following policy could be defined.

```
ADDREPLACE POLICY STMT(DEL1) DELETE Check(IBMTSOE,TSOE_USERLOGS)
```

Then issue F hzsproc,ADD,PARMLIB=xx to add the HZSPRMxx member containing the new policy statement to the list of members containing the IBM Health Checker for z/OS policy.

For more information on the TSO/E health checks, and the IBM Health Checker for z/OS facility, see *IBM Health Checker for z/OS: User's Guide*.

Part 4. Maintaining the UADS, RACF data base, and broadcast data set

Before users can log on to TSO/E, you must make certain data sets available to the system. These data sets are needed to regulate access to the system and to store messages intended for terminal users. If RACF is used for TSO security, the SYS1.UADS data set is optional, and the security access (UACC value) should be NONE.

- **Regulating Access to the System**

  To regulate access to the system, you can use either the *RACF data base* or SYS1.UADS, the *user attribute data set (UADS)*.

  – If RACF is installed, you can use the RACF data base to regulate access to the system and store information about each TSO/E user. The RACF data base contains profiles for every entity (users, data sets or groups) defined to RACF. If you use the RACF data base to maintain information about TSO/E users, additional information about users is also stored in the RACF data base. For more information about the RACF data base, see *z/OS Security Server RACF System Programmer's Guide*.

    To add, change and delete user IDs, use RACF commands or panels. *z/OS TSO/E Administration* provides an overview of the RACF commands you can use to maintain user information.

  – If RACF is not installed, you must use the user attribute data set. The UADS is basically a list of terminal users who are authorized to use TSO/E.

    To add, change and delete user IDs, use the TSO/E ACCOUNT command. The ACCOUNT command and its subcommands are described in *z/OS TSO/E System Programming Command Reference*.

  To use the RACF database instead of the UADS to store information about each TSO/E user, convert user information that is defined in the UADS to the RACF database. You can convert all or only some user IDs to the RACF data base. TSO/E continues to retrieve user information from the UADS for user IDs that are not converted to the RACF data base. For more information, see Chapter 25, "Using the RACF data base to maintain TSO/E users," on page 209.

  An advantage of using the RACF data base instead of the UADS is that maintenance of user ID information is simpler. You can use RACF commands to maintain TSO/E users and allow them to use the RACF security functions. Adding, changing, and deleting user ID information is simpler using the RACF data base because you use only RACF commands.

- **Storing Messages Intended for Terminal Users**

  The broadcast data set, for example, SYS1.BRODCAST, contains messages intended for terminal users. These messages are sent using the SEND command or the SEND subcommand of OPERATOR. The broadcast data set contains messages intended for all users (notices) and messages sent to individual users (mail). When you use either RACF commands or the ACCOUNT command and its subcommands to add, change or delete user information, the broadcast data set is updated simultaneously.

  As an alternative to using the broadcast data set to store messages sent to individual users (MAIL), you can store messages in a separate user log for each user. However, the broadcast data set is needed to store messages intended for all users (NOTICES). IBM suggests implementing user logs to avoid the single point of failure that exists if all users rely on the broadcast data set for their individual mail.

The broadcast data set is discussed in the sections that follow. For information about individual user logs, refer to Chapter 36, “Customizing how users send and retrieve messages,” on page 333.

- **Maintaining the Required Data Sets**

  If you decide to use the UADS and broadcast data sets to maintain TSO/E users, you must make both data sets available to allow users to log on. That is, if you are installing TSO/E on your system for the first time, you must create both data sets. If you are installing a new release of TSO/E, you may have to reformat both data sets. Periodically, you may have to reformat both data sets to eliminate wasted space caused by adding, changing and deleting information. See Chapter 24, “Working with the UADS and broadcast data set,” on page 199 for more information.

  If you convert from the UADS to the RACF data base, you may have to reformat the broadcast data set periodically to eliminate wasted space caused by additions, changes and deletions. See “Synchronizing the RACF data base with the broadcast data set” on page 213 for more information.

  You can change the number of records in the broadcast data set that are used for messages intended for all users (NOTICES). For more information, see Chapter 26, “Changing the amount of space reserved for notices,” on page 215.

Chapter 23. Content and structure of the UADS and broadcast data set

This topic contains information about UADS and the broadcast data set.

Content and structure of the UADS

The UADS (SYS1.UADS) is basically a list of terminal users who are authorized to use TSO/E. The UADS contains information about each of the users, and is used to regulate access to the system. An entry exists in the UADS for each authorized user of TSO/E. Each entry contains:

- A user identification
- One or more passwords, or a single null field, associated with the user identification
- One or more account numbers, or a single null field, associated with each password
- One or more procedure names associated with each account number. Each name identifies a procedure that may be invoked when the user enters the LOGON command.
- Additional attribute information as described in *z/OS TSO/E System Programming Command Reference* under the ACCOUNT command. The ACCOUNT command and its subcommands allow you to create and update the entries in the UADS.

The organization of the information contained in the UADS is shown in Figure 22 on page 196. Figure 23 on page 197 shows the simplest structure that an entry in the UADS can have, and Figure 24 on page 197 shows a more complex structure.

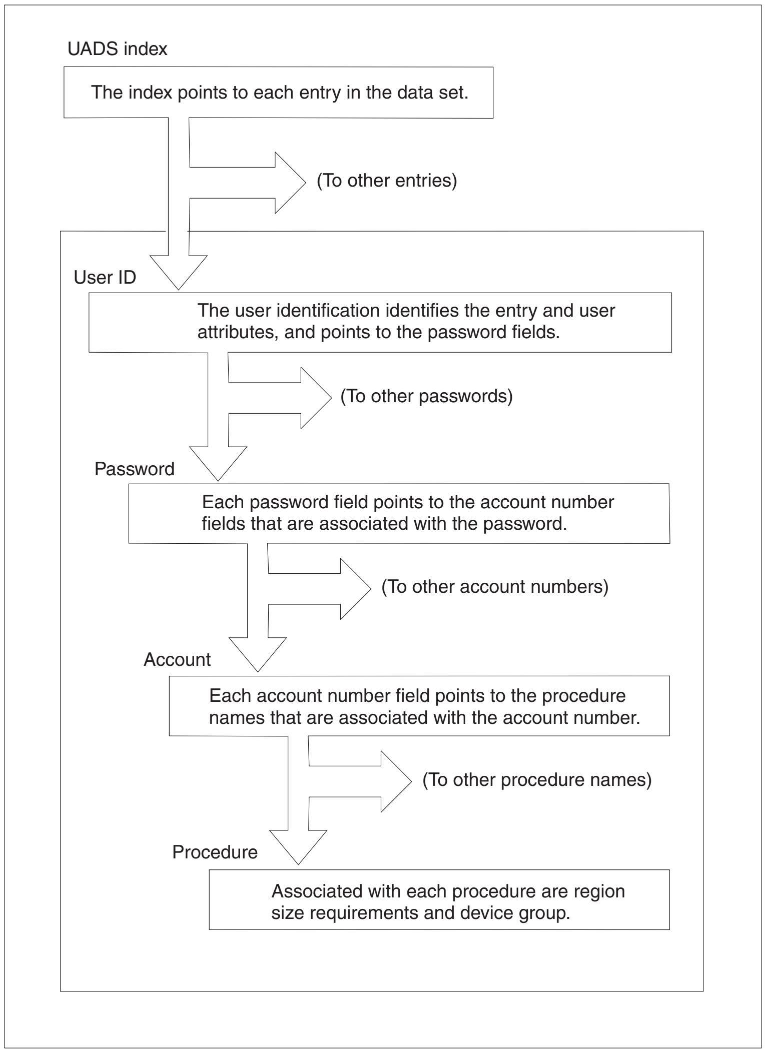

UADS index
The index points to each entry in the data set.
(To other entries)
User ID
The user identification identifies the entry and user attributes, and points to the password fields.
(To other passwords)
Password
Each password field points to the account number fields that are associated with the password.
(To other account numbers)
Account
Each account number field points to the procedure names that are associated with the account number.
(To other procedure names)
Procedure
Associated with each procedure are region size requirements and device group.

Figure 22. Organization of the UADS

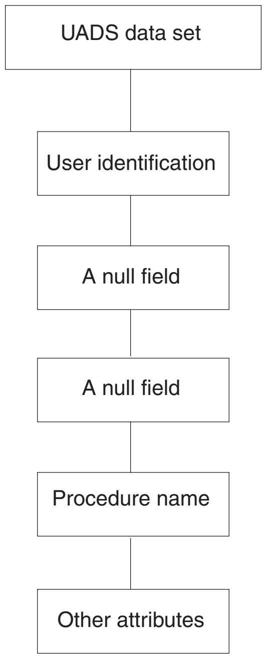


Figure 23. The simplest structure for a typical UADS entry

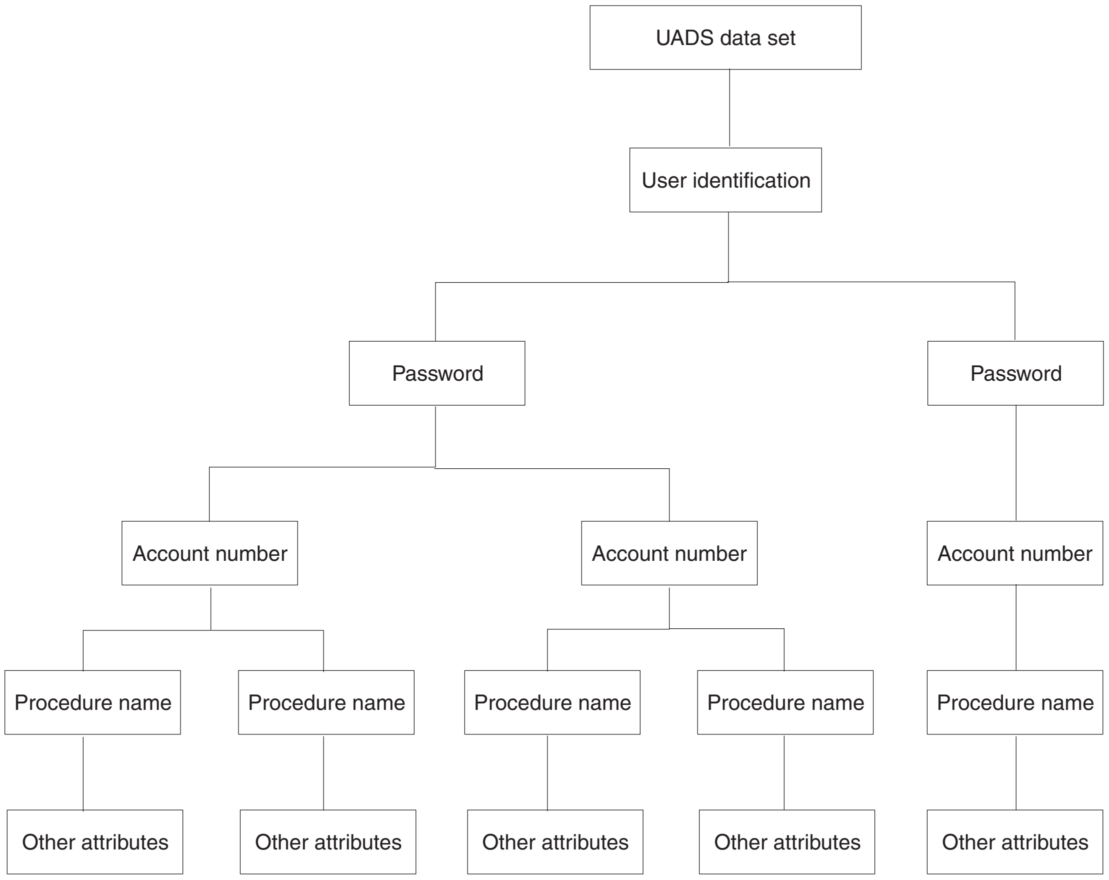


Figure 24. A complex structure for a typical UADS entry

Content of the broadcast data set

The broadcast data set contains messages intended for terminal users. The messages are saved in the broadcast data set using the SEND command or the SEND subcommand of OPERATOR. The broadcast data set contains two sections: the mail section and the notices section. The mail section contains messages intended for particular users; the notices section contains messages intended for all users.

By default, 100 records that are each 129 bytes long are set aside for broadcast messages in the broadcast data set. You can change the number of records for your installation's processing needs. For more information, see Chapter 26, “Changing the amount of space reserved for notices,” on page 215.

Chapter 24. Working with the UADS and broadcast data set

If you are installing TSO/E on your system for the first time, you must create both the UADS and the broadcast data sets. If you are installing a new release of TSO/E, you may have to reformat both data sets. During subsequent operation, you may have to update both data sets by adding, changing, or deleting entries. In addition, you may reformat the data sets to eliminate wasted space caused by periodic additions, changes, and deletions.

When allocating or processing the UADS and the broadcast data set, take the following into account:

- Do not allocate either SYS1.UADS or the broadcast data set on shared DASD that is accessed by more than one processor, unless you use global resource serialization. See "Global resource serialization" on page 207 for additional information.
- If SYS1.UADS is modified by programs other than the reformatting program (UADSREFM) or the ACCOUNT command, unpredictable results may occur later during reformatting, or during processing using the subcommands of ACCOUNT.
- Do not move or copy either data set. If you want to allocate them to different volumes, or change their attributes or sizes, read "Reformatting the UADS and the broadcast data set" on page 201 and "Changing the allocation of the broadcast data set" on page 205.
- With RACF installed, your installation can use security labels. If security labels are being used at your installation:
  - SYS1.UADS should be defined to RACF with a UACC(NONE) and SECLABEL(SYSLOW), and
  - The broadcast data set should be defined to RACF with a UACC(READ) and SECLABEL(SYSLOW).

    Users that have ACCOUNT authority must be given UPDATE authority to the UADS and broadcast data set.

    If the RACF SECLABEL class is active and the RACF SETROPTS MLS option is in effect, the user of the ACCOUNT command must log on using the SYSLOW security label to use the ACCOUNT command. For information about the SECLABEL class, the SYSLOW security label, and SETROPTS options, see *z/OS Security Server RACF Security Administrator's Guide*.

When allocating the UADS, reserve enough space to accommodate all of the users you plan to allow to use TSO/E. Each user requires an entry in the directory and a member in the data portion. Choose a block size for the data portion that makes the most efficient use of storage. The block size should be large enough to contain all of the logon data for most users. To determine if the block size for an existing UADS is large enough list the data set's members. If several users have more than one entry, you should increase the block size. Note that one block is used for each user record.

When creating or enlarging the broadcast data set, reserve enough space to ensure that the data set does not become full. If the data set does become full, additional messages cannot be stored. In addition, reallocating the data set causes the contents to be lost.

When creating a new UADS or broadcast data set on the same volume as the old data set, TSO/E can use the new data sets without an IPL, however, we suggest that you IPL your system. If you are creating a new UADS or broadcast data set on a different volume, you must IPL your system.

Creating the UADS and the broadcast data set

You can create the UADS and the broadcast data set either from a terminal session or by executing the terminal monitor program from a batch job. Use the ACCOUNT command and its subcommands to create entries in the UADS and in the broadcast data set. Specifically, use the ACCOUNT command to:

- Add new entries (ADD subcommand)
- Delete entries (DELETE subcommand)
- Build a new broadcast data set, and synchronize it with an existing UADS (SYNC command or SYNC subcommand of ACCOUNT)

Creating from a terminal

To create the UADS and the broadcast data set from a terminal, add a LOGON procedure named IKJACCNT to the procedure library. During system installation, one user ID (IBMUSER) is copied into the newly-created UADS. IBMUSER is authorized to use one LOGON procedure, IKJACCNT. A sample IKJACCNT LOGON procedure, which is also in IP01.PROCLIB for ServerPac users, follows.

```
//IKJACCNT  EXEC   PGM=IKJEFT01,DYNAMNBR=10
//SYSUADS   DD     DSN=SYS1.UADS
//SYSLBC    DD     DSN=SYS1.BRODCAST
```

Perform the following steps:

1. Activate VTAM.
2. Log on using the following command:
   ```
   logon ibmuser nonotices nomail
   ```
   The keywords NONOTICES and NOMAIL are needed to prevent the LOGON processor from accessing the broadcast data set before the data set is formatted.
3. Enter the ACCOUNT command and issue the SYNC subcommand to format a skeleton of the broadcast data set.
4. Issue ADD subcommands to add the new user IDs to both the UADS and the broadcast data set.
5. For security reasons, you may want to delete the IBMUSER user ID. To delete IBMUSER, perform the following steps:
   a. Log on again with a new user ID that has ACCOUNT authority
   b. Enter the ACCOUNT command and issue the DELETE subcommand to delete the IBMUSER user ID.

      Note: If you plan to install a security server (for example, RACF) that requires the use of IBMUSER for installation, do not delete IBMUSER. Instead, perform the following steps:
      1) After creating the UADS, log off.
      2) Log on with a new user ID that has ACCOUNT authority.
      3) Issue the ACCOUNT command followed by the ADD subcommand. Add a password and any necessary items, such as account number and procedure name for the IBMUSER user ID.

Creating with a batch job

To create the UADS and the broadcast data set without having TSO/E active, execute the terminal monitor program (TMP) as a batch job. Use the ACCOUNT command and its subcommands, as follows:

1. Include the ACCOUNT command. Use the SYNC subcommand with the UADS operand to format a skeleton of the broadcast data set.
2. Use the ADD subcommand to add each user ID and make a corresponding entry in the broadcast data set.
3. For security reasons, you may want to delete IBMUSER (the user ID with ACCOUNT authority provided during system installation) after creating the UADS and ensuring that the data set is usable. To do this, use the DELETE subcommand of ACCOUNT.

   Note: If you plan to install a security server (for example, RACF) that requires the use of IBMUSER for installation, do not delete IBMUSER. Instead, perform the following steps:
   a. Create a new user ID that has ACCOUNT authority.
   b. Issue the ACCOUNT command followed by the ADD subcommand. Add a password and any necessary items, such as account number and procedure name for the IBMUSER user ID.

Figure 25 is a sample listing showing the creation of the UADS and broadcast data set. An explanation of the JCL can be found in Appendix A, “Executing the terminal monitor program,” on page 729.

```
//jobname     JOB      job statement parameters
//            EXEC     PGM=IKJEFT01
//SYSTSPRT    DD       SYSOUT=A
//SYSUADS     DD       DSN=uads-data-set
//SYSLBC      DD       DSN=broadcast-data-set,DISP=SHR
//SYSTSIN     DD       *
ACCOUNT
SYNC UADS
ADD new user ID (see ADD subcommand of ACCOUNT for other operands)
⋮
DELETE (IBMUSER)
END
/*
```

Figure 25. Creating the UADS and the broadcast data set with a batch job

Reformatting the UADS and the broadcast data set

There are four major steps you must follow to reformat the UADS and the broadcast data set:

1. Allocate a new UADS (see “Allocating a new UADS” on page 202).
2. Reformat the UADS and the broadcast data set, using UADSREFM and either the SYNC command or the SYNC subcommand of ACCOUNT (see “Using UADSREFM and the SYNC command or subcommand of ACCOUNT” on page 202).
3. Reset the UADS catalog entry (see “Resetting the UADS catalog entry” on page 203).
4. IPL your system again.

Restriction: Failure to follow these procedures will cause unpredictable results with the use of SYS1.UADS and the broadcast data set. This is because the data set is accessed through the basic direct access method; never copy or move this data set.

Allocating a new UADS

To allocate a new UADS using ISPF/PDF, enter ISPF/PDF option 3.2, and allocate a partitioned data set whose name is different from the name of the existing UADS. To allocate a new UADS using a batch job, include the data set attributes on the DD statement for the new data set.

Figure 26 shows an example of using a batch job to allocate the new UADS. A logical record length (LRECL) of 172 is suggested but not required. The block size must be a multiple of the logical record length and greater than or equal to 256. Optimize the block size to minimize wasted space by following the instructions for optimization in the introduction to Chapter 22, "Working with UADS and the Broadcast Data Set."

```
//jobname  JOB  job card parameters
//         EXEC PGM=IEFBR14
//SYSUT2   DD   DSN=new.UADS.name,UNIT=unit,VOL=SER=volser,
//          DISP=(NEW,CATLG),SPACE=(TRK,(?,?,?)),
//          DCB=(RECFM=FB,LRECL=172,BLKSIZE=1720,DSORG=PO)
/*
```

Figure 26. Allocating the new UADS as a batch job

Using UADSREFM and the SYNC command or subcommand of ACCOUNT

UADSREFM reads an entry from the old UADS, builds a logical copy of that entry, eliminates any wasted space, and writes the newly-formatted entry into the new UADS. The process automatically repeats for each entry in the UADS. However, UADSREFM does not reformat an entry if the user is currently logged on. It writes messages indicating which entries were not reformatted.

You can also use the UADSREFM program to change the block size of the UADS.

To reformat the UADS and broadcast data sets, execute the TMP in the background and include the following in the batch job:

1. The UADS reformatting program, UADSREFM.
2. Either the SYNC command or the SYNC subcommand of ACCOUNT.

   Use the SYNC command or subcommand with the UADS operand to reformat the broadcast data set and to synchronize it with SYS1.UADS. If, for security reasons, you want to delete the IBMUSER user ID after reformatting, use the DELETE subcommand of ACCOUNT.

Figure 27 on page 203 is a sample listing showing the reformatting of the UADS and the broadcast data set. For an explanation of this JCL, see Appendix A, “Executing the terminal monitor program,” on page 729.

```
//jobname     JOB      job card parameters
//            EXEC     PGM=IKJEFT01
//SYSTSPRT    DD       SYSOUT=A
//SYSUADN     DD       DSN=old format uads,DISP=SHR
//SYSUADS     DD       DSN=reformatted uads,DISP=SHR
//SYSLBC      DD       DSN=broadcast-data-set,DISP=SHR
//SYSTSIN     DD       *
UADSREFM
ACCOUNT
SYNC UADS
DELETE (IBMUSER)
END
/*
```

Figure 27. Reformatting the UADS and the broadcast data set

Note: With dynamic broadcast support, the SYSLBC DDNAME is no longer required. If the SYSLBC DDNAME is not specified, the currently active broadcast data set will be used by the job. If the SYSLBC DDNAME is specified, the referenced data set will be used by the job.

The UADSREFM program saves all logon default values except for account number and procedure name. Therefore, users with full-screen logon panels may see a change in those fields after you run UADSREFM.

When a program other than UADSREFM or the SYNC command or subcommand of the ACCOUNT command is used with the UADS data set, unpredictable results may occur. If the UADS data set is damaged, run UADSREFM. UADSREFM lists the entries in alphanumeric order. The last entry listed precedes the damaged entry. You can then delete the damaged entry and add it correctly.

Resetting the UADS catalog entry

After the UADS has been reformatted, you must reset the UADS catalog entry. Use one of the following methods:

- To reset the UADS catalog entry using ISPF/PDF, enter option 3.2 and delete the old UADS, then rename the new UADS to the name of the old UADS.
- To reset the UADS catalog entry using TSO/E commands, use the DELETE command to delete the old UADS, then use the RENAME command.
- To reset the UADS catalog entry using access method services, use the IDCAMS utility. For information on how to use IDCAMS, see *z/OS DFSMS Access Method Services Commands*.
- To reset the UADS catalog entry using a batch job, execute IEHPROGM as shown in Figure 28 on page 204. The batch job:
  1. Deletes the old UADS (`SCRATCH`)
  2. Uncatalogs the old UADS (`UNCATLG`)
  3. Renames the new UADS to the old UADS name (`RENAME`)
  4. Catalogs the new UADS under the old UADS name (`CATLG`)
  5. Uncatalogs the old UADS name (`UNCATLG`).

```
//jobname  JOB  job card parameters
//         EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=device,VOL=SER=volser,DISP=OLD
//DD2      DD   UNIT=device,VOL=SER=volser,DISP=SHR
//SYSIN    DD   *
       SCRATCH  DSNAME=old.UADS.name,VOL=device=volser
       UNCATLG  DSNAME=old.UADS.name
       RENAME   DSNAME=new.UADS.name,old.UADS.name,
                  VOL=device=volser
       CATLG    DSNAME=old.UADS.name,VOL=device=volser
       UNCATLG  DSNAME=new.UADS.name
/*
```

Figure 28. Resetting the UADS catalog entry with a batch job

After you reset the UADS catalog entry by deleting the old UADS, renaming the new UADS to the name of the old UADS, you *must* IPL your system.

Updating the UADS and the broadcast data set

To update the UADS and the broadcast data set from a terminal, ensure that the UADS to be updated is allocated by the SYSUADS DD statement, either in a LOGON procedure or by using the ALLOCATE command. Make the updates (additions, changes, and deletions) using the ACCOUNT command and its subcommands.

To update the UADS and the broadcast data set with a batch job, you can use the sample JCL shown in Figure 29. For an explanation of this JCL, see Appendix A, “Executing the terminal monitor program,” on page 729. If you do not include a DD statement specifying the broadcast data set, the data set must be cataloged.

```
//jobname     JOB     job card parameters
//            EXEC    PGM=IKJEFT01
//SYSTSPRT    DD      SYSOUT=A
//SYSUADS     DD      DSN=SYS1.UADS,DISP=SHR
//SYSLBC      DD      DSN=broadcast-data-set,DISP=SHR,VOL=SER=volser,UNIT=device
//SYSTSIN     DD      *
ACCOUNT
:
(ACCOUNT subcommands for updating)
:
END
/*
```

Figure 29. Updating the UADS with a batch job

Note: With dynamic broadcast support, the SYSLBC DDNAME is no longer required. If the SYSLBC DDNAME is not specified, the currently active broadcast data set will be used by the job. If the SYSLBC DDNAME is specified, the referenced data set will be used by the job.

Switching between broadcast data sets

You can switch dynamically between broadcast data sets by using the SET IKJTSO=xx system command or the TSO/E PARMLIB UPDATE command. An IPL will not be required. In order to switch broadcast data sets, you must first specify the name of the broadcast data set to switch to, along with the associated processing options, in the BROADCAST keyword of the SEND statement of the IKJTSOxx member of parmlib. The switch will be attempted if either of the following conditions apply:

- The name of the broadcast data set in the BROADCAST keyword of the SEND statement in the IKJTSOxx member of parmlib, specified by the command, is different from the broadcast data set name currently being used.
- The data set names are the same but the volumes they reside on differ.

The operator will have to confirm the switch unless NOPROMPT is specified in the parmlib member. However, if the PARMLIB UPDATE ROUTE command is issued on a system where no switch is detected, no prompt occurs; no matter whether this leads to that a broadcast data set switches on one or more of the systems where the settings increased. If you need a separate prompt for each system, use the MVS ROUTE *ALL,SET IKJTSO=xx command.

For more information, see *z/OS MVS Initialization and Tuning Guide*.

Switching the broadcast data set using SET IKJTSO=xx or PARMLIB UPDATE

Use SET IKJTSO=xx or PARMLIB UPDATE to switch from the current broadcast data set to the broadcast data set named in the IKJTSOxx member of parmlib.

The SET command or PARMLIB UPDATE(xx) command will compare the current broadcast data set to the broadcast data set name found in the BROADCAST keyword of the IKJTSOxx member of parmlib. If they are different (or they are the same, but the volumes they reside on differ) and PROMPT is specified in the BROADCAST keyword, the operator will be notified of the pending switch of the broadcast data set. To make the switch, reply YES to the prompt. Any other response will cancel processing of the IKJTSOxx member of parmlib.

If the broadcast data set names are different (or they are the same, but the volumes they reside on differ) and NOPROMPT is specified in the BROADCAST keyword, the broadcast data set will be switched without operator intervention.

For information about the SET IKJTSO=xx command, see *z/OS MVS System Commands*. For information about the PARMLIB UPDATE command, see *z/OS TSO/E System Programming Command Reference*.

Changing the allocation of the broadcast data set

You may want to change the allocation of the broadcast data set, for example, to change its data set attributes, its size, or its location.

To reinitialize the new broadcast data set, you must synchronize it with an existing UADS data set, the RACF data base, or both. Use the SYNC command, or the SYNC subcommand of the ACCOUNT command, with operand BOTH, RACF, or UADS to do so. This will add the current list of authorized users to the new broadcast data set.

You can allocate a new broadcast data set, with either a name different from the current broadcast data set or the same name as the current broadcast data set but residing on a different volume. To do this, perform the following steps:

1. Note the DCB information from the existing broadcast data set.

2. Delete or uncatalog the existing broadcast data set if it has the same name as the new broadcast data set.
3. Allocate the new broadcast data set, modifying the DCB information as needed.
4. Reinitialize the data set.
5. Switch to the new broadcast data set.

You can allocate a new broadcast data set, with the same name as the current broadcast data set, on the same volume. To do this, perform the following steps:

1. Note the DCB information from the existing broadcast data set.
2. Allocate a new broadcast data set with a different name (for example, SYS1.NEWBROD), modifying the DCB information as needed.
3. Delete the current broadcast data set.
4. Rename the new broadcast data set to the name of the current broadcast data set.
5. Reinitialize the data set.

Restriction: Failure to follow these procedures will cause unpredictable results with the use of the broadcast data set. This is because the data set is accessed through the basic direct access method; never copy or move this data set.

Maintaining directory entries in the broadcast data set

You can use the broadcast data set interface routine to add, delete or change a user ID directory entry in the broadcast data set. You can use this interface routine instead of the SYNC command or the SYNC subcommand of ACCOUNT to make minor updates to the broadcast data set. However, this interface routine does not reformat the broadcast data set.

To invoke the broadcast data set interface, write a command processor that issues the IKJIFRIF macro instruction. IKJIFRIF generates the parameter list and linkage to the broadcast data set interface routine.

To use the IKJIFRIF macro instruction:

- Code either the ADD, DEL, or ALT parameters on the macro invocation to specify the user ID that is to be added, deleted or changed.
- Specify the addresses of the following control blocks on the macro invocation:
  - The user profile table (UPT)
  - The protected step control block (PSCB)
  - The environment control table (ECT)

  The address of each of these control blocks is passed to your command processor in the command processor parameter list (CPPL).
- Include the CVT mapping macro (CVT) and the TSVT mapping macro (IKJTSVT) contained in SYS1.MACLIB.

For a description of the syntax of IKJIFRIF, see “IKJIFRIF macro” on page 714. For information on how to write a command processor, see *z/OS TSO/E Programming Guide*.

Global resource serialization

You can place a single version of both SYS1.UADS and the broadcast data set on a shared DASD and access each one from any system in a multisystem complex by using global resource serialization. That is, the resources SYS1.UADS and the broadcast data set may be globally shared. However, to ensure that you can evaluate the applicability of global resource serialization in your installation's TSO/E environment before using it, the major name (SYSDSN) and the minor names (SYS1.UADS and SYS1.BRODCAST) of both data sets are included in the default SYSTEMS exclusion resource name list distributed by IBM. That is, the resources are excluded from global sharing. To ensure proper serialization, if you rename the broadcast data set or switch to a broadcast data set with a different name, you must change the minor name to the name of the new broadcast data set. That is, the resources are excluded from global sharing. In the process of evaluation, plan in advance to investigate and measure:

- Resource requirements (the effort required to merge multiple versions of the two data sets into a single version of each and test the new versions)
- Performance implications (one version of each data set accessed by all users versus *n* versions of the same data sets, each accessed by a subset of those users)

Advantages:

- There are only two data sets to maintain, rather than 2*n*, where *n* is the number of systems in a complex.
- A user can log on to any system in a complex to allow a better workload balance.
- For foreground-initiated background jobs, a user who specifies NOTIFY always receives the job-ended message regardless of which system in a complex processed the job.

Requirements:

1. Merge all existing versions of SYS1.UADS and the broadcast data set into a single version of each data set.
2. Modify the resource name lists, as distributed by IBM, as follows:
   a. Delete the minor names SYS1.UADS and SYS1.BRODCAST (or, if the broadcast data set was renamed, the name of the broadcast data set) from the distributed default SYSTEMS exclusion resource name list.
   b. For SYS1.UADS sharing, add the major name SYSIKJUA as a generic entry in the SYSTEM inclusion resource name list.
   c. For broadcast data set sharing, add the major name SYSIKJBC as a generic entry in the SYSTEM inclusion resource name list.

For information regarding the contents of, and how to modify, the resource name lists, see *z/OS MVS Planning: Global Resource Serialization*.

Broadcast data set in a sysplex

If your installation shares the broadcast data set with other systems, you should customize the IKJTSOxx member of SYS1.PARMLIB to indicate the environment in which the LISTBC command will execute. If you have configured your system as a sysplex, and all of the systems in your global resource serialization complex are also members of the sysplex, you can avoid I/O on the broadcast data set for the display of system notices by setting the SYSPLEXSHR keyword on the SEND parmlib statement to `ON`. When the keyword of the SEND parmlib statement is `ON`, the LISTBC command communicates changes to the broadcast data set to the other systems using sysplex communication facilities.

If your installation shares the broadcast data set with other systems outside the sysplex, you should set the SYSPLEXSHR keyword of the SEND parmlib statement to `OFF`. When the keyword of the SEND parmlib statement is `OFF`, updates are made to the broadcast data set and are read from the broadcast data set.This ensures that all updates are observed by all systems sharing the data set.

Chapter 25. Using the RACF data base to maintain TSO/E users

If RACF is installed, the information that is required for users to log on to TSO/E can be stored in the RACF data base. You can convert user information defined in the UADS to the RACF data base and use the RACF data base to maintain TSO/E users. You can convert user information for all TSO/E users, or only selected users. TSO/E retrieves user information from the UADS for any user IDs that are not converted to the RACF data base.

Using the RACF data base to maintain TSO/E users has several advantages:

- Simplified maintenance of user ID information

  You can maintain TSO/E users and allow them to use the RACF security functions without having to use TSO/E and RACF commands to update the UADS, and use RACF commands to update the RACF data base. Adding, changing and deleting user ID information is simpler using the RACF data base because you use only RACF commands.

Recommendation: For recovery purposes, you should keep a minimal set of user IDs. For additional information, see "Maintaining UADS for recovery mode" on page 212.

Processing of user information

If you convert user information that is defined in the UADS to the RACF data base, the information is processed as follows:

- Some information is defined as a RACF resource. This gives you the ability to allow groups of users to access the same user information without having to define the information for each individual user. In the following table, the UADS information indicates the resource name that needs to be defined in the corresponding RACF resource class. (For example, if the user has an account number 123, then a profile named 123 needs to be defined in the ACCTNUM resource class). The user must have READ access to the profiles defined:

UADS information	RACF resource class name
Account numbers	ACCTNUM
Procedure names	TSOPROC
Performance groups	PERFGRP
TSO/E authority information (The following are the actual resource names): • JCL • OPER • MOUNT • ACCT • RECOVER.	TSOAUTH

- The following user information is stored directly in the RACF data base:
  - Maximum region size
  - USERDATA
  - RBA (relative block address) of the MAIL directory

  - Default SYSOUT destination
  - Default job, message, hold and SYSOUT classes
  - Default UNIT name
  - Console profile settings
  - User profile table (UPT)
  - Logon default information, which consists of the account number, procedure name, region size, performance group, mail, notices, OIDCARD, and TSO/E command

  Values for logon default block information are obtained from the parameters that a user specifies when he logs on to TSO/E. This information is stored in the RACF data base each time the user logs on.
- The password and UADSADRF field are deleted because they are not needed.

The default UNIT name is carried over to the RACF data base with full-screen logon only when the logon default information is stored in UADS. The UNIT is not saved because there is no place on the full-screen panel to specify it. Users who have never logged on to TSO/E will not have logon default information in UADS. If RACONVRT is issued after UADSREFM, and users have not have not logged on and off TSO/E in the meantime, their default account numbers, procedures and UNITs are lost.

Converting to the RACF data base

To convert user information from the UADS to the RACF data base, follow these steps:
1. Convert user information using the RACONVRT command
2. Test the system using the RACF data base
3. Delete user information from the UADS
4. Synchronize the RACF data base with the broadcast data set

After you convert user information to the RACF data base, you can use RACF commands to add, delete, and change the information that defines users to TSO/E. For more information about using RACF commands, see *z/OS Security Server RACF Command Language Reference*.

Using the RACONVRT command

To convert from the UADS to the RACF data base, use the RACONVRT command. Options provided on the RACONVRT command allow you to convert all or some user IDs from the UADS to the RACF data base. To convert all user information, specify the ALL operand. If you do not want to convert all users to the RACF data base, use the INCLUDE or EXCLUDE operands. Use the INCLUDE operand to convert specific user IDs; use EXCLUDE to indicate those user IDs that you do *not* want converted to the RACF data base.

To convert a very large number of users it is suggested to issue several RACONVRT commands and to subdivide the users into groups by means of the INCLUDE/ EXCLUDE operands.

During the conversion process, RACONVRT does not migrate the TSO/E command that was specified on the previous logon. Therefore, the command field in the logon panel contains no data the first time the user logs on after the conversion is complete. If the user specifies a command in the TSO/E command field on the logon panel, TSO/E saves that command for the next logon, if the logoff is successful. For information about the syntax of the RACONVRT command, see *z/OS TSO/E System Programming Command Reference*.

If a user ID has not logged on and off since changes were made to its SYS1.UADS entry, RACONVRT will not include an account number or procedure in the ADDUSER or ALTUSER command for that user ID. When a user ID logs off, the default account number and procedure are saved in SYS1.UADS. RACONVRT uses these defaults when creating the ADDUSER or ALTUSER command to add the user ID to RACF. If the user ID has not logged on and off since changes were made to SYS1.UADS, then defaults do not exist for the user ID in SYS1.UADS. This causes RACONVRT to create an ADDUSER or ALTUSER command without the ACCT or PROC operands.

RACONVRT generates the RACF commands needed to update or create RACF user profiles, and places these commands into members of a partitioned data set. This data set, '*prefix*.IKJ.RACONVRT.CLIST', contains the commands necessary to convert to the RACF data base. RACONVRT creates, but does not execute the RACF commands; therefore, you can edit and change the data set's members to customize the conversion process before issuing the RACF commands.

Table 24 describes the members of the partitioned data set generated by the RACONVRT command.

Table 24. Members created by RACONVRT

Member	Contents
ADDUSER	Commands to: • Define users to RACF and define a TSO segment within each newly created profile • Create a data set generic profile for each new RACF profile to provide data set security
ALTUSER	Commands to define a TSO segment within existing RACF user profiles.
DEFAUTH	Commands to define each of the TSO/E authorities (OPER for OPERATOR, ACCT for ACCOUNT, JCL, MOUNT and RECOVER) as RACF resources and give users authority to access these resources.
DEFPROC	Commands to define logon procedures as RACF resources and give users authority to access these resources.
DEFACCT	Commands to define account numbers as RACF resources and give users authority to access these resources.
DEFPERF	Commands to define performance groups as RACF resources and give users authority to access these resources.
RUN	Commands to invoke the other members in the proper order to complete the conversion to the RACF data base. This member contains commands to execute only the members created or updated by the latest execution of RACONVRT.

To issue the RACF commands, do the following:

1. Issue the SETROPTS command with the CLASSACT operand to activate the new RACF resource classes. The format of the command is as follows:
   ```
   SETROPTS CLASSACT(TSOPROC ACCTNUM PERFGRP TSOAUTH)
   ```
2. Execute the RUN member of the CLIST data set to issue the RACF commands generated by RACONVRT.

3. Issue the RACF SETROPTS command with the RACLIST operand for the new TSO/E resource classes. Issuing this command for the resource classes brings the resource profiles into storage and performs the following functions:
   - Having the profiles in storage eliminates the need to perform I/O on the RACF data base when checking access to resources in the TSO/E classes.
   - When the profile for the ACCTNUM class is in storage, TSO/E LOGON processing can determine whether a user, who is defined to TSO/E in the RACF data base, is authorized to use account numbers. If a user is not authorized to use account numbers, the user is allowed to log on without specifying an account number.
   - If you change any user information within the resource classes, issue SETROPTS RACLIST(classname...) REFRESH to make the changes active.

   Note: If this step is not performed, TSO/E will be unable to determine certain user information such as the default logon procedure and the default account number during logon. TSO/E may prompt users for this information.

Testing the conversion

To test the conversion from the UADS to the RACF data base, you should log off TSO/E and log back on TSO/E.

Deleting user information from the UADS

After you have tested the RACF data base to ensure that the conversion was successful, you can delete information from the UADS for user IDs converted to the RACF data base. To delete information from the UADS, use the DELETE subcommand of ACCOUNT.

Do not delete information from the UADS until each user logs on to TSO/E for the *first* time after the conversion. User information stored in the UADS must be available to obtain a user's current user profile table (UPT), because the conversion does not migrate the UPT to the RACF data base.

Maintaining UADS for recovery mode

Attention: Continue to maintain information for at least one user ID in the UADS. This allows you to log on to TSO/E if RACF is not active or the RACF data base cannot be accessed.

- If you decide to delete the UADS entries for most users, you must meet the following conditions:
  1. You have tested the RACF data base to ensure that the conversion was successful.
  2. The SYS1.UADS data set is available when each user logs on to TSO/E for the first time after the conversion. The UADS must be available to obtain a user's current user profile table (UPT), because the conversion does not migrate the UPT to the RACF data base.
- If you decide to delete the SYS1.UADS data set, you must also remove the reference to it in MSTJCL00 in linklib. The SYSUADS DD statement must either be removed or commented out, or your system will not IPL. For information on how to modify MSTJCL00, see *z/OS MVS Initialization and Tuning Reference*.

Synchronizing the RACF data base with the broadcast data set

To synchronize the RACF data base with the broadcast data set use either the SYNC command or the SYNC subcommand of ACCOUNT.

The SYNC command (or subcommand) must be invoked APF-authorized. Therefore, you must ensure that SYNC is an authorized command by either creating an entry in the authorized command table, or by updating the PARMLIB member IKJTSOxx. For information on updating PARMLIB, see Chapter 11, "Specifying authorized commands/programs, and commands not supported in the background," on page 151.

If you have converted *all* user information from the UADS to the RACF data base, specify the RACF operand when you issue the SYNC command. The RACF operand causes the broadcast data set to be synchronized with the RACF data base. That is, for every user ID defined to TSO/E in the TSO/E segment of the RACF data base, a corresponding entry is made in the broadcast data set.

Figure 30 is a sample listing showing how to synchronize the broadcast data set with the RACF data base. For an explanation of this JCL, see Appendix A, "Executing the terminal monitor program," on page 729.

```
//jobname     JOB     job card parameters
//            EXEC    PGM=IKJEFT01
//SYSTSPRT    DD      SYSOUT=A
//SYSLBC      DD      DSN=broadcast-data-set,DISP=SHR,VOL=SER=volser,UNIT=device
//SYSTSIN     DD      *
SYNC RACF
/*
```

Figure 30. Synchronizing the broadcast data set and RACF data base

Note: With dynamic broadcast support, the SYSLBC DDNAME is no longer required. If the SYSLBC DDNAME is not specified, the currently active broadcast data set will be used by the job. If the SYSLBC DDNAME is specified, the referenced data set will be used by the job.

If you have *not* converted *all* user information to the RACF data base, specify the BOTH operand when you issue the SYNC command. The BOTH operand causes the broadcast data set to be synchronized with both the UADS and the RACF data base. That is, for every user ID defined in either the UADS or the TSO/E segment of the RACF data base, a corresponding entry is made in the broadcast data set.

Figure 31 on page 214 is a sample listing showing how to synchronize the broadcast data set with the UADS and the RACF data base. For an explanation of this JCL, see Appendix A, "Executing the terminal monitor program," on page 729.

```
//jobname     JOB      job card parameters
//            EXEC     PGM=IKJEFT01
//SYSTSPRT    DD       SYSOUT=A
//SYSUADS     DD       DSN=SYS1.UADS,DISP=SHR
//SYSLBC      DD       DSN=broadcast-data-set,DISP=SHR
//SYSTSIN     DD       *
SYNC BOTH
/*
```

Figure 31. Synchronizing the broadcast data set, UADS and RACF data base

Note: With dynamic broadcast support, the SYSLBC DDNAME is no longer required. If the SYSLBC DDNAME is not specified, the currently active broadcast data set will be used by the job. If the SYSLBC DDNAME is specified, the referenced data set will be used by the job.

Chapter 26. Changing the amount of space reserved for notices

The broadcast data set contains messages intended for terminal users. This data set contains two sections:

1. The **mail section** contains messages intended for particular users
2. The **notices section** contains messages intended for all users.

The amount of space you reserve in the broadcast data set for notices depends upon how frequently your installation uses the SEND subcommand of the OPERATOR command to send notices. After you reserve a certain amount of space for notices, the remainder is used for mail. The amount of space required for mail depends on the following factors:

- Whether you use individual user logs to contain mail. If you use user logs for all users, you do not have to reserve any space in the broadcast data set for mail. For information on using individual logs, see Chapter 36, “Customizing how users send and retrieve messages,” on page 333.
- The number of users defined to TSO/E, and how often users request that messages be saved when they are sent.

By default, 100 records are reserved in the notices section of the broadcast data set for messages. You can change the number of records used for notices to suit your installation's processing needs. To change this number, submit a System Modification Program Extended (SMP/E) job that contains the IKJBCAST macro instruction. On the IKJBCAST macro instruction, specify the BCLMT operand to indicate the number of records to be reserved in the notices section.

TSO/E provides a sample job in SYS1.SAMPLIB member IKJBSMPE that shows how to use IKJBCAST. Instructions for using the sample job are included in the SYS1.SAMPLIB member.

For more information on the IKJBCAST macro instruction, see “IKJBCAST macro” on page 709.

As an example, if you want to reserve 250 records in the notices section for messages, you would code IKJBCAST as follows:

```
IKJBCAST  BCLMT=250
```

Part 5. Customizing TSO/E commands

TSO/E provides many commands and subcommands that users issue to interact with TSO/E and the MVS system. Each command and subcommand operates in a specific way depending on the operands a user specifies. For more information about using commands, see *z/OS TSO/E User's Guide*. For more information about the syntax of commands, see *z/OS TSO/E Command Reference* and *z/OS TSO/E System Programming Command Reference*.

TSO/E provides different ways in which you can customize the use of certain commands. This part describes how you can customize these commands to suit your installation's data processing requirements.

One way to customize a particular command is by writing an exit routine. TSO/E provides exits you can use to customize the following commands and subcommands:

- ALLOCATE
- ALTLIB
- CONSOLE and CONSPROF
- EDIT
  - RENUM subcommand
  - MOVE subcommand
  - COPY subcommand
- EXEC
- FREE
- LOGON and LOGOFF
- LISTBC
- OPERATOR SEND
- OUTDES
- OUPUT, STATUS, and CANCEL
- PARMLIB
- PRINTDS
- SEND
- SUBMIT
- TEST
- TESTAUTH
- TRANSMIT and RECEIVE
- TSOLIB

Using exits, you can customize commands in various ways. At a minimum, you can use the exits to change the operands a user specifies.

The chapters in this part describe how you can use the individual exits to customize each of the commands listed above, except LOGON and LOGOFF. For information about the LOGON and LOGOFF commands, see Chapter 8, "Customizing the logon and logoff process," on page 83. For general information about writing exits, see Chapter 2, "Writing exit routines," on page 23.

Chapter 27. Customizing how users allocate and manage data sets

Users issue the TSO/E ALLOCATE command to create new data sets, allocate data sets to a job, and create system output (SYSOUT) data sets. You do not have to perform any tasks to make the ALLOCATE command available to users. You can, however, customize how users allocate data sets.

You can use Storage Management Subsystem (SMS) to manage your system's data and storage and simplify data set allocation for your users. With SMS, you define classes such as data class, management class, and storage class that contain particular allocation, management, and storage attributes for data sets. You define the classes using automatic class selection (ACS) routines. For information about the classes and ACS routines, see *z/OS DFSMSdfp Storage Administration*.

When users allocate a data set, they do not have to specify different operands for particular data set attributes. Instead, they can specify the DATACLASS, MGMTCLAS, and STORCLAS operands on the ALLOCATE command, which simplify data set allocation. For more information about the ALLOCATE command and its operands, see *z/OS TSO/E Command Reference*.

You can use SYS1.PARMLIB member IKJTSOxx to set a default value for the data set disposition specified on the ALLOCATE command. For more information, see "Specifying a default data set disposition for the ALLOCATE command" on page 220.

You can use the MVS allocation input validation routine (exit IEFDB401) to change the information users specify on the ALLOCATE command. If you use Program Control Facility (PCF), you must not use the MVS allocation input validation routine. PCF uses the routine to control allocation. For more information about the MVS allocation input validation routine, see *z/OS MVS Installation Exits*.

When users issue the ALLOCATE command to allocate SYSOUT data sets, they specify operands related to the printer and printing options they want. You can use OUTPUT JCL statements to define output descriptors that associate different printers and options with a single name, the name of the output descriptor. If you have JES2 installed, you and your installation's users can also use the TSO/E OUTDES command to define output descriptors. When users issue the ALLOCATE command, they can specify the OUTDES operand with the name of the output descriptor. They do not need to specify the individual printing options. For more information about using OUTPUT JCL statements to define output descriptors, see "Defining OUTPUT JCL statements" on page 317. For more information about the OUTDES command, see *z/OS TSO/E Command Reference*.

Users can also use the space management service to manage the free space in data sets. Space management can compress, reallocate, or create a data set. Space management uses default allocation and protection data set attributes, but you can change the default values for your own requirements. "Changing the defaults for managing data set space" on page 220 describes the values you can change.

TSO/E provides initialization and termination exits that you can use to customize the ALLOCATE command. The initialization exit receives control before the

ALLOCATE command processor invokes the parse service routine, and can change operands supplied by the user or pass the address of a new command buffer. The termination exit receives control just before the ALLOCATE command terminates processing, and can perform clean-up processing, such as releasing storage obtained by the initialization exit. For more information about the ALLOCATE exits, see "Writing exits for the ALLOCATE command" on page 223.

Specifying a default data set disposition for the ALLOCATE command

With TSO/E, you can use SYS1.PARMLIB member IKJTSOxx to set a default value for the data set disposition specified on the ALLOCATE command.

To specify the default data set disposition for ALLOCATE in IKJTSOxx, do the following:

- If you have not already done so, copy sample member IKJTSO00 from SYS1.SAMPLIB to SYS1.PARMLIB. You may have already copied IKJTSO00 to define other installation defaults.
- You can create alternative members in SYS1.PARMLIB using the IKJTSOxx naming convention.
- Edit the member in SYS1.PARMLIB to contain the appropriate ALLOCATE default for your installation.

In IKJTSOxx, you can set the following values on the ALLOCATE DEFAULT statement:

SHR specifies that the data set already exists and allows more than one person to use it at the same time.

OLD specifies that the data set already exists and gives the user exclusive use of the data set.

If a user issues the ALLOCATE command without specifying a data set disposition, the disposition defaults to the setting in IKJTSOxx. If you do not set an ALLOCATE DEFAULT value in IKJTSOxx, the default is OLD.

The following example shows how to specify the default data set disposition of SHR on a line of IKJTSOxx:

```
ALLOCATE DEFAULT(SHR)
```

When you set a default data set disposition in IKJTSOxx, you can make it take effect immediately using the PARMLIB command with the UPDATE operand. You can also list the current ALLOCATE default using the LIST operand of PARMLIB. For more information about the PARMLIB command, see *z/OS TSO/E System Programming Command Reference*.

Changing the defaults for managing data set space

Space management is an Information Center Facility service that monitors the free space in a data set and either compresses or reallocates the data set when the data set is almost full. It can also allocate a data set that does not exist. Several Information Center Facility services use space management to manage data sets associated with the services. The names and chart (GDDM/PGF) services use space management to prevent ABENDs when they try to add data to data sets that are nearly full.

Users can use space management in CLISTs and programs or directly from the ISPF command line to manage data sets. They invoke space management using the CLIST ICQSPC00 and its parameters. Using the parameters, users specify different space and protection options for a data set. Space management provides default values for the space and protection options. You can change the defaults to suit your installation's processing requirements.

Table 25 lists the space management variables and default settings. To change the defaults, edit the CLIST ICQSPC00 and change the parameters on the PROC statement. The ICQSPC00 CLIST is located in the same library in which the Information Center Facility user CLISTs and REXX execs are installed. This is usually the ICQ.ICQCCLIB library unless your installation installs the user CLISTs and REXX execs in a different library.

For more information about using space management and the variables and their values, see *z/OS TSO/E Programming Services*.

Table 25. Space management parameters and defaults

Variable	Contents or meaning	Default
SPACEFULL	The percent of the space allocated to the data set that must be filled before space management compresses or reallocates the data set.	80
SPACEINCREASE	The percent by which space management increases the primary space allocation of the data set when it reallocates the data set. The value you supply must be an integer.	50
KBYTESFREE	The number of kilobytes of free space the data set must have.	Null
DIRFULL	The percent of a PDS directory that must be filled before space management compresses or reallocates the PDS.	80
DIRINCREASE	The percent by which space management increases the space for the PDS directory when it reallocates the PDS. The value you supply must be an integer.	50
DIRBLOCKSFREE	The number of free directory blocks the PDS must have.	Null
RECALL	Indicates whether a data set migrated by the Data Facility Hierarchical Storage Manager (DFHSM) should be recalled. You can set the parameter to YES to include RECALL on the LISTDSI command that retrieves information about the data set, NO to include NORECALL, or null to omit both RECALL and NORECALL.	Null
PROTECTNEW	Indicates whether a new data set that space management creates should be protected by RACF. You can set PROTECTNEW to YES to have new data sets protected with the universal access in RACFUACC, or NO if you do not want new data sets to be protected by RACF.	NO

Table 25. Space management parameters and defaults (continued)

Variable	Contents or meaning	Default
RACFUACC	The universal access for RACF that space management uses if space management either: • Allocated a new data set, and PROTECTNEW is set to YES • Enlarged a data set, and a level of RACF earlier than RACF 1.7 is installed You can set RACFUACC to NONE, READ, UPDATE, or ALTER.	NONE
ALLOWPASSWORDS	Indicates if space management should compress or reallocate data sets that are password-protected. You can set ALLOWPASSWORDS to YES or NO.	NO
REALLOCATENEW	Indicates if space management should reallocate a data set that is running out of space. You can set REALLOCATENEW to YES to indicate that space management should reallocate it, or NO to indicate that space management should not reallocate it.	YES
ALLOCATENEW	Indicates whether space management should allocate a new data set if the name passed to space management does not match an existing data set. You can set ALLOCATENEW to: • YES, if you want space management to allocate the data set automatically. • NO, if you do not want space management to allocate the data set. • ASK, if you want space management to display a panel asking the user whether the data set should be allocated. The name of the panel is supplied in the parameter ASKPANEL.	ASK
PRIMSPACE	The number of primary space units that space management should allocate for the new data set. PRIMSPACE is used only if ALLOCATENEW is set to ASK or YES.	Null
SECSPACE	The number of secondary space units that space management should allocate for the new data set. SECSPACE is used only if ALLOCATENEW is set to ASK or YES.	Null
UNITS	The units that space management should use when allocating the new data set. UNITS is used only if ALLOCATENEW is set to ASK or YES.	Null
DIRBLOCKS	The number of directory blocks that space management should allocate for the new data set. DIRBLOCKS is used only if ALLOCATENEW is set to ASK or YES.	Null
BLKSIZE	The block size that space management should use when allocating the new data set. BLKSIZE is used only if ALLOCATENEW is set to ASK or YES.	Null
LRECL	The record length that space management should use when allocating the new data set. LRECL is used only if ALLOCATENEW is set to ASK or YES.	Null

Table 25. Space management parameters and defaults (continued)

Variable	Contents or meaning	Default
RECFM	The record format that space management should use when allocating the new data set. RECFM is used only if ALLOCATENEW is set to ASK or YES.	Null
LIKE	A data set that space management can use as a model data set when allocating the new data set. LIKE is used only if ALLOCATENEW is set to ASK or YES.	Null
INFOPANEL	The name of a panel that space management displays while compressing or reallocating a data set.	ICQSPE00
ASKPANEL	The name of the panel that space management displays to ask users whether they want new data sets to be allocated when the data set names they supply do not match existing data sets. ASKPANEL is used only if ALLOCATENEW is set to ASK.	ICQSPE01
VERIFYPARMS	Indicates whether space management should check the syntax of the input parameters. You can set VERIFYPARMS to YES or NO. Setting VERIFYPARMS to NO can improve performance.	YES

Space management invokes IEBCOPY to compress or reallocate a partitioned data set. It invokes IEBGENER to reallocate a sequential data set. You can change the way in which space management invokes IEBCOPY and IEBGENER by changing the settings of the following variables in ICQSIE00:

Variable	Invocation
QCCIEBCP	'TSOEXEC CALL ''SYS1.LINKLIB(IEBCOPY)'''
QCCIEBGR	'TSOEXEC CALL ''SYS1.LINKLIB(IEBGENER)'''

Note: You do not need to include TSOEXEC on these invocations, but including TSOEXEC improves the execution time of the invocation.

Writing exits for the ALLOCATE command

TSO/E users issue the ALLOCATE command to create new data sets, allocate existing data sets for a job, and create system output (SYSOUT) data sets.

TSO/E provides initialization and termination exits that you can use to customize the ALLOCATE command. These exits receive control during ALLOCATE command processing, as follows:

- The initialization exit receives control before the ALLOCATE command processor invokes the parse service routine to syntax check the input parameters.
- The termination exit receives control just before the ALLOCATE command terminates processing.

The following highlights some ways you can use the initialization and termination exits. For more information about how you can use the exits, see “Possible uses” on page 226.

Some ways you can use the initialization exit are:

- Providing a command buffer address to replace the command syntax specified by the user

- Changing the default values for operands of the ALLOCATE command
- Specifying an alternative return code

Some ways you can use the termination exit are:
- Performing clean-up processing
- Specifying an alternative return code

TSO/E-supplied exits

TSO/E does not provide default exit routines for the ALLOCATE command.

Entry specifications

The contents of the registers on entry for the initialization and termination exits for the ALLOCATE command are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions for the initialization exit

The initialization exit receives the standard TSO/E exit parameter list. No exit-dependent data is passed to the initialization exit. For information about the standard exit parameter list and the parameter entry keys, see "TSO/E standard exit parameter list" on page 32.

Parameter descriptions for the termination exit

The termination exit receives the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. If you provide an initialization exit, the termination exit is passed the same parameter entries for the new command buffer and exit-to-exit communication word that were passed to the initialization exit.

Return specifications

The contents of the registers on return from the initialization and termination exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the initialization and termination exits

Table 26 on page 225 shows the return codes that the initialization and termination exits provide.

Table 26. Return codes for the ALLOCATE initialization and termination exits

Return code (decimal)	Description
0	Exit processing was successful. ALLOCATE command processing continues.
12	The exit requested termination. The exit router issues an error message to the user and the ALLOCATE command processor terminates processing with a return code of 12.
16	The exit requested termination. The ALLOCATE command processor issues a message and terminates processing with a return code of 12.

Note:
1. If an exit returns an undefined return code, the ALLOCATE command processor terminates with a return code of 12 and issues an error message to the user, with the exit's reason code included in the message.
2. When requesting that the exit reason code be used as the return code from ALLOCATE, you must insure that the reason code does not duplicate existing ALLOCATE return codes.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exits must be reentrant, refreshable, and reusable.

If the processing done in the initialization exit requires clean-up to be performed, you must write a termination exit. For example, if the initialization exit obtains storage to return a new command buffer to the ALLOCATE command processor, you must provide a termination exit to free this storage.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

If the exits deal with device numbers, for example, on the ALLOCATE UNIT operand, the extended addressability capabilities, introduced with MVS/ESA SP 5.1, must be considered. Four-digit device numbers are preceded by a slash (/) to differentiate them from device types that contain only four-character numerics. Device numbers with less that four characters *may* be preceded by a preceding slash. See also the description on the ALLOCATE UNIT operand in *z/OS TSO/E Command Reference* if this needs to be considered.

Environment

- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY)

Installing the exits

You must name the exits as follows:

Initialization
IKJEFD47

Termination
IKJEFD49

Link-edit each exit as a separate load module. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see "Installing the standard-format exits" on page 39.

Possible uses

Some possible uses of the ALLOCATE exits are described below:

- Change the operands that the user specifies on the command

  You can use the initialization exit to change the operands that users specify on the ALLOCATE command. The initialization exit receives the address of the command buffer. It can change the operands the user specifies on the ALLOCATE command by using a new command buffer. For example, the initialization exit can scan the command buffer to:

  – Look for conflicts with the operands a user specifies and correct any errors. For example, the user cannot specify both the COPIES and OLD operands on the ALLOCATE command.

  – Prevent users from specifying certain operands or certain values for operands.

  To check the command buffer and change its contents, the initialization exit can:

  – Scan the command buffer and decide, based on your own criteria, to change the command the user issued

  – Obtain storage for a new command buffer

  – Build the new command buffer

  – Update the key, length, and data fields for the new command buffer as follows:

Key	X'02'
Length	the length of the new command buffer
Data	the address of the new command buffer

  – Set a return code of 0 and return control to the ALLOCATE command processor.

  The exit must not change the command buffer it receives. It must create a new command buffer and return the address of the new command buffer to ALLOCATE.

  For more information about the command buffer and the new command buffer, see "TSO/E standard exit parameter list" on page 32. For information about the format of the command buffer, see "Command buffer" on page 35.

  You must also write a termination exit to free the storage the initialization exit obtains for the new command buffer. When the ALLOCATE command processor invokes the termination exit, it passes the address of the new command buffer to the termination exit. If the initialization exit returned a new command buffer to the ALLOCATE command in the new buffer parameter, the ALLOCATE command replaces the old command buffer address in the command processor parameter list (CPPL) with the address of the new command buffer. Therefore, when the termination exit is invoked, the command buffer parameter will point to the new command buffer obtained by the initialization exit. For the termination exit, the new command buffer parameter is not used.

- Provide installation-defined pseudo-operands

  If users at your installation print data sets with the same types of characteristics, you can define *pseudo-operands* that are equivalent to two or more ALLOCATE operands. Providing pseudo-operands makes it easier for users to issue the ALLOCATE command. Users need not remember several ALLOCATE operands. They can specify the pseudo- operand.

  For example, you could associate a pseudo-operand named ALTINV with three ALLOCATE operands. The initialization exit can scan the command buffer. If the exit finds the pseudo-operand ALTINV, it can:

  – Obtain storage for a new command buffer

  – Build a new command buffer and replace the pseudo-operand with the appropriate ALLOCATE operands in the new command buffer

  – Update the “Key”, “Length”, and “Data” fields for the new command buffer as follows:

    Key X'02'

    Length
    the length of the new command buffer

    Data the address of the new command buffer

  – Set a return code of 0 and return control to the ALLOCATE command processor.

  You must also provide a termination exit. The termination exit must free the storage that the initialization exit obtained for the new command buffer.

- Monitor how long it takes the ALLOCATE command to complete processing.

  You can use the initialization and termination exits to monitor the approximate time it takes the ALLOCATE command processor to complete processing. When the initialization exit receives control, it can:

  – Invoke the TIME macro

  – Use the exit-to-exit communication word to return the time to the ALLOCATE command processor. The exit updates the “Key”, “Length”, and “Data” fields for the exit-to-exit communication word as follows:

    Key X'01'

    Length
    the length of the data (time)

    Data the data (time)

  – Set a return code of 0 and return to the ALLOCATE command processor.

  When the termination exit gets control, it receives the time from the initialization exit in the exit-to-exit communication word. Before the termination exit returns control to ALLOCATE, it can invoke the TIME macro. The exit can calculate the time difference between the time from the initialization exit (in the exit-to-exit communication word) and the time it receives from issuing the TIME macro. The result is the approximate time it took the ALLOCATE command to complete its processing. The termination exit can include the processing time in a data set. You can then periodically print the data set and review the time calculations.

Chapter 28. Customizing the ALTLIB command

This section describes ways to customize the ALTLIB command.

Writing exits for the ALTLIB command

TSO/E users issue the ALTLIB command to define alternative user-level or application-level libraries containing REXX execs and CLISTs. The defined libraries are searched before, or instead of, the system-level libraries. Therefore, the user does *not* need:

- To allocate application-level REXX execs or CLISTs to the appropriate ddname before implicitly executing them.
- To include all application-level REXX execs or CLISTs in the user's logon procedure.

Table 27 lists the search order of the user-, application-, and system-level libraries. Also shown are the ddnames associated with each library level. These ddnames can be allocated either dynamically by the ALLOCATE command or included as part of a logon procedure.

Table 27. Library search order

Search order	Library level		Associated ddname
1.	User	REXX exec	SYSEXEC
2.	User	CLIST	SYSUPROC
3.	Application	REXX exec	Define with FILE or DATASET operand
4.	Application	CLIST	Define with FILE or DATASET operand
5.	System	REXX exec	SYSEXEC (installation can define this ddname)
6.	System	CLIST	SYSPROC

With the defaults that TSO/E provides, and before an ALTLIB command is invoked, TSO/E searches the system EXEC library (default ddname SYSEXEC) first, followed by the system CLIST library (ddname SYSPROC). Note that your system programmer can change this by

- Defining an alternate ddname of SYSEXEC
- Indicating that TSO/E is not to search the system-level exec ddname of SYSEXEC. Then only the system-level CLIST (SYSPROC) is searched.

You can alter the default library search order by using either the ALTLIB command or the EXECUTIL command.

- Use EXECUTIL to indicate that the system-level exec ddname is to be searched for the duration of the current REXX language processor environment.
- Use ALTLIB to indicate that the system-level exec ddname is to be searched for the duration of the current application. ALTLIB always overrides EXECUTIL within an application.

Use ALTLIB DISPLAY to see which libraries are being searched for.

TSO/E provides initialization and termination exits that you can use to customize the ALTLIB command. These exits receive control during ALTLIB command processing, as follows:

- The initialization exit receives control before the ALTLIB command processor invokes the parse service routine to syntax check the input parameters.
- The termination exit receives control just before the ALTLIB command terminates processing.

TSO/E-supplied exits

TSO/E does not provide default exit routines for the ALTLIB command.

Entry specifications

The contents of the registers on entry for the initialization and termination exits for the ALTLIB command are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions for the initialization exit

The initialization exit receives the standard exit parameter list. However, no exit-dependent data is passed to the initialization exit. For information about the standard exit parameter list and the parameter entry keys, see “TSO/E standard exit parameter list” on page 32.

Parameter descriptions for the termination exit

The termination exit receives the standard exit parameter list. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32. If you provide an initialization exit, the termination exit is passed the same parameter entries for the new command buffer and exit-to-exit communication word that were passed to the initialization exit. Figure 32 on page 231 shows the exit-dependent data that the termination exit receives beginning at offset +36 (decimal) in the parameter list. The parameter entry is described following the figure.

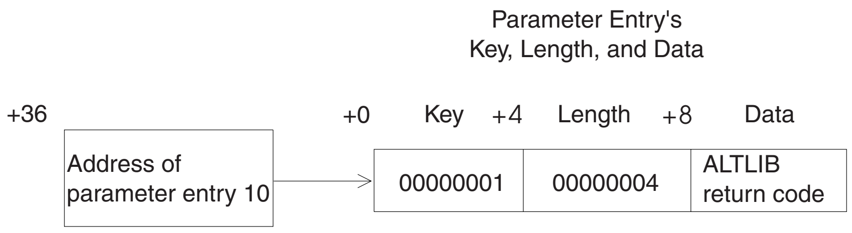


Figure 32. Exit-dependent data for the ALTLIB command termination exit

`ALTLIB Return Code (Parameter Entry 10)`
This parameter entry is the return code from the ALTLIB command processor. For information on the return codes from the ALTLIB command, see *z/OS TSO/E Command Reference*.

Return specifications

The contents of the registers on return from the initialization and termination exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the initialization and termination exits

Table 28 shows the return codes that the initialization and termination exits support.

Table 28. Return codes for the ALTLIB initialization and termination exits

Return code (decimal)	Description
0	Exit processing was successful. ALTLIB processing continues.
12, 16	An error occurred in the exit. The ALTLIB command processor terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the ALTLIB command processor. For more information, see the notes following this table and “Exit reason code” on page 38. If your exit sets a return code of either 12 or 16, you should consider displaying an informational message to the user. You can use the PUTLINE service routine to issue an informational message. See *z/OS TSO/E Programming Services* for more information.

Note:

1. If an exit returns an undefined return code, the ALTLIB command processor terminates without issuing an error message to the user.
2. If an initialization or termination exit sets a reason code that has a key value of X'03', this reason code is used as the return code from the ALTLIB command. However, if both exits indicate that the reason code is to be used as the return code from the ALTLIB command, the reason code from the termination exit overrides that from the initialization exit.
3. When requesting that the exit reason code be used as the return code from ALTLIB, you must insure that the reason code does not duplicate existing ALTLIB return codes.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exits must be reentrant, refreshable, and reusable.

If the processing done in the initialization exit requires clean-up to be performed, you must write a termination exit. For example, if the initialization exit obtains storage to return a new command buffer to the ALTLIB command processor, you must provide a termination exit to free this storage.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

Environment

- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY)

Installing the exits

You must name the exits as follows:

Initialization
IKJADINI

Termination
IKJADTER

Link-edit each exit as a separate load module. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in the following locations; for more information, see “Installing the standard-format exits” on page 39.
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

Possible uses

You can use the initialization exit to change the default values of the ALTLIB command. For example, you can cause the COND operand to be the default.

You can use the termination exit to perform clean-up processing. For example, you can free storage that was obtained in the initialization exit.

Chapter 29. Customizing the CONSOLE and CONSPROF commands

The CONSOLE command allows authorized users to establish an extended MCS console session with MVS console services. When the session is active, users can enter MVS system and subsystem commands and obtain responses to those commands (solicited messages), as well as unsolicited messages.

The CONSPROF command allows authorized users to establish a profile to be used to tailor system message processing during an extended MCS console session. The console profile contains settings for controlling:
- The display of solicited messages (command responses) at the terminal
- The display of unsolicited messages at the terminal
- The size of the tables used to store messages that users do not want displayed

Note: Extended MCS console support is provided in line-mode only.

Before users at your installation can use the CONSPROF command, you must add CONSPROF to the table of authorized commands. Chapter 11, “Specifying authorized commands/programs, and commands not supported in the background,” on page 151 describes how to maintain and update the table.

Individual users must be granted CONSOLE command authority to use the CONSOLE and CONSPROF commands. You can control access to these commands using one of the following methods:
- Use the RACF RDEFINE command to define CONSOLE as a RACF resource belonging to the TSOAUTH RACF class. You can then give selected users access to the CONSOLE resource using the RACF PERMIT command. Users must be defined in a TSO/E segment of the RACF data base.
- Use the logon pre-prompt exit routine, IKJEFLD or IKJEFLD1, to grant or deny CONSOLE command authority. For more information on IKJEFLD and IKJEFLD1, see Chapter 8, “Customizing the logon and logoff process,” on page 83.
- Use the CONSOLE exit routine IKJCNXAC and CONSPROF exit routine IKJCNXCI.

  Note: These exits only change CONSOLE command authority on a temporary basis. If you use this method, it is suggested that you use both IKJCNXAC and IKJCNXCI to grant or deny CONSOLE command authority.

In addition to controlling access to the CONSOLE and CONSPROF commands, you can customize the use and processing of these commands by:
- Defining installation defaults for the message tables used to store messages that users do not want displayed at the terminal.
- Setting up the console profile for a user with the logon pre-prompt exit IKJEFLD1. You can also use the CONSPROF exits to add information to or change a user's console profile. For information about using IKJEFLD1, see Chapter 8, “Customizing the logon and logoff process,” on page 83. The CONSOLE profile can be found in SYS1.MACLIB, member IKJCNCCB, at label CONSOLE_PROFILE.

- Specifying console attributes for each user with CONSOLE command authority. The console attributes control various functions including the types of MVS commands a user can issue during an extended MCS console session, the routing of MVS messages and commands, and the display of MVS message formats. You can use the MVS VARY command to specify the console attributes for a user.

  If you have RACF installed, you can optionally define an OPERPARM segment with the console attributes in the user's RACF profile. If an OPERPARM segment has been defined for a user, the user's console attributes are saved from session to session. You can use the RACF ADDUSER and ALTUSER commands to define OPERPARM segments for users.

  Note: OPERPARM is ignored when OPERCMDS class is not activated.

  If you do not specify console attributes for a user, the system defaults are used. For information about each of the console attributes and their defaults, see *z/OS MVS Planning: Operations*.

  You may provide console attribute information for a user via the console activation exit (IKJCNXAC) in place of using the OPERPARM segment in the user's RACF profile. See "Writing exits for the CONSOLE command" on page 235 for more information.
- Writing CONSOLE and CONSPROF exit routines. You can use these exits to:
  - Grant or deny a user CONSOLE command authority
  - Supply console attributes for a user
  - Control the size of the message tables
  - Take action if a message table reaches 80% or 100% capacity
  - Change the operands a user specifies on the CONSOLE or CONSPROF command
  - Customize the console profile message displayed to the user
  - Add information to or change a user's console profile
  - Release the migration ID

This chapter describes how you can customize the CONSOLE and CONSPROF commands by:
- Defining installation defaults in SYS1.PARMLIB member IKJTSOxx
- Writing CONSOLE and CONSPROF exits

Defining installation defaults for the CONSOLE command

You can use SYS1.PARMLIB member IKJTSOxx to define message processing defaults for the CONSOLE command and its services. IKJTSOxx contains a CONSOLE statement you can use to:
- Specify an initial limit for the number of unsolicited messages to be routed to a user's console without being retrieved or displayed
- Specify an initial limit for the number of solicited messages to be routed to a user's console without being retrieved or displayed
- Specify the maximum number of unsolicited messages to be routed to a user's console without being retrieved or displayed
- Specify the maximum number of solicited messages to be routed to a user's console without being retrieved or displayed

TSO/E provides a sample IKJTSO00 member in SYS1.SAMPLIB. To define installation defaults in IKJTSOxx, do the following:

- If you have not already done so, copy sample member IKJTSO00 from SYS1.SAMPLIB to SYS1.PARMLIB. You may have already copied IKJTSO00 to define other installation defaults.
- Create alternative members in SYS1.PARMLIB using the IKJTSOxx naming convention
- Edit the member in SYS1.PARMLIB and locate the CONSOLE PARMLIB statement
- Specify the operands for your installation's requirements

You can make the CONSOLE defaults in IKJTSOxx take effect immediately using the PARMLIB command. You can also use the PARMLIB command to list the current CONSOLE defaults and to check the syntax of any IKJTSOxx member. For more information about using the PARMLIB command, see *z/OS TSO/E System Programming Command Reference*.

The operands for the CONSOLE PARMLIB statement are:
- INITUNUM
- INITSNUM
- MAXUNUM
- MAXSNUM

For information about these operands, see *z/OS MVS Initialization and Tuning Reference*.

Writing exits for the CONSOLE command

TSO/E provides exits you can use to customize the use and processing of the CONSOLE command. These exits are:
- Pre-parse - control what the user is allowed to specify on the CONSOLE command and in response to the CONSOLE command prompt
- Activation - establish a communication area for related exits, change the initial settings specified by the user, and grant or deny the user CONSOLE command authority
- Deactivation - perform cleanup after other exits and release the migration ID
- 80% message capacity - control the size of the message tables for the user
- 100% message capacity - control the size of the message tables for the user.

For information about how you can use the exits, see “Possible uses” on page 246.

Overview of when the CONSOLE exits receive control

The CONSOLE exits receive control when a user issues the CONSOLE command or responds to a CONSOLE prompt, or when a message table reaches 80% or 100% capacity.
- The **pre-parse exit** receives control before CONSOLE invokes the parse service to parse the command. It is also invoked when the user responds to a console prompt with input. The exit can be invoked several times on each invocation of the CONSOLE command (if the CONSOLE command enters conversational mode). The exit is invoked synchronously in the TSO/E user's address space (for example, as a result of a user entering the CONSOLE command or a subcommand to the CONSOLE command).
- The **activation exit** receives control just before the console session is activated through MVS console services. It is invoked on the first invocation of the CONSOLE command when a console session is not active.

- The **deactivation exit** receives control just after console deactivation occurs. It cleans up the communication area established by the CONSOLE activation exit. If the X'80000000' bit of the flags field is on, this exit is invoked asynchronously in the TSO/E user's address space.
- The **80% message capacity exit** receives control at the first point in which the message tables reach 80% capacity. Eighty percent capacity is defined as 80% of the maximum table size defined in the user's console profile. The exit will not be invoked again for this type of message table until the situation has been resolved and has occurred again. The situation is considered to be resolved when the table decreases to 70% capacity or the resume percentage is updated by the exit. This exit is invoked asynchronously in the TSO/E user's address space.
- The **100% message capacity exit** receives control at the first point in which the message tables reach 100% capacity. One hundred percent capacity is defined as 100% of the maximum table size defined in the user's console profile. The exit will not be invoked again for this type of message table until the situation has been resolved and has occurred again. The situation is considered to be resolved when the table decreases to 90% capacity or the resume percentage is updated by the exit. This exit is invoked asynchronously in the TSO/E user's address space.

TSO/E-supplied exits

TSO/E does not provide default exit routines for any of the CONSOLE exits.

Entry specifications

For all CONSOLE exits, the contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Entry point address

Parameter descriptions for the pre-parse exit

The pre-parse exit receives the standard exit parameter list with the following exceptions:

- The exit reason code field is not used. Its value in the parameter list is:

  Key X'00'

  Length
  X'04'

  Data X'00'

- The exit-to-exit communication word is read-only for this exit. All other exits related to this exit execute in key 1, supervisor state; the exit-to-exit communication word can be set only by them.
- The exit-to-exit word will be zero on the first invocation of this exit because it receives control before the activation exit IKJCNXAC.

For a description of the standard exit parameter list, see "TSO/E standard exit parameter list" on page 32.

Figure 33 shows the exit-dependent data that the pre-parse exit receives beginning at offset +36 (decimal) in the parameter list.

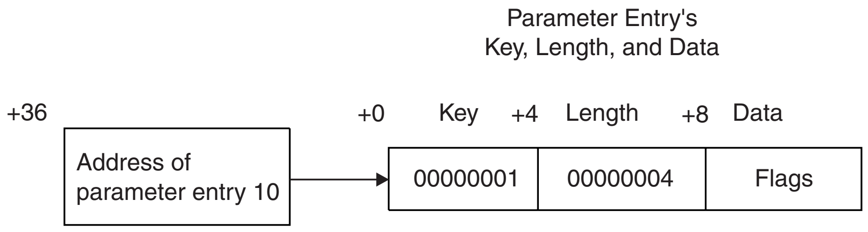


Figure 33. Exit-dependent data for the CONSOLE pre-parse exit

`Flags (Parameter Entry 10)`
This parameter contains a word consisting of flags, in which the command passes indicators to the exit. The flags are described in Table 29.

Table 29. Flags passed to the CONSOLE pre-parse exit

Flag	Meaning
X'80000000'	The command buffer was obtained in conversational mode. If this bit is off, the user is in command mode. Changing this bit has no effect on processing.
X'7FFFFFFF'	Reserved.

Parameter descriptions for the activation exit

The activation exit receives the standard exit parameter list with the following exceptions:

- The parameters are in key 1 storage.
- The command buffer, new command buffer, and exit reason code fields are not used. Their values in the parameter list are:

Key	X'00'
Length	X'04'
Data	X'00'

For a description of the standard exit parameter list, see "TSO/E standard exit parameter list" on page 32.

Figure 34 on page 238 shows the exit-dependent data that the activation exit receives beginning at offset +36 (decimal) in the parameter list.

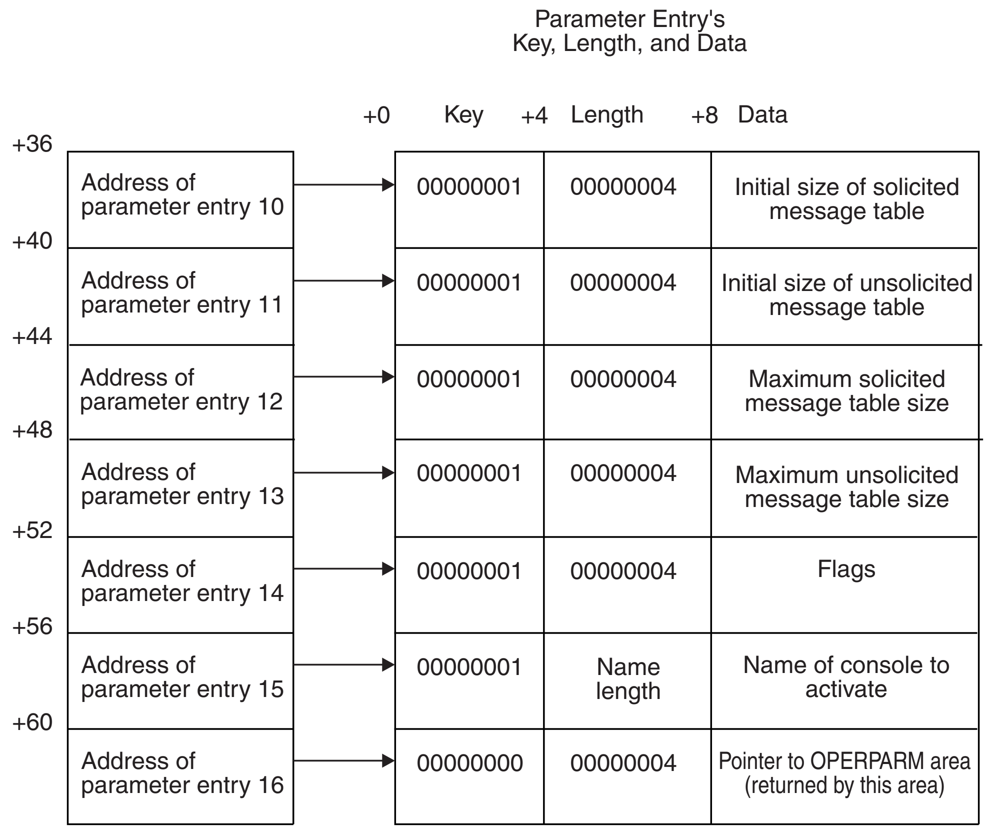

Parameter Entry's
Key, Length, and Data
+0
Key
+4
Length
+8
Data
+36
+40
+44
+48
+52
+56
+60
Address of parameter entry 10
Address of parameter entry 11
Address of parameter entry 12
Address of parameter entry 13
Address of parameter entry 14
Address of parameter entry 15
Address of parameter entry 16
00000001
00000004
Initial size of solicited message table
00000001
00000004
Initial size of unsolicited message table
00000001
00000004
Maximum solicited message table size
00000001
00000004
Maximum unsolicited message table size
00000001
00000004
Flags
00000001
Name length
Name of console to activate
00000000
00000004
Pointer to OPERPARM area (returned by this area)

Figure 34. Exit-dependent data for the CONSOLE activation exit

Solicited Message Table Initial Size (Parameter Entry 10)
This parameter contains the initial size of the solicited message table. This exit can change the initial size if it determines that another value is more appropriate. The value specified by the exit must not be greater than the value specified in parameter 12.

Unsolicited Message Table Initial Size (Parameter Entry 11)
This parameter contains the initial size of the unsolicited message table. This exit can change the initial size if it determines that another value is more appropriate. The value specified by the exit must not be greater than the value specified in parameter 13.

Solicited Message Table Maximum Size (Parameter Entry 12)
This parameter contains the absolute maximum size of the solicited message table. This is the value specified in the active IKJTSOxx member of SYS1.PARMLIB. This parameter is for information only. Any changes to this parameter have no effect on processing.

Unsolicited Message Table Maximum Size (Parameter Entry 13)
This parameter contains the absolute maximum size of the unsolicited message table. This is the value specified in the active IKJTSOxx member of SYS1.PARMLIB. This parameter is for information only. Any changes to this parameter have no effect on processing.

Flags (Parameter Entry 14)
This parameter contains a word consisting of flags, in which the command passes indicators to the exit. The flags are described in Table 30 on page 239.

Table 30. Flags passed to the CONSOLE activation exit

Flag	Meaning
X'80000000'	Solicited messages are to be displayed at the terminal. If this bit is off, solicited messages are not to be displayed. This value can be changed by the exit.
X'40000000'	Unsolicited messages are to be displayed at the terminal. If this bit is off, unsolicited messages are not to be displayed. This value can be changed by the exit.
X'20000000'	The console profile provided to the exit is a default profile. If this bit is off, a profile was obtained from either RACF or the logon exit IKJEFLD1.
X'10000000'	CONSOLE command authority bit. This bit is on if the user has CONSOLE command authority. If this bit is off and the exit determines that the user should have CONSOLE command authority, the exit should turn this bit on. If this bit is on and the exit determines that the user should not have CONSOLE command authority, the exit should turn this bit off.
X'0FFFFFFF'	Reserved.

Name of Console to Activate (Parameter Entry 15)
This parameter contains the name of the console to be activated. If the exit changes the console name, the length field should also be updated to reflect the correct length of the console name.

Pointer to OPERPARM area (Parameter Entry 16)
This parameter allows the installation to return an OPERPARM area (mapped by IEZVG111) to the CONSOLE command for use during activation. See the OPERPARM parameter on the MCSOPER macro in *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU* for details on how to supply this information. The exit must issue the GETMAIN macro to obtain storage for this area and return a pointer to it in this parameter. If the exit returns a pointer, it must also set the key field to 2 (indirect data returned) and the length field to the length of the area being returned. This value is passed to the deactivation exit in parameter 11 of that exit.

Note: You are responsible for issuing the FREEMAIN macro to free this storage in the deactivation exit unless you choose a task-related subpool.

Parameter descriptions for the deactivation exit

The deactivation exit receives the standard parameter list with the following exceptions:

- The parameters are in key 1 storage.
- The command buffer, new command buffer, and exit reason code fields are not used. Their values in the parameter list are:

Key	X'00'
Length	X'04'
Data	X'00'

For a description of the standard exit parameter list, see "TSO/E standard exit parameter list" on page 32.

Figure 35 shows the exit-dependent data that the deactivation exit receives beginning at offset +36 (decimal) in the parameter list.

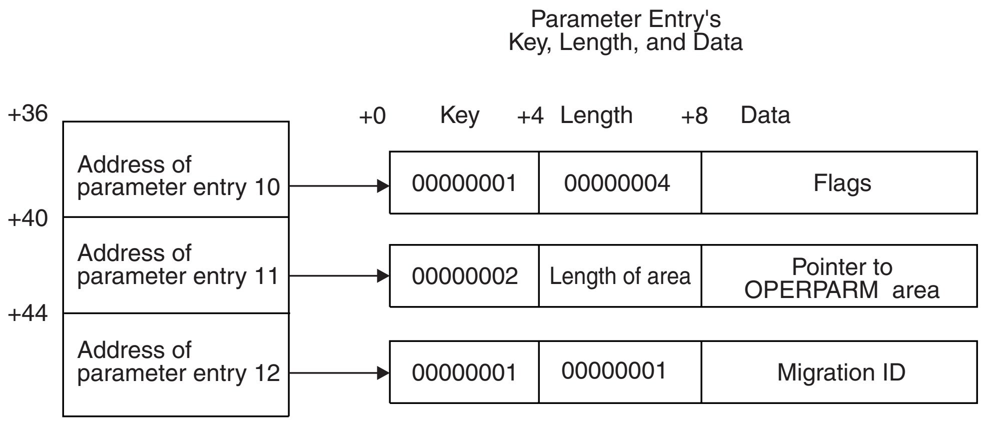


Figure 35. Exit-dependent data for the CONSOLE deactivation exit

`Flags (Parameter Entry 10)`

This parameter contains a word consisting of flags, in which the command passes indicators to the exit. The flags are described in Table 31.

Table 31. Flags passed to the CONSOLE deactivation exit

Flag	Meaning
X'80000000'	If this bit is on, the exit should not use the TSO/E I/O services (for example, PUTLINE, GETLINE, PUTGET, and STACK). Unpredictable results can occur.
X'7FFFFFFF'	Reserved.

`Pointer to OPERPARM area (Parameter Entry 11)`

This parameter contains the pointer to the OPERPARM area provided by the activation exit in parameter 16. This parameter can be used by the FREEMAIN macro to free the OPERPARM area obtained by the activation exit.

Note: This parameter may point to storage which has been already freed by task termination if the subpool, in which the area was obtained with the GETMAIN macro, is task-related.

Note: This parameter will contain

Key X'00'

Length
X'04'

Data X'00'

if no OPERPARM area was provided in parameter 16 of the activation exit.

`Migration ID (Parameter Entry 12)`

This parameter contains the migration ID assigned to this console if one was requested. For more information about migration IDs, see *z/OS MVS Programming: Authorized Assembler Services Guide*. To release the migration ID, use the MCSOPER macro with the RELEASE operand.

Parameter descriptions for the 80% message capacity exit

The 80% message capacity exit receives the standard parameter list with the following exceptions:

- The parameters are in key 1 storage.
- The command buffer, new command buffer, and exit reason code fields are not used. Their values in the parameter list are:

  Key X'00'

  Length
  X'04'

  Data X'00'

Figure 36 shows the exit-dependent data that the 80% message capacity exit receives beginning at offset +36 (decimal) in the parameter list.

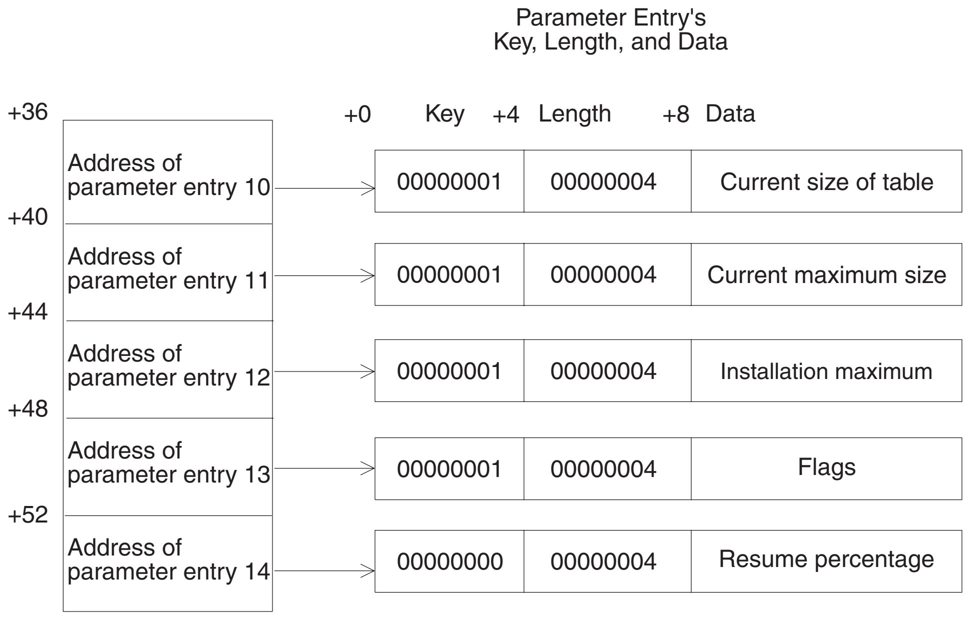

Parameter Entry's
Key, Length, and Data
+36
+40
+44
+48
+52
Address of parameter entry 10
Address of parameter entry 11
Address of parameter entry 12
Address of parameter entry 13
Address of parameter entry 14
+0 Key +4 Length +8 Data
00000001 00000004 Current size of table
00000001 00000004 Current maximum size
00000001 00000004 Installation maximum
00000001 00000004 Flags
00000000 00000004 Resume percentage

Figure 36. Exit-dependent data for the CONSOLE 80% message capacity exit

Current® Table Size (Parameter Entry 10)
This parameter contains the number of messages in the table. This parameter is information only and should not be changed. Any changes are ignored.

Current Maximum Table Size (Parameter Entry 11)
This parameter contains the current maximum size of the message table. If the exit changes the maximum size, the new value must not be greater than the value contained in parameter 12 (installation maximum). If the new value is larger than the value contained in parameter 12, parameter 12 is used.

Installation Maximum Table Size (Parameter Entry 12)
This parameter contains the absolute maximum size allowed for this type of message table as defined in the IKJTSOxx member of SYS1.PARMLIB. This parameter is information only and should not be changed. Any changes are ignored.

Flags (Parameter Entry 13)
This parameter contains a word consisting of flags, in which the command passes indicators to the exit. The flags are described in Table 32 on page 242.

Table 32. Flags passed to the CONSOLE 80% message capacity exit

Flag	Meaning
X'80000000'	The message table is for solicited messages. If this bit is off, the table is for unsolicited messages. This field should not be changed.
X'40000000'	Messages are currently being displayed at the terminal. If this bit is off, the user is not having messages displayed at the terminal. The exit can change this bit to indicate that the user should or should not have messages displayed at the terminal.
X'20000000'	Reserved.
X'10000000'	The TSO/E message service routine should issue a message to the user indicating the situation. By default, this bit is off. If a message is to be issued to the user, the exit must turn this bit on.
X'08000000'	Messages are to be displayed at the terminal regardless of the setting of the X'40000000' flag. At the point in which the message table reaches the resume percentage, the option of the X'40000000' bit is reinstated and processing resumes as normal. By default, this bit is off. If the exit turns this bit on, it must also specify the resume percentage in the parameter list (parameter entry 14).
X'07FFFFFF'	Reserved.

Resume Percentage (Parameter Entry 14)

This parameter contains a value that specifies the percent capacity that the message table must be reduced to before the normal profile setting for displaying messages is resumed (X'40000000' flag). If the exit updates this entry, the key value should be changed to X'01'. This value is used only if the X'08000000' flag is on.

Parameter descriptions for the 100% message capacity exit

The 100% message capacity exit receives the standard parameter list with the following exceptions:

- The parameters are in key 1 storage.
- The command buffer, new command buffer, and exit reason code fields are not used. Their values in the parameter list are:

  Key X'00'

  Length X'04'

  Data X'00'

Figure 37 on page 243 shows the exit-dependent data that the 100% message capacity exit receives beginning at offset +36 (decimal) in the parameter list.

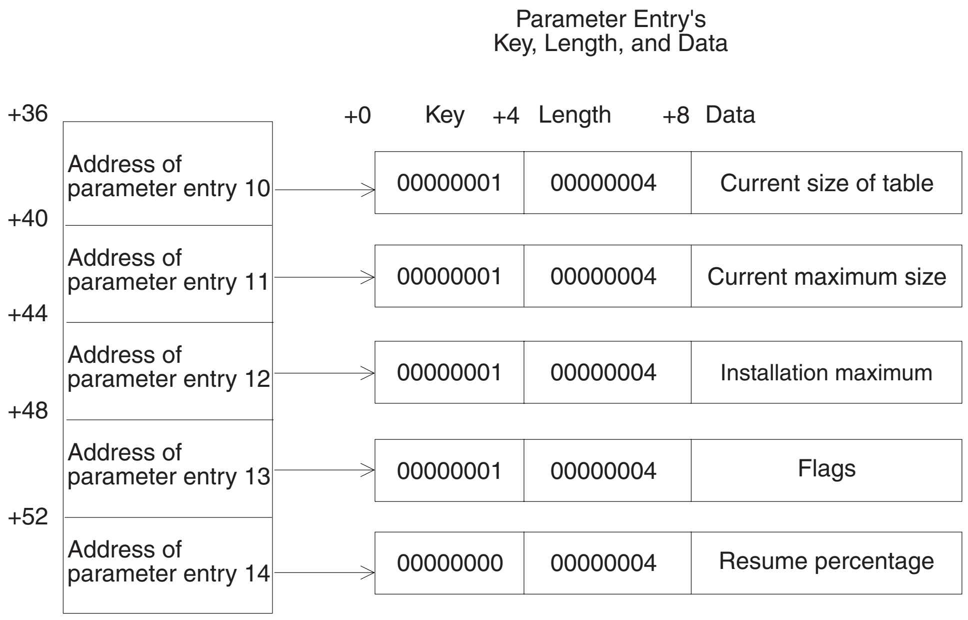

Parameter Entry's
Key, Length, and Data
+36
+40
+44
+48
+52
Address of parameter entry 10
Address of parameter entry 11
Address of parameter entry 12
Address of parameter entry 13
Address of parameter entry 14
+0 Key
+4 Length
+8 Data
00000001 00000004 Current size of table
00000001 00000004 Current maximum size
00000001 00000004 Installation maximum
00000001 00000004 Flags
00000000 00000004 Resume percentage

Figure 37. Exit-dependent data for the CONSOLE 100% message capacity Exit

Current Table Size (Parameter Entry 10)
This parameter contains the number of messages in the table. This parameter is information only and should not be changed. Any changes are ignored.

Current Maximum Table Size (Parameter Entry 11)
This parameter contains the current maximum size of the message table. If the exit changes the maximum size, the new value must not be greater than the value contained in parameter 12 (installation maximum). If the new value is larger than the value contained in parameter 12, parameter 12 is used.

Installation Maximum Table Size (Parameter Entry 12)
This parameter contains the absolute maximum size allowed for this type of message table as defined in the IKJTSOxx member of SYS1.PARMLIB. This parameter is information only and should not be changed. Any changes are ignored.

Flags (Parameter Entry 13)
This parameter contains a word consisting of flags, in which the command passes indicators to the exit. The flags are described in Table 33.

Table 33. Flags passed to the CONSOLE 100% message capacity exit

Flag	Meaning
X'80000000'	The message table is for solicited messages. If this bit is off, the table is for unsolicited messages. This field should not be changed.
X'40000000'	Messages are currently being displayed at the terminal. If this bit is off, the user is not having messages displayed at the terminal. The exit can change this bit to indicate that the user should or should not have messages displayed at the terminal.
X'20000000'	Reserved.
X'10000000'	The TSO/E message service routine should issue a message to the user indicating the situation. By default, this bit is off. If a message is to be issued to the user, the exit must turn this bit on.

Table 33. Flags passed to the CONSOLE 100% message capacity exit (continued)

Flag	Meaning
X'08000000'	Messages are to be displayed at the terminal regardless of the setting of the X'40000000' flag. At the point in which the message table reaches the resume percentage, the option of the X'40000000' bit is reinstated and processing resumes as normal. By default, this bit is off. If the exit turns this bit on, it must also specify the resume percentage in the parameter list (parameter entry 14).
X'07FFFFFF'	Reserved.

Resume Percentage (Parameter Entry 14)

This parameter contains a value that specifies the percent capacity that the message table must be reduced to before the normal profile setting for displaying messages is resumed (X'40000000' flag). If the exit updates this entry, the key value should be changed to X'01'. This value is used only if the X'08000000' flag is on.

Return specifications

On return from the CONSOLE exits, the contents of the registers must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the CONSOLE exits

Table 34 shows the standard return codes that all of the CONSOLE exits support.

Table 34. Standard return codes that all CONSOLE exits support

Return code (decimal)	Description
0	Exit processing was successful. CONSOLE processing continues.
12	Exit processing was unsuccessful. CONSOLE issues an error message to the user and then terminates processing. The console session remains active if it was active before the exit received control. If the exit uses return code 12, it can also pass back an exit reason code to the CONSOLE command processor. For more information about the exit reason code, see “Exit reason code” on page 38.
16	Exit processing was unsuccessful. CONSOLE terminates processing. The console session remains active if it was active before the exit received control. The CONSOLE command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user; for example, using PUTLINE.

If any other non-zero return code is received from the exit, it is assumed to be a return code of 16.

Programming considerations

The CONSOLE exits must follow standard linkage conventions. They must save the registers on entry and restore them on return. The exits must be re-entrant, refreshable, and reusable.

The pre-parse and activation exits and the deactivation exit (when the X'80000000' bit of the flags field is off) can use any of the TSO/E I/O service routines (for example, PUTLINE, GETLINE, PUTGET, and STACK). The activation and deactivation exits must be in key 8 problem program state to invoke the service routines. After using the service routines, these exits must return to key 1 supervisor state. For a description of the service routines, see *z/OS TSO/E Programming Services*.

All dynamic storage obtained for use by the CONSOLE exits, except the pre-parse exit, should be in a protected key and subpool. Because the exits can be invoked asynchronously in the TSO/E user's address space, other tasks can be dispatched during the execution of the exits. Special care must be taken to ensure the integrity of the exit data.

The pre-parse exit can change CONSOLE operands using the command buffer. This exit checks the command buffer it receives and determines whether to change any operands. To change the operands, the exit must:

- Obtain storage from subpool 1 for a new command buffer
- Build the new command buffer
- Update the key, length, and data fields for the new command buffer (parameter entry 2)

For more information about the format of the command buffer, see “Command buffer” on page 35 and “New command buffer” on page 36.

The activation exit can establish a communication area for the exits and place the address in the exit-to-exit communication word parameter. The communication area should be obtained in a protected key so that unauthorized programs cannot alter the storage. In addition, it should be obtained in a non-task-related subpool (254), or the storage should be related to the jobstep task and obtained in an authorized subpool (230). This requirement is because the exits can be invoked under multiple tasks in the address space and the task under which this exit runs might end during the life of the console session.

Note: The pre-parse exit is invoked in key 8 problem program state and cannot alter the communication area.

The activation exit can be used to grant or deny the user CONSOLE command authority for the duration of the console session. If you do not use RACF or the logon pre-prompt exit (IKJEFLD or IKJEFLD1) to control access to the CONSOLE command, use both the CONSOLE activation exit and the CONSPROF initialization exit to give users CONSOLE command authority.

The pre-parse exit can allow installation-specific keywords on the CONSOLE command. If they are allowed, they should be removed before the exit returns control to CONSOLE. A serialization routine should be used to provide for exits that use the same storage area and are called simultaneously.

The 80% and 100% message capacity exits can be invoked once for each type of message table when the criteria for the exit pertaining to that table is met. After the first invocation for that type of message table, they are not invoked again for that type of table until the situation is resolved.

Environment

Activation, Deactivation, 80% and 100% Message Capacity Exits
- State: Supervisor
- Key: 1
- AMODE(31), RMODE(ANY), ASCMODE(PRIMARY)
- Not APF-Authorized

Pre-Parse Exit
- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY), ASCMODE(PRIMARY)
- Not APF-Authorized

Restrictions and limitations

The following exits are invoked asynchronously in the TSO/E address space and should not use any TSO/E I/O service routines.
- 80% message capacity
- 100% message capacity
- Deactivation (when the X'80000000' bit of the flags field is on)

If these exits need to issue a message to the user, they should use the TPUT service.

Installing the exits

You must name the exits as follows:

Pre-parse
IKJCNXPP

Activation
IKJCNXAC

Deactivation
IKJCNXDE

80% message capacity
IKJCNX50

100% message capacity
IKJCNX64

Link-edit each exit as a separate load module. The exits must reside in an authorized library.

Possible uses

The pre-parse exit can be used to control what the user is allowed to specify on the CONSOLE command and in response to CONSOLE command prompts. It can also allow installation-specific keywords on the CONSOLE command. If installation-specific keywords are allowed, they should be removed before the exit returns control to the CONSOLE command.

The activation exit can be used to:
- Establish a communication area for the other exits related to it.
- Supply console attributes for a user

- Change the initial settings that were specified by the user for the console session.
- Enforce the use of console names.
- Grant or deny the user CONSOLE command authority for the duration of the console session.

  Note: If you do not use RACF or the logon pre-prompt exit (IKJEFLD or IKJEFLD1) to control access to the CONSOLE command,use both the CONSOLE activation exit and the CONSPROF initialization exit to grant users CONSOLE command authority. The CONSPROF initialization exit can grant or deny CONSOLE command authority for the duration of the CONSPROF command.

The deactivation exit can be used to clean up after other exits and release the migration ID.

The 80% and 100% message capacity exits can be used to control the size of the message tables for the user. When the user's solicited or unsolicited message table becomes full, these exits can take the following actions so that messages are not lost:
- Make the message table larger if the maximum table size specified in the IKJTSOxx member of SYS1.PARMLIB is greater than the current maximum table size.
- Indicate that the user should have messages displayed at the terminal. This should begin to reduce the number of messages in the message table. After the table decreases to the resume percentage level specified in the parameter list, the processing of messages resumes as normal.
- Indicate that the console session should be terminated.
- Change the dispatching priorities of the tasks so that messages are retrieved faster.

Writing exits for the CONSPROF command

TSO/E provides exits you can use to customize the use and processing of the CONSPROF command. These exits are:
- Initialization - control what the user is allowed to specify on the CONSPROF command and grant or deny the user CONSOLE command authority for the duration of the CONSPROF command
- Pre-display - add information to message IKJ55351I or issue an installation-defined message
- Termination - perform cleanup after other exits

For information about how you can use these exits, see “Possible uses” on page 253.

Overview of when CONSPROF exits receive control

The CONSPROF exits receive control when a user issues the CONSPROF command or before CONSPROF issues message IKJ55351I.
- The initialization exit receives control before CONSPROF invokes the parse service routine to parse the command. The exit can be invoked once per invocation of the CONSPROF command.

- The pre-display exit receives control before the CONSPROF command issues message IKJ55351I. The exit is invoked when either the user enters the CONSPROF command with no keywords or exit IKJCNXCI indicates that message IKJ55351I is to be issued.
- The termination exit receives control before the CONSPROF command returns control to its invoker and is invoked once for each invocation of the CONSPROF command.

TSO/E-supplied exits

TSO/E does not provide default exit routines for any of the CONSPROF exits.

Entry specifications

For all CONSPROF exits, the contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Entry point address

Parameter descriptions for the initialization exit

The initialization exit receives the standard exit parameter list with the following exception:

- The exit reason code field is not used
- Its value in the parameter list is:

  Key X'00'

  Length
  X'04'

  Data X'00'

For a description of the standard exit parameter list, see “TSO/E standard exit parameter list” on page 32.

Figure 38 on page 249 shows the exit-dependent data that the initialization exit receives beginning at offset +36 (decimal) in the parameter list.

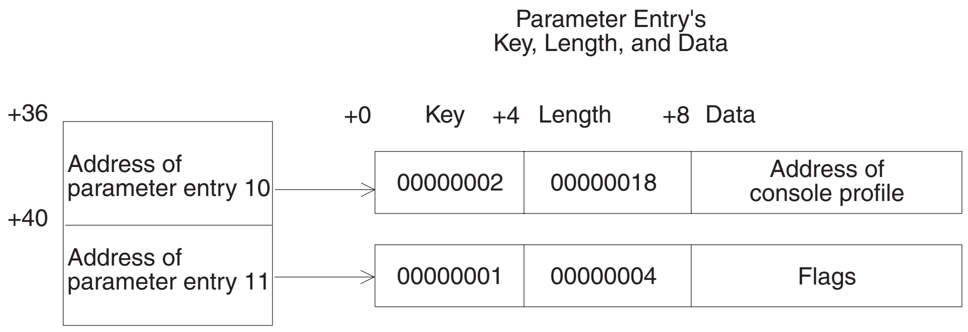


Figure 38. Exit-dependent data for the CONSPROF initialization exit

Console Profile Address (Parameter Entry 10)
This parameter contains the address of the console profile. At the time this exit is invoked, the console profile does not reflect changes requested for this command. The exit can update the profile if it determines that changes are necessary.

Flags (Parameter Entry 11)
This parameter contains a word consisting of flags, in which the command passes indicators to the exit. The flags for this exit's processing are described in Table 35.

Table 35. Flags passed to the CONSPROF initialization exit

Flag	Meaning
X'80000000'	CONSOLE command authority bit. If this bit is on, the user has CONSOLE command authority. If this bit is off, the exit can grant the authority for the duration of the command by turning the bit on. If this bit is on, the exit can remove the user's authority for the duration of the command by turning the bit off.
X'40000000'	The console profile provided to the exit is a default profile. If this bit is off, a profile was obtained from either RACF or the logon exit IKJEFLD1. If the exit alters the profile, it must turn this bit on.
X'20000000'	Specifies that the exit has done all processing and no further action is required by the CONSPROF command. The CONSPROF command ends with return code 0. If the exit sets the X'10000000' bit, the message is displayed before the command ends. By default, this bit is off.
X'10000000'	Specifies that the CONSPROF command should display message IKJ55351I. The pre-display exit IKJCNXCD is invoked before the message is issued. By default, this bit is off.
X'0FFFFFFF'	Reserved.

Parameter descriptions for the pre-display exit

The pre-display exit receives the standard parameter list with the following exceptions:

- The exit reason code field is not used. Its value in the parameter list is:

  Key X'00'

  Length
  X'04'

  Data X'00'

- The new command buffer field is irrelevant at this point in processing and is not used by the CONSPROF command.

For a description of the standard exit parameter list, see "TSO/E standard exit parameter list" on page 32.

Figure 39 shows the exit-dependent data that the pre-display exit receives beginning at offset +36 (decimal) in the parameter list.

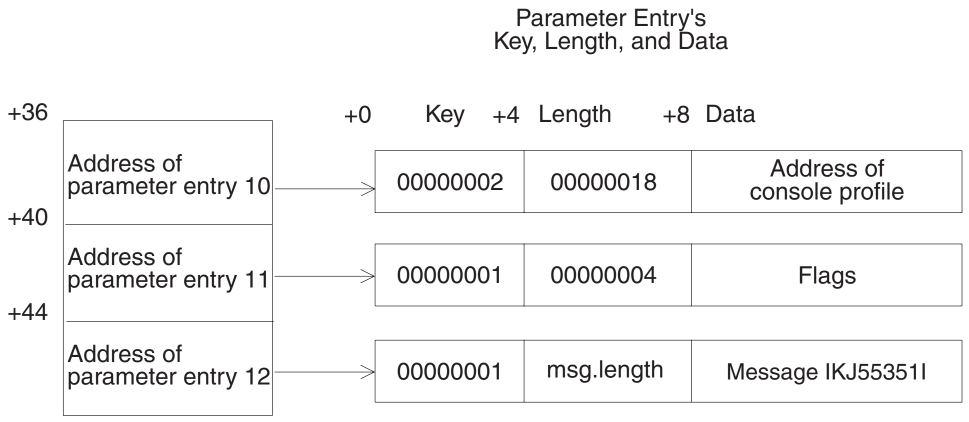


Figure 39. Exit-dependent data for the CONSPROF pre-display exit

Console Profile Address (Parameter Entry 10)
This parameter contains the address of the console profile. At the time this exit is invoked, the console profile reflects changes requested for this command.

Flags (Parameter Entry 11)
This parameter contains a word consisting of flags, in which the command passes indicators to the exit. The flags for this exit's processing are described in Table 36.

Table 36. Flags passed to the CONSPROF pre-display exit

Flag	Meaning
X'80000000'	Issue message IKJ55351I. If the exit turns this bit off, the CONSPROF command does not issue message IKJ55351I. This bit is on at entry to the exit.
X'40000000'	The console profile provided to the exit is a default profile. If this bit is off, a profile was obtained from either RACF or the logon exit IKJEFLD1. This bit is information only to the exit. Changes made to this value have no effect on CONSPROF command processing.
X'3FFFFFFF'	Reserved.

Message IKJ55351I Text (Parameter Entry 12)
This parameter contains the message text of IKJ55351I. IKJ55351I contains the console profile information. The exit can update the message by adding information to the end of the current message text. If the exit updates the message, the length should also be updated to reflect the new length of the message. The message text is in key 8 storage. The buffer can hold a maximum of 253 characters. Should a larger buffer be required, the exit should issue its own message and indicate that the CONSPROF command should not issue message IKJ55351I.

Parameter descriptions for the termination exit

The termination exit receives the standard parameter list with the following exceptions:

- The exit reason code field is not used. Its value in the parameter list is:

  Key X'00'

  Length
  X'04'

  Data X'00'

- The new command buffer field is irrelevant at this point in processing and is not used by the CONSPROF command.

For a description of the standard exit parameter list, see "TSO/E standard exit parameter list" on page 32.

Figure 40 shows the exit-dependent data that the termination exit receives beginning at offset +36 (decimal) in the parameter list.

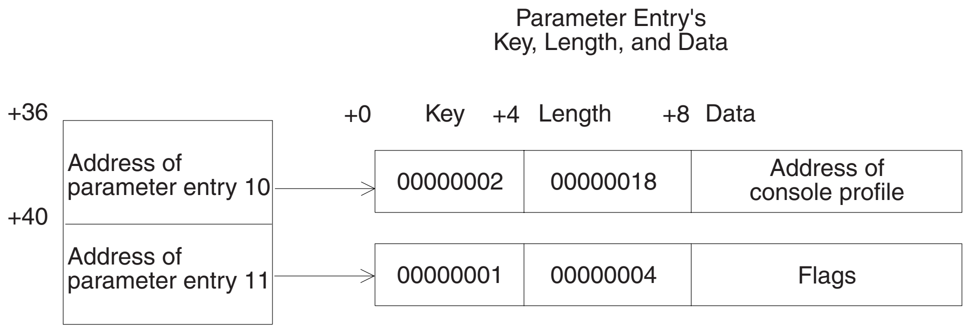


Figure 40. Exit-dependent data for the CONSPROF termination exit

`Console Profile Address (Parameter Entry 10)`
This parameter contains the address of the console profile. At the time this exit is invoked, the console profile reflects changes requested for this command.

`Flags (Parameter Entry 11)`
This parameter contains a word consisting of flags, in which the command passes indicators to the exit. The flags for this exit's processing are described in Table 37.

Table 37. Flags passed to the CONSPROF termination exit

Flag	Meaning
X'80000000'	The console profile provided to the exit is a default profile. If this bit is off, a profile was obtained from either RACF or the logon exit IKJEFLD1. This bit is information only to the exit. Changes made to this value have no effect on CONSPROF command processing.
X'7FFFFFFF'	Reserved.

Return specifications

On return from the termination exit, the contents of the registers must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the CONSPROF exits

Table 38 shows the standard return codes that all of the CONSPROF initialization and termination exits support.

Table 38. Standard return codes that all CONSPROF exits support

Return code (decimal)	Description
0	Exit processing was successful. CONSPROF processing continues.
12	Exit processing was unsuccessful. CONSPROF issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the CONSPROF command processor. For more information about the exit reason code, see "Exit reason code" on page 38.
16	Exit processing was unsuccessful. CONSPROF terminates processing. The CONSPROF command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user; for example, using PUTLINE (for IKJCNXAC and IKJCNXPP) or TPUT (for IKJCNX50, IKJCNX64, and IKJCNXAC).

If a return code of 12 or 16 is received, the CONSPROF command ends. If any other non-zero return code is received from the exit, it is assumed to be a return code of 16.

Programming considerations

The CONSPROF exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exits must be re-entrant, refreshable, reusable, and APF-authorized.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

In some cases, you may not need to write a termination exit. This depends on:

- Whether the processing that the initialization exit performs requires a termination exit to perform clean-up activities.
- How you use the exits to customize CONSPROF processing.

If the initialization exit obtains a system resource, you must write a termination exit to free the resource. For example, if the initialization exit obtains storage to return a new command buffer to the CONSPROF command processor, you must provide a termination exit to free the storage.

The initialization exit can change CONSPROF operands using the command buffer. The exit checks the command buffer it receives and determines whether to change any operands. To change operands, the exit must:

- Obtain storage for a new command buffer
- Build the new command buffer

- Update the key, length, and data fields for the new command buffer (parameter entry 2)
- Set return code 0 and return control to CONSPROF

For more information about the parameter entries, see “TSO/E standard exit parameter list” on page 32. For more information about the format of the command buffer, see “Command buffer” on page 35.

Environment

The CONSPROF exits require the following environment:
- State: Problem Program
- Key: 8
- AMODE(31), RMODE(ANY), ASCMODE(PRIMARY)
- APF-Authorized.

Installing the exits

You must name the exits as follows:

Initialization
IKJCNXCI

Pre-display
IKJCNXCD

Termination
IKJCNXCT

Link-edit each exit as a separate load module. The exits must reside in an authorized library.

Possible uses

The initialization exit can be used to control what the user is allowed to specify on the CONSPROF command. The exit can allow installation-specific keywords on the CONSPROF command. If installation-specific keywords are allowed, they should be removed before the exit returns control to the CONSPROF command.

The initialization exit can be used to grant or deny the user CONSOLE command authority for the duration of the CONSPROF command. If you do not use RACF or the logon pre-prompt exit (IKJEFLD or IKJEFLD1) to control access to the CONSOLE command, use both the CONSPROF initialization exit and the CONSOLE activation exit to maintain CONSOLE command authority. The CONSOLE activation exit can grant or deny the user CONSOLE command authority for the duration of the console session.

The initialization exit can update the console profile to include an installation portion. A word is reserved in the profile that can be set by installation exits for additional customization. This word can be used to point to another block of storage that can contain additional information necessary for CONSOLE command processing.

The pre-display exit can add additional information to message IKJ55351I or issue an installation-defined message. The additional information can be maintained in the installation portion of the console profile if the exits are maintaining more information for the user.

The termination exit should be used to clean up after the exits for the CONSPROF command. This exit can be used to store the additional information kept in the profile area in a permanent place if one exists.

Chapter 30. Customizing the EDIT command

Users issue the EDIT command to enter data into TSO/E. Much of EDIT processing and how you customize EDIT is related to the data set type. TSO/E provides pre-defined data set types with default attributes. Table 39 shows the data set types TSO/E provides and their corresponding default settings.

This section describes the different ways you can customize EDIT processing, which includes:

- Defining data set types and changing the default attributes of data set types. The data set types that TSO/E provides may not suit your editing needs. You can define your own data set types for your installation's requirements. You can also change the attributes of the data set types that TSO/E provides or change the attributes of data set types you define.
- Writing syntax checkers. You can write a syntax checker for any data set types that you define, which checks for syntax errors when a user edits a data set.
- Writing an exit for syntax checkers. You can also write an exit for the syntax checkers you define.
- Changing line numbering. You can write an exit to change how the RENUM, MOVE, and COPY subcommands handle the line numbering of records in the data set.
- Adding subcommands. You can write your own EDIT subcommand processors for additional editing functions.
- Allocating space for utility work data sets. EDIT uses certain algorithms to calculate default space allocation for its utility work data sets. You can preallocate the data sets for your installation.

Table 39. TSO/E pre-defined data set types and attributes

Data set type	Default settings

PLIF	BLOCK=400
	FORMAT=FXDONLY
	FIXED=(80,100)
	CONVERT=CAPSONLY
	CHECKER=PLIFSCAN
	USERSRC=DATASET

FORTE	BLOCK=400
	FORMAT=FXDONLY
	FIXED=(80,80)
	CONVERT=CAPSONLY
	CHECKER=IPDSNEXC
	USERSRC=DATASET

FORTG	BLOCK=400
	FORMAT=FXDONLY
	FIXED=(80,80)
	CONVERT=CAPSONLY
	CHECKER=IPDSNEXC
	USERSRC=DATASET

Table 39. TSO/E pre-defined data set types and attributes (continued)

Data set type	Default settings

FORTH	BLOCK=400
	FORMAT=FXDONLY
	FIXED=(80,80)
	CONVERT=CAPSONLY
	CHECKER=IPDSNEXC
	USERSRC=DATASET

TEXT	BLOCK=3120
	FORMAT=VAR
	FIXED=(0,255)
	VAR=(255,255)
	CONVERT=ASIS
	USERSRC=DATASET

DATA	BLOCK=3120
	FORMAT=FIXED
	FIXED=(80,255)
	VAR=(0,255)
	CONVERT=CAPS
	USERSRC=DATASET

CLIST	BLOCK=3120
	FORMAT=VAR
	FIXED=(0,255)
	VAR=(255,255)
	CONVERT=CAPSONLY
	USERSRC=DATASET

CNTL	BLOCK=3120
	FORMAT=FXDONLY
	FIXED=(80,80)
	CONVERT=CAPSONLY
	USERSRC=DATASET

ASM	BLOCK=3120
	FORMAT=FXDONLY
	FIXED=(80,80)
	CONVERT=CAPSONLY
	PRMPTR=ASM
	USERSRC=DATASET

COBOL	BLOCK=400
	FORMAT=FXDONLY
	FIXED=(80,80)
	CONVERT=CAPSONLY
	PRMPTR=COBOL
	USERSRC=DATASET

FORTGI	BLOCK=400
	FORMAT=FXDONLY
	FIXED=(80,80)
	CONVERT=CAPSONLY
	CHECKER=IPDSNEXC
	PRMPTR=FORT
	USERSRC=DATASET

Table 39. TSO/E pre-defined data set types and attributes (continued)

Data set type	Default settings

GOFORT	BLOCK=3120
	FORMAT=VAR
	FIXED=(80,255)
	VAR=(255,255)
	CONVERT=CAPS
	CHECKER=IPDSNEXC
	PRMPTR=GOFORT
	USERSRC=INCORE

PLI	BLOCK=400
	FORMAT=VAR
	FIXED=(0,100)
	VAR=(104,104)
	CONVERT=CAPS
	PRMPTR=PLIC
	USERSRC=DATASET

VSBASIC	BLOCK=3120
	FORMAT=VAR
	FIXED=(0,80)
	VAR=(255,255)
	CONVERT=CAPSONLY
	PRMPTR=VSBASIC
	USERSRC=INLIST
	DATEXIT=ICDQRNME

Defining data set types and changing the default attributes

If the pre-defined data set types TSO/E provides are not suitable for your installation, you can define your own data set types with specific attributes. You can also change the default attributes of the data set types TSO/E provides for your processing requirements. If you define a data set type, you can change the attributes you specified.

To define data set types or change the attributes, use the IKJEDIT macro instruction and submit a System Modification Program Extended (SMP/E) job to perform the definition or updates. Using IKJEDIT, specify the name of the data set type you are defining or changing and the specific attributes for the data set type. For information about the syntax of IKJEDIT, see “IKJEDIT macro” on page 709.

IKJEDIT generates the following two CSECTs:
- IKJEBEPD, which is link-edited with the IKJEBEPS module in SYS1.CMDLIB
- IKJEBINS, which is link-edited with the EDIT command in SYS1.CMDLIB.

Both CSECTs are required.

TSO/E provides a sample job in SYS1.SAMPLIB member IKJTSMPE that shows how to use IKJEDIT.Instructions for using the sample job are included in the SYS1.SAMPLIB member.

If you change the attributes of a data set type, any attributes that you do not explicitly specify on IKJEDIT are not changed. If you define a new data set type and do not explicitly specify a particular attribute, you get the following default values:

Block size:
3120

Record format:
Fixed

Default LRECL:
Fixed - 80, Variable - 255

Maximum LRECL:
Fixed - 255, Variable - 255

Data conversion:
CAPS

TSO/E does not provide defaults for the following attributes:
- Checker name
- Prompter name
- Prompter input

If you used the EDIT SYSGEN macro to change the default attributes of TSO/E-supplied data set types, the defaults TSO/E provides overlay your changes. After system generation, use IKJEDIT to change the default attributes. By using IKJEDIT, future SYSGENs or IOGENs do not replace your values.

Writing a syntax checker

A syntax checker checks for syntax errors in each statement, according to the data set type. The syntax checker scans each line a user enters, in input mode, when the user edits a data set. Before the syntax checker scans a record, the record is put into the data set. If a syntax error is found, EDIT displays an error message and switches from input mode to edit mode. The user can then use EDIT subcommands to correct the error. For information about EDIT and the SCAN operand, see *z/OS TSO/E Command Reference*.

Syntax checkers are associated with specific data set types. TSO/E provides the following syntax checkers for theassociated data set types:

Data set type	Syntax checker
FORTH	IPDSNEXC
PLI	IPDSNEXC
GOFORT	IPDSNEXC
PLIF	PLIFSCAN

You can write a syntax checker for data set types that you define. You perform the following steps to provide a syntax checker:
1. Write the syntax checking routine
2. Install the routine on your system before system generation
3. Use the IKJEDIT macro instruction to associate the syntax checker with the data set type.

"Record format and interface for syntax checking" on page 259 provides information for writing the syntax checking routine. This includes:
- The format of the records that are passed to syntax checkers

- The standard interface that TSO/E provides that allows the EDIT modules to invoke a syntax checker
- The contents of the control blocks to which the parameter list for the syntax checker points

"Associating the syntax checker with a data set type" on page 262 describes how to use IKJEDIT to associate the syntax checker with a specific data set type.

Record format and interface for syntax checking

Either fixed-length (numbered) or variable-length (numbered) records can be passed to syntax checkers. Figure 41 shows the format of the records that are passed.

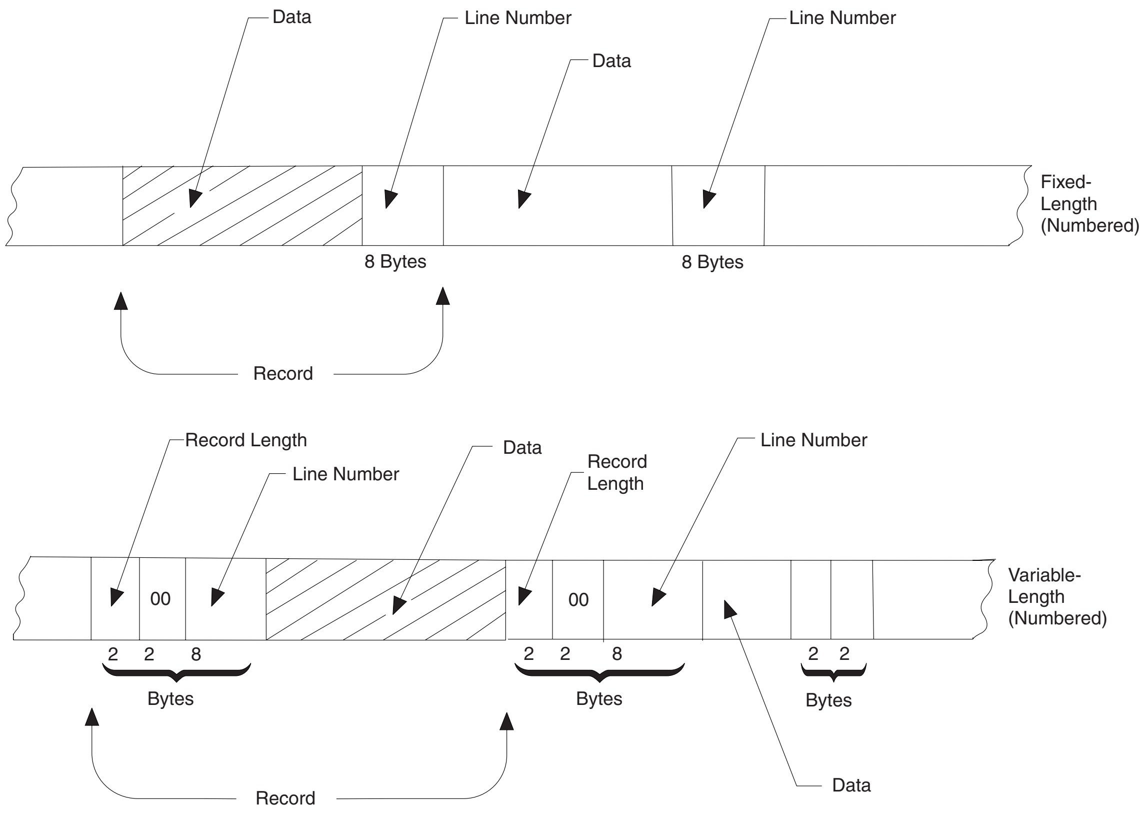


Figure 41. Format of Records Passed to Syntax Checkers

TSO/E provides a standard interface that allows the EDIT modules to invoke any syntax checker. Figure 42 on page 260 shows the EDIT modules that invoke syntax checkers, the standard interface, and the syntax checkers that TSO/E provides. The interface consists of a syntax checker parameter list that you use for writing your syntax checker.

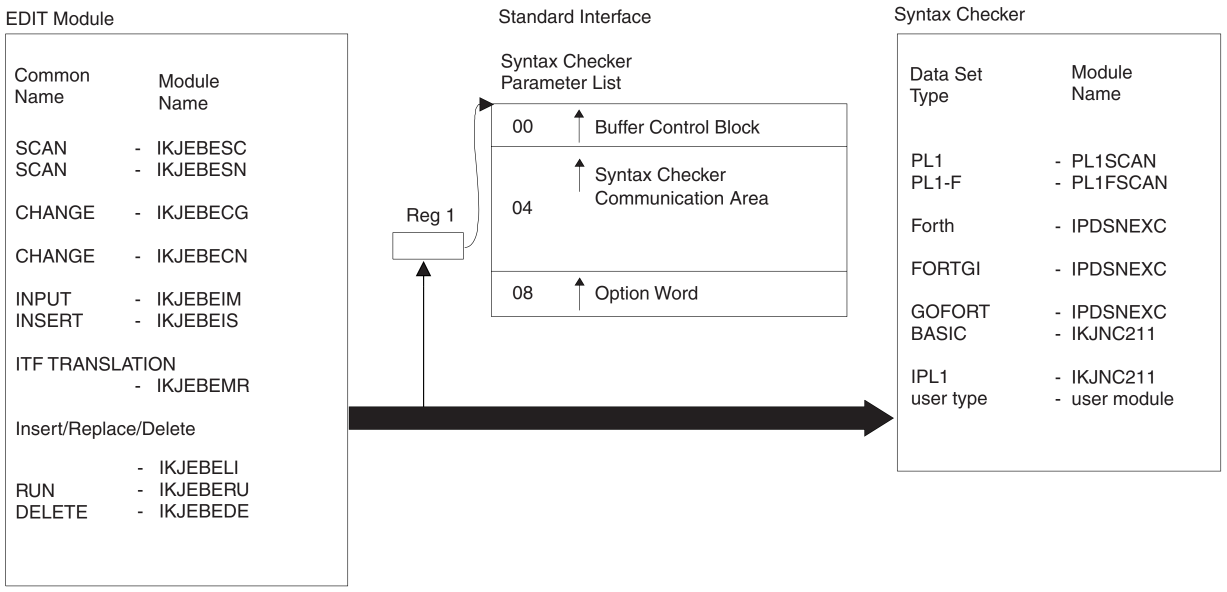


Figure 42. Interface Between the EDIT Program and Syntax Checkers

The parameter list for the syntax checker points to the:
- Buffer control block
- Syntax checker communication area
- Option word

Table 40, Table 41, and Table 42 on page 261 show the contents of the buffer control block, syntax checker communication area, and option word.

Table 40. Contents of the buffer control block

Disp dec.	Disp hex.	Field name	Field size	Contents or meaning
0	0	C	1	Number of records in buffer (maximum – 127). Bit zero is set to 1 when the syntax checker has scanned all records in the buffer.
1	1	Chain	3	Address of next buffer; set to zero if this is the last buffer in chain.
4	4	Record	Variable	Line or lines of source input data to be syntax checked. Can be fixed- or variable-length, numbered or unnumbered.

Table 41. Contents of the syntax checker communication area

Disp desc.	Disp hex.	Field name	Field size	Setting	Meaning (instruction to syntax checker)
				Bits 0-3	(where *n*=0 or 1).
				0*nnn*	First entry — obtain and initialize work area. If a buffer chain is supplied, the syntax checker will set the relative line number counter to zero.
				1*n* 1*n*	Last entry — release the work area and return. Syntax checking is not performed.
				1000	Normal entry — set relative line number to counter zero. Perform syntax checking.
0	0	None	1	110n	Entry after return code 8 (error - buffer checking incomplete) — continue syntax checking.

Table 41. Contents of the syntax checker communication area (continued)

Disp desc.	Disp hex.	Field name	Field size	Setting	Meaning (instruction to syntax checker)
				1001	Entry after return code 12 (complete statements have been checked, but last statement in input buffer is incomplete). If there is no more input (chain address of last buffer or buffer address is zero), syntax check the incomplete statement and return. If there is a new buffer chain, that is, more input, (chain address or buffer address is not zero), resume syntax checking at the incomplete statement.
				Bits 4-7	Reserved
1	1	None	4	xxxx	Address of work area stored by syntax checker on first entry.
4	4	None	4	xxxx	Initial entry — maximum statement size specified when the EDIT defaults for the data set type were set (if 0, checker assumes sufficient storage for largest legal statement is available). Entry after return code 4 (error detected, syntax checking complete, second-level message present), or 8 (error detected, syntax checking incomplete) — address of error message area.
8	8	None	4	xxxx	Initial entry — temporary area. Subsequent entries — address of second error message, if any.
12	C	None	4	xxxx	Temporary storage area used for GETMAIN.

Table 42. Contents of the option word

Disp desc.	Disp hex.	Field name	Field size	Setting	Meaning	Syntax checker
0	0	None	1	X'00'	FORTRAN H level	FORTRAN
				X'03'	GOFORT	FORTRAN
				X'04'	FORTRANG1	FORTRAN
				X'00'	IPLI level	IPLI
				X'01'	BASIC level	BASIC
				xxxx	Value of left source margin	PLIF
1	1	None	1	Bits 0-5	Reserved	FORTRAN
				Bit 6 = 1	FORTRAN G1/H Code and Go definition to be loaded on initial entry.	FORTRAN
				= 0	FORTRAN G1/H Code and Go definition not to be loaded on initial entry.	FORTRAN
				Bit 0 = 1	Entry from INPUT, Insert, linenum, *, CHANGE	IPLI or BASIC
				Bit 1 = 1	Entry from DELETE	IPLI or BASIC
				Bit 2 = 1	Entry from MERGE or RENUM	IPLI or BASIC
				Bit 3 = 1	Translation already complete	IPLI or BASIC
				Bit 4 = 1	Entry from RUN	IPLI or BASIC
				Bits 5-7	Reserved	IPLI or BASIC
				xxxx	Value of right source margin	PLIF

Table 42. Contents of the option word (continued)

Disp desc.	Disp hex.	Field name	Field size	Setting	Meaning	Syntax checker
2	2	None	1	xxxx	Record length of fixed-length records; binary zero, if variable-length records.	All
3	3	None	1	Bit 0 = 0	CHAR 60	PLI or IPLI
				= 1	CHAR 48	PLI or IPLI
				Bit 1 = 0	Line-numbered data set.	All
				= 1	Data set not line-numbered.	All
				Bit 2	Reserved	All
				Bit 3 = 0	Diagnose an incomplete statement.	All
				= 1	Delayed scan – return with code of 12, if last statement in input buffer is incomplete. Immediate scan – possible incomplete statement in buffer.	All
				Bit 4 = 0	Fixed-length records.	All
				= 1	Variable-length records.	All
				Bit 5 = 0	Standard form source input.	All
				= 1	Free form source input.	All
				Bit 6 = 0		
				= 1		
				Bit 7 = 0	SCAN or SCAN ON specified.	All
				= 1	NOSCAN or SCAN OFF specified.	All

The syntax checkers that TSO/E provides determine the attributes of the associated data set type by referring to information that EDIT initialization sets in the option word. For your own data set types and syntax checkers, EDIT initialization does not place attribute information in the option word. You can write an exit routine for your syntax checker that fills in the option word according to information that the user provides when editing a data set. "Writing an exit for syntax checkers" on page 263 describes how to write an exit routine.

Associating the syntax checker with a data set type

After you write and install your syntax checker, you associate the syntax checker with a data set type. You can associate a syntax checker with one or several data set types.

To associate the syntax checker with a data set type, use the IKJEDIT macro instruction and submit a System Modification Program Extended (SMP/E) job to perform the association. Using IKJEDIT, specify the name of the data set type and the name of the syntax checker. As an example, suppose you defined a data set type named EXTT and a syntax checker named SYN1. You would code IKJEDIT as follows; for information about the syntax of IKJEDIT, see "IKJEDIT macro" on page 709.

```
EDIT IKJEDIT DSTYPE=(EXTT),CHECKER=(SYN1)
```

IKJEDIT generates the following two CSECTs; both CSECTs are required.

- IKJEBEPD, which is link-edited with the IKJEBEPS module in SYS1.CMDLIB
- IKJEBINS, which is link-edited with the EDIT command in SYS1.CMDLIB

TSO/E provides a sample job in SYS1.SAMPLIB member IKJTSMPE that shows how to use IKJEDIT. Instructions for using the sample job are included in the SYS1.SAMPLIB member.

Writing an exit for syntax checkers

TSO/E provides syntax checkers for several data set types that it provides. For these data set types, the associated syntax checker determines the attributes of the data set type by referring to information that EDIT initialization sets in the option word. The option word is a fullword in the parameter list of the syntax checker. For more information about the parameter list and option word, see "Record format and interface for syntax checking" on page 259.

You can write your own syntax checkers. However, for installation- written syntax checkers, EDIT initialization does not place attribute information about the data set type in the option word. You can, however, write an exit for your syntax checker that fills in the option word for your syntax checker with information the user specifies on the EDIT command.

TSO/E supplied exit

TSO/E does not supply a default exit routine for installation-written syntax checkers.

Entry specifications

The contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of a three-word parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions

The following describes the three-word parameter list.

Word 1
Address of the subfield parameter descriptor element (PDE), which is created by the parse service routine. For information about the format and content of the PDE, see the description of the IKJIDENT PDE in *z/OS TSO/E Programming Services*.

Word 2
Address of bytes 0 and 1 of the syntax checker option word. The syntax checker you provide can assign its own meanings for the bit settings.

Word 3
Address of the command processor parameter list (CPPL) that the TMP

passes to the EDIT command processor. For information about the format and content of the CPPL, see the description of the CPPL in *z/OS TSO/E Programming Services*.

Return specifications

The contents of the registers on return must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes

Return code (decimal)	Description
0	Processing was successful
4	Processing was unsuccessful

Programming considerations

The exit must follow standard linkage conventions. It must save the registers on entry and restore the registers when it returns.

The exit can access the ECT and UPT to invoke the parse service routine or the TSO/E service routines. For information about the service routines, see *z/OS TSO/E Programming Services*.

Use the exit to fill in the option word for your syntax checker according to information the user enters. When users issue the EDIT command and specify your installation's data set type keyword, they can also specify a subfield. This subfield can contain any valid alphanumeric data defined. The data cannot exceed 256 characters and cannot contain any blanks, tabulation characters, or commas. The information is passed to the exit, which interprets it and encodes it into bytes 0 and 1 of the option word. For information about the standard interface between EDIT and syntax checkers and the option word, see “Record format and interface for syntax checking” on page 259.

Environment

- State: Problem program
- Key: 8
- AMODE(24), RMODE(24)

Installing the exit

There are no required naming conventions for the exit. After you write and install the exit, you must associate the name of the exit with its corresponding syntax checker and data set type. To associate the exit name with a syntax checker and data set type, use the IKJEDIT macro instruction and submit a System Modification Program Extended (SMP/E) job to perform the association. Using IKJEDIT, specify the names of the data set type, syntax checker, and exit. For more information about the syntax of the IKJEDIT macro, see “IKJEDIT macro” on page 709.

As an example, suppose you define your own data set type, associated syntax checker, and corresponding exit as follows:

- Name of data set type — TYPEX

- Syntax checker name — SYNTAX1
- Exit name for syntax checker — SYN1EXIT

On IKJEDIT, include the following operands:
- DSTYPE(TYPEX)
- CHECKER(SYNTAX1)
- USEREXT(SYN1EXIT)

IKJEDIT generates the following two CSECTs:
- IKJEBEPD, which is link-edited with the IKJEBEPS module in SYS1.CMDLIB
- IKJEBINS, which is link-edited with the EDIT command in SYS1.CMDLIB

Both CSECTs are required.

TSO/E provides a sample job in SYS1.SAMPLIB member IKJTSMPE that shows how to use IKJEDIT. Instructions for using the sample job are included in the SYS1.SAMPLIB member.

Writing an exit for the RENUM, MOVE, and COPY subcommands

Whenever a terminal user issues the RENUM, MOVE, or COPY subcommand of the EDIT command, the line numbering of the records in the data set is affected.

RENUM
Assigns a line number to each record in data sets that do not already have line numbers. RENUM also renumbers records in data sets that have line numbers.

MOVE
Moves records in a data set from one location to another, and renumbers the lines as needed.

COPY Copies records in a data set, and renumbers the lines as needed.

For more information about the subcommands and how line numbering is affected, see *z/OS TSO/E Command Reference*.

You can use an exit to tailor the way line numbering is done. The exit is common to all three subcommands, RENUM, MOVE, and COPY. The exit receives control whenever a terminal user issues RENUM, MOVE, or COPY when editing a data set.

By default, the exit processes only VSBASIC data sets. To have the exit process other data set types, use the IKJEDIT macro instruction. For information on IKJEDIT, see “Installing the exit” on page 268.

The following highlights some ways you can use the RENUM, MOVE, and COPY exit. For more information, see “Possible uses” on page 270.
- Change line references that are imbedded within a statement
- Flag specific statements
- Change default line numbering

TSO/E-supplied exit

TSO/E does not provide a default exit routine for the RENUM, MOVE, and COPY subcommands.

Entry specifications

The contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of a parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

The exit receives the data set in storage as input. The data set contains all of the records in the current EDIT utility data set. The RECFM of the EDIT input data set indicates the format of the data set. Standard header words (LL/00) give the lengths of variable records. The data attribute parameters give the length of fixed records. All records in the data set are contiguous and are contained in a single area of virtual storage, assigned from subpool 1.

Parameter descriptions

On entry, register 1 contains the address of the following parameter list:

Offset (hex)	Length	Contents or meaning	
0	4	UPT address	
4	4	ECT address	
8	4	EDIT attention ECB address	
C	1	Flags:	
		Bit 0	0 - Records have standard line numbers
			1 - Records do not have standard line numbers
		Bit 1	0 - Not RENUM function
			1 - RENUM function
		Bit 2	0 - Not MOVE function
			1 - MOVE function
		Bit 3	0 - Not COPY function
			1 - COPY function
		Bit 4	0 - Normal line range specification
			1 - '* COUNT' range notation
		Bit 5	0 - Fixed-length records
			1 - Variable-length records
D	3	Address of data set	
10	4	Address of subcommand interface area. The two formats of the subcommand interface area are described below.	
14	4	Address of data attribute parameters. The data attribute parameters are described below.	

Subcommand interface area

The subcommand interface area has two different formats; one for RENUM and one for MOVE/COPY. The two formats are described below.

Table 43. RENUM format of subcommand interface area

Offset (hex)	Length	Contents or meaning
0	4	Line number position
4	4	Length of line number
8	4	First line to be renumbered (0 indicates first line of data set)
C	4	Last line to be renumbered (X'7FFFFFFF' requests renumbering through end of data set)
10	4	Line number to be assigned to the first renumbered line
14	4	Increment to be used
18	4	Address of current line reference (the exit must update this before it returns control to RENUM)

Table 44. MOVE/COPY format of subcommand interface area

Offset (hex)	Length	Contents or meaning
0	4	Line number position
4	4	Length of line number
8	4	First line of MOVE/COPY range (0 indicates first line in data set)
C	4	Last line of MOVE/COPY range (Bit 4 of flags = 1)
10	4	Line number of line preceding insertion point
14	4	Increment to be used
18	4	Current line pointer

Data attribute parameters

The data attribute parameters consist of a two-word list:

Offset (hex)	Length	Contents or meaning
0	4	Logical record length (in bytes)
4	4	Length of the data set (in bytes)

Return specifications

The contents of the registers on return must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes

Return code (decimal)	Description
0	Indicates successful completion or attention interruption
4	Requests RENUM processing as if the exit were unavailable
8	Indicates function not performed; message sent to the terminal user.

Programming considerations

The exit must follow standard linkage conventions. It must save the registers on entry and restore the registers when it returns. The exit must be reentrant, reusable, and refreshable.

The exit can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

The subcommand processor that calls the exit does not pass a work area. Therefore, if the exit needs storage, it must obtain the storage from subpool 1.

The exit must check periodically the EDIT command's attention ECB for an attention interruption. When an interrupt occurs, the exit must release any resources that it has obtained, and return control to the subcommand processor with a return code of zero.

When the exit returns control to the subcommand processor, it must return the address of the updated data set and the length of the data set, if applicable. The exit must also update the current line pointer in the subcommand interface area and release any resources it obtained.

Environment

- State: Problem program
- Key: 8
- AMODE(24), RMODE(24)

Installing the exit

There are no required naming conventions for the exit. The exit can reside in a step library, in the LPA library, or in any data set in LNKLSTxx.

After you write and install the exit, you must specify the data set types that you want the exit to process. By default, the exit processes only VSBASIC data sets. To have the exit process other data set types, use the IKJEDIT macro instruction and submit a System Modification Program Extended (SMP/E) job. Using IKJEDIT, specify the data set types you want the exit to process. For more information about the syntax of the IKJEDIT macro, see "IKJEDIT macro" on page 709.

TSO/E provides a sample job in SYS1.SAMPLIB member IKJTSMPE that shows how to use IKJEDIT. Instructions for using the sample job are included in the SYS1.SAMPLIB member.

As an example, suppose the name of your exit is NUMEXT and you want it to process FORT and COBOL data set types in addition to VSBASIC data sets. Figure 43 on page 270 shows a sample SMP/E job and how you specify the exit name and corresponding data set types on IKJEDIT. In the job, you:

- Update the job statement
- Specify the name of the SMP/E procedure on the EXEC statement
- Update SET BDY with your target name
- Optionally, change the USERMOD number on the ++USERMOD, RECEIVE, APPLY, and ACCEPT control statements
- Supply any DD statements that you need
- Update the DSTYPE and DATEXIT operands of IKJEDIT. Specify the data set types (FORT and COBOL) on the DSTYPE operand. Specify the name of your exit on the DATEXIT operand.

You must update IKJEDIT in two places in the job in order for the job to generate two CSECTs. Both CSECTs, IKJEBEPD and IKJEBINS, are required.

Note: You must place the current FMID for TSO/E in the FMID operand of the ++VER statement and the current RMID of IKJEBEPD in the RMID operand in the ++VER statement before installing this usermod. To find these values on your system, you can use SMP/E panels to do a CSI query against the target zone for SRC IKJEBEPD, or run an SMP/E job to list it using this SMP/E control statement.

```
SET BDY(target zone)
LIST SRC IKJEBEPD
```

```
//USERMOD   JOB   MSGLEVEL=(1,1)
//STEP1     EXEC  <SMP/E-PROCEDURE-NAME>
//*
//*   SMP/E DD STATEMENTS
//*
//*SMPPTS   DD DSN=SYS1.SMPPTS,DISP=SHR
//*SMPCSI   DD DSN=SMPE.SMPCSI.CSI,DISP=SHR
//*SMPSCDS  DD DSN=SYS1.SMPCDS,DISP=SHR
//*
//*  DLIB DD STATEMENTS
//*
//*ACMDLIB   DD    DSN=SYS1.ACMDLIB,DISP=SHR
//*ASAMPLIB  DD    DSN=SYS1.ASAMPLIB,DISP=SHR
//*
//*  LIBRARY DD STATEMENTS
//*
//*LINKLIB   DD    DSN=SYS1.LINKLIB,DISP=SHR
//*
//SMPCNTL DD *
  SET BDY(GLOBAL) . <== CHANGE TO YOUR TARGET NAME ============
  RECEIVE S(UM99999) .
  APPLY S(TARGET) .
  ACCEPT S(DLIB) USERMODS .
/*
//SMPPTFIN DD *
++USERMOD (UM99999) .
++VER (Z038) FMID (Current TSO/E FMID) RMID(RMID of IKJEBEPD) .
++SRC(IKJEBEPD) DISTLIB(ASAMPLIB) DISTOBJ(ACMDLIB) .
         IKJEDIT MODULE=IKJEBEPD,                                  X
               DSTYPE=(FORT,COBOL),                                X
               BLOCK=,                                             X
               FORMAT=,                                            X
               FIXED=,                                             X
               VAR=,                                               X
               CONVERT=,                                           X
               CHECKER=,                                           X
               PRMPTR=,                                            X
               USERSRC=,                                           X
               DATEXIT=NUMEXT
         END
++SRC(IKJEBINS) DISTLIB(ASAMPLIB) DISTOBJ(ACMDLIB) .
         IKJEDIT MODULE=IKJEBINS,                                  X
               DSTYPE=(FORT,COBOL),                                X
               BLOCK=,                                             X
               FORMAT=,                                            X
               FIXED=,                                             X
               VAR=,                                               X
               CONVERT=,                                           X
               CHECKER=,                                           X
               PRMPTR=,                                            X
               USERSRC=,                                           X
               DATEXIT=NUMEXT
         END
/*
```

Figure 43. Example of specifying the data set types that a RENUM, MOVE, COPY exit processes

Possible uses

- A user may use a line number within a statement to pass control to another part of the data set. For example, a user may pass control to line number 200 using the statement `GOTO 200`. The RENUM, MOVE, and COPY subcommand processors do not handle line references that are within a statement; that is, they do not change the GOTO statement. If the line numbering is changed and line 200 becomes line 250, the GOTO statement will pass control to an incorrect statement (line 200 instead of line 250). You can write an exit that adjusts the line references within statements, as needed, whenever a terminal user issues RENUM, MOVE, or COPY.

- Your exit can flag a statement based on some condition. One way of flagging a particular statement is to add a comment at the end of the statement.
- RENUM, MOVE, and COPY renumber the line numbers in a data set based on either a default increment value or a value that the user specifies on the subcommand. You can use the exit to establish a different numbering scheme. For a description of how each of the subcommands handles line numbering, see *z/OS TSO/E Command Reference*.

Adding EDIT subcommands

The EDIT command supplies many subcommands for various editing functions. If you need additional editing functions, you can write your own subcommand processors. You must first write the subcommand processor itself, and then add the name of your subcommand to the system.

Writing a subcommand of EDIT

The steps for writing a subcommand processor are listed in *z/OS TSO/E Programming Guide*. The steps for writing an EDIT subcommand are the same as the steps for writing a subcommand processor, except in the first and last steps: accessing parameters and passing return codes.

Accessing parameters

Unlike other commands or subcommands, EDIT subcommands do not directly access the command processor parameter list (CPPL). Instead, EDIT subcommands begin by accessing the EDIT command processor communication area (CA), which is mapped by macro IKJEBECA.

When an EDIT subcommand receives control, register 1 points to the CA. In the CA, the significant field for EDIT subcommands is the CAPTTMP field, which points to the CPPL that was passed as input to the EDIT command processor. For more information about the CA, refer to *z/OS TSO/E System Diagnosis: Data Areas*.

Return codes from EDIT subcommands

An EDIT subcommand can return control to EDIT with one of the following decimal return codes set in register 15. EDIT will then perform the indicated actions.

Table 45. Return codes for EDIT subcommands

Return code (decimal)	Description
0	EDIT will process the next subcommand.
4	EDIT will enter input mode.
8	EDIT will clear any waiting or stacked input and then process the next subcommand.
12	EDIT will terminate.

Defining a subcommand to EDIT

TSO/E stores the names of the EDIT subcommand processors that you write in a table called IKJEBMA9. IKJEBMA9 is a CSECT within load module IKJEBEMA. When a user enters an EDIT subcommand, IKJEBMA invokes the appropriate subcommand processor.

After you write an EDIT subcommand processor, you must add the name to the IKJEBMA9 table. You use the IKJEBEST macro instruction to add the name. You must ensure that the name of your subcommand processor is not the same as the name of any subcommand processor that TSO/E provides. For a list of the subcommand processors TSO/E provides, see *z/OS TSO/E Command Reference*.

Using IKJEBEST, specify the following operands:
- Name of the subcommand
- Abbreviation for the subcommand name
- Name of the module that processes the subcommand
- CSECT=USER

The abbreviation for the subcommand name is optional. The three other operands are required.

You can add one or more subcommand processor names on one invocation of the IKJEBEST macro. The format of the macro is:

```
IKJEBEST(subcmd,abbr,mod-name)[,...],CSECT=USER
```

When you assemble the IKJEBEST macro instruction, the CSECT=USER operand causes the resulting object module to be named IKJEBMA9. You must link-edit the IKJEBMA9 module with the IKJEBEMA load module, performing a CSECT-replacement operation for IKJEBMA9 object module. The EDIT subcommands that you added are then available for use.

You can obtain the IKJEBEST macro from SYS1.MABLIB. You can also include IKJEBEST in your macro library using the code shown in Figure 44 on page 273.

The following examples illustrate how to use IKJEBEST.

Example 1: To add the EDIT subcommand processor named SWTCH, code IKJEBEST as follows:

```
IKJEBEST (SWTCH,SW,XXXSWTCH),CSECT=USER
```

The abbreviation for the SWTCH subcommand is SW and the name of the module that processes the subcommand is XXXSWTCH.

Example 2: To add the following subcommand processor names, code IKJEBEST as follows:

```
IKJEBEST (SWTCH,SW,XXXSWTCH),(SRTRN,SRT,XXXSRTRN),CSECT=USER
```

Name	Abbreviation	Module name
SWTCH	SW	XXXSWTCH
SRTRN	SRT	XXXSRTRN

```
         MACRO
         IKJEBEST     &CSECT=IBM;
         LCLA     &A,&B,&C,&D,&E
         LCLA     &F
         LCLC     &CNAME,&SCNAME,&ABBR,&LDMOD,&LABEL,&LABEL1,&LABEL2,&NMBR;
         AIF      ('&CSECT' NE 'IBM').CONT0
&CNAME,    SETC     'IKJEBMA8'      DEFINE CSECT NAME FOR IBM TABLE.
IKJEBMA8  CSECT
         ENTRY    MA8IP002
         ENTRY    MA8LI002
         AGO      .CONT1
.CONT0    ANOP
         AIF      ('&CSECT' NE 'USER').ERROR2
&CNAME    SETC     'IKJEBMA9'       DEFINE CSECT NAME FOR USER TABLE.
IKJEBMA9  CSECT
.CONT1    ANOP
&A        SETA     N'&SYSLIST
         AIF      (&A EQ 0).END
&B        SETA     1
&F        SETA     1
.CONT2    ANOP
&C        SETA     N'&SYSLIST(&B)
         AIF      (&C LT 2 OR &C GT 3).ERROR1
&E        SETA     K'&SYSLIST(&B,&C)
&D        SETA     &E-1

.*  THE FOLLOWING FLAGGED INSTRUCTIONS WERE ADDED TO PROVIDE UNIQUE
.*  LABELS, EVEN IF MODULES HAVE IDENTICAL LAST TWO CHARACTERS IN
.*  ENTRY POINT NAMES.  THE LABELS FOR MODULES IKJEBELI AND IKJEBEIP
.*  ARE UNCHANGED. SINCE THEY ARE REFERENCED WITHIN IKJEBEMA.
         AIF       ('&CSECT' NE 'IBM').CONT10
         AIF       ('&SYSLIST(&B,&C)'(&D,&E) EQ 'LI'OR                X
                   '&SYSLIST(&B,&C)'(&D,&E) EQ 'IP' ).CONT11
.CONT10   ANOP
&LABEL1   SETC      '&CNAME'(6,8).'@'.'&F'
&F        SETA      &F+1
&LABEL2   SETC      '&CNAME'(6,8).'@'.'&F'
&F        SETA      &F+1
         AGO       .CONT12
.CONT11   ANOP
&LABEL1   SETC      '&CNAME'(6,8).'&SYSLIST(&B,&C)'(&D,&E).'001'
&LABEL2   SETC      '&CNAME'(6,8).'&SYSLIST(&B,&C)'(&D,&E).'002'
.CONT12   ANOP
&SCNAME   SETC      '&SYSLIST(&B,1)'
         SPACE 2
         DC        AL1(&LABEL1-*-1) LENGTH OF SUBCOMMAND NAME.
         DC        C'&SCNAME' SUBCOMMAND NAME.
&LABEL1   EQU       *
         DC        AL1(&LABEL2-*-1) LENGTH OF ABBREVIATION.
         AIF       (K'&SYSLIST(&B,2) EQ 0).CONT5
&ABBR     SETC      '&SYSLIST(&B,2)'
         DC        C'&ABBR' ABBREVIATION FOR SUBCOMMAND.
.CONT5    ANOP
&LABEL2   EQU       *
&LDMOD    SETC      '&SYSLIST(&B,&C)'
         DC        CL8'&LDMOD' LOAD MODULE NAME.
         AIF       (&B EQ &A).END
&B        SETA      &B+1
         AGO       .CONT2
.END      ANOP
         SPACE     2
         DC        AL1(255)    END OF TABLE MARKER.
         MEXIT
.ERROR1   MNOTE     12,'INVALID TABLE ENTRY'
         MEXIT
.ERROR2   MNOTE     12,'INVALID KEYWORD VALUE'
         MEND
```

Figure 44. Example of Including IKJEBEST in your macro library

Allocating space for the utility work data sets

EDIT uses permanent or temporary utility data sets to record user input and perform certain other functions. These other functions include, but are not limited to, the RENUM subcommand and some variations of the MOVE or COPY subcommands.

EDIT uses temporary data sets if:
- The user's profile specifies NORECOVER, or
- The user is invoking EDIT in the background

EDIT uses permanent utility work data sets when the user's profile specifies RECOVER and the user is not invoking the EDIT command in the background

You have two options for space allocation for both temporary and permanent utility work data sets:

1. Default space allocation

   EDIT uses its own algorithms to calculate the default space allocation. "Default space allocation" describes the algorithms.
2. Controlled space allocation

   You can preallocate the data sets to control the space allocation and direct access space for the data sets. "Controlled space allocation" on page 275 describes controlled space allocation.

Default space allocation

EDIT uses different algorithms to calculate the default space allocation for utility work data sets. The algorithm used depends on whether the user is editing a new data set or an existing data set.

Note: Regardless of the physical space allocated, the maximum number of records that the edit access method can handle depends on the record length. This maximum applies to both new and existing data sets. For example:
- For 80-byte records, the maximum number of records EDIT can handle is approximately 390,000.
- For 255-byte records, the maximum number of records EDIT can handle is approximately 130,000.

Editing a new data set

EDIT allocates a utility work data set of four blocks with a default block size of:
- 4096 bytes for primary space (16384 bytes)
- One half that amount for secondary space (8192 bytes).

If the data set being edited contains 80 character records, the utility data set has a total capacity of approximately 300 records.

Editing an existing data set

A utility work data set is allocated as follows:
- Primary space, in 4K blocks, equals:

```
2 ((((x + y)b)/4096)+2)
```

x = the number of records in the existing data set

y = the additional number of records to be added to the data set. This number is variable and depends on the user specifications on a MOVE or COPY subcommand. The value may be 0.

b = the number of bytes (characters) per record

- Secondary space equals:

```
Primary space/2
```

Example of default space allocation

The following illustrates an example of default space allocation. Suppose an existing data set has the following characteristics:

- Contains 6000 records
- 200 additional records are to be copied into the data set
- Each record contains 120 characters

The primary space calculation (number of 4K blocks) is:

```
2 ((((6000 + 200) 120)/4096) +2) = 366
```

The secondary space calculation (number of 4K blocks) is:

```
366/2 = 183
```

The utility work data set has a capacity of approximately 18,600 records.

Controlled space allocation

If you want to control the allocation of the utility work data sets and the direct access space used for the data sets, you can preallocate the data sets. You can preallocate both temporary and permanent utility data sets.

Note: You may also need to preallocate data sets when using VIO due to the compounding effect of TSO/E and VIO allocation algorithms.

Temporary utility work data sets

To preallocate temporary data sets, include two DD statements that define the data sets &EDIT and &EDIT2 in the user's logon procedure. For example:

```
//SYSEDIT   DD   DSN=&EDIT,UNIT=SYSDA,SPACE=(2048,(20,10))

//SYSEDIT2  DD   DSN=&EDIT2,UNIT=SYSDA,SPACE=(6144,(50,20))
```

SYSEDIT is a sample ddname for the temporary utility work data set &EDIT, which is used whenever the user edits a data set. SYSEDIT2 is a sample ddname for the temporary utility work data set &EDIT2, which is used when the user performs certain functions. These functions include, but are not limited to, the RENUM subcommand and some variations of the MOVE or COPY subcommands. EDIT may use one or both of these data sets, depending upon the functions invoked by the user and the timing of events which affect the user's edit session.

When the EDIT command is executed in the background, DD statements defining both temporary work data sets must be included in the job step that calls the EDIT command.

Permanent utility work data sets

To preallocate permanent data sets, name the two data sets as:

- DSN=userid.EDITUTL1
- DSN=userid.EDITUTL2

Catalog the data sets prior to the edit session. You do not need DD statements in the user's logon procedure because the catalog is searched for their location. The first edit session, which occurs after you catalog the data sets, will initially use the data set userid.EDITUTL1. EDIT may also use userid.EDITUTL2 during this initial session if the user invokes certain subcommands such as RENUM. In later edit sessions EDIT may use one or both of these data sets, depending upon the functions invoked by the user and the timing of events which affect the user's edit session.

Chapter 31. Customizing the EXEC command

This section describes ways to customize the EXEC command.

Writing exits for the EXEC command

TSO/E users issue the EXEC command to execute REXX execs or CLISTs. The EXEC command receives control when the user specifies it in one of the following forms:

- **Explicit form** - EXEC or EX is followed by the name of the data set containing the REXX exec or CLIST.
- **Implicit form** - The name of a member of a REXX exec or CLIST library is entered *without* the command name, EXEC or EX. A library is a partitioned data set (PDS) that must be allocated to the specific REXX exec or CLIST (SYSEXEC or SYSPROC) either dynamically by the ALLOCATE command or the ALTLIB command, or as part of the logon procedure.
- **Extended implicit form** - A percent sign (%) precedes the name of a member of a REXX exec or CLIST library.

Processing for a CLIST is done in two steps: phase 1, which is performed by the EXEC command processor, and phase 2. In phase 1, the EXEC command processor reads the CLIST records from the input data set and builds an in-storage command procedure. EXEC then places the command procedure that it built on the input stack. This stack is maintained by TSO/E to determine the source of input. Phase 2 processing receives control as each record in the command procedure is removed from the stack.

TSO/E provides initialization and termination exits that you can use to customize the EXEC command. These exits receive control during EXEC command processing, as follows:

- If the explicit form is used to invoke EXEC, the initialization exit receives control before the EXEC command processor invokes the parse service routine to syntax check the input parameters.
- If the implicit form is used to invoke EXEC, the initialization exit receives control before the EXEC command processor invokes the command scan service routine to syntax check the input parameters.
- The termination exit receives control just before the EXEC command terminates processing.

TSO/E users can invoke the EXEC command in the TEST or EDIT environment to perform the same basic functions as the EXEC command. Exits that you write to customize the EXEC command are also in effect when the EXEC command is executed in the TEST or EDIT environment.

The following paragraphs highlight some ways you can use the initialization and termination exits. For more information about how you can use the exits, see "Possible uses" on page 281.

Some ways you can use the initialization exit include:
- Providing default values for symbolic variables that are used in CLISTs
- Changing the default values for operands of the EXEC command

- Invoking alternate interpreters

Some ways you can use the termination exit include:
- Performing clean-up processing
- Changing the default CONTROL options for CLISTs

TSO/E also provides several other exits that you can use to customize CLIST processing:
- The CLIST built-in function exit allows you to add your own built-in functions to the CLIST language. This exit is described in “Writing an exit for installation-written built-in functions (IKJCT44B)” on page 495.
- The CLIST statement exit allows you to add your own statements to the CLIST language. This exit is described in “Writing an exit for installation-written statements (IKJCT44S)” on page 499.

TSO/E-supplied exits

TSO/E does not provide default exit routines for the EXEC command.

Entry specifications

The contents of the registers on entry for the initialization and termination exits for the EXEC command are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions for the initialization exit

The initialization exit receives the standard exit parameter list. However, no exit-dependent data is passed to the initialization exit. For information about the standard exit parameter list and the parameter entry keys, see “TSO/E standard exit parameter list” on page 32.

Parameter descriptions for the termination exit

The termination exit receives the standard exit parameter list. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32. If you provide an initialization exit, the termination exit is passed the same parameter entries for the new command buffer and exit-to-exit communication word that were passed to the initialization exit.

Figure 45 on page 279 shows the exit-dependent data that the termination exit receives beginning at offset +36 (decimal) in the parameter list. The parameter entry is described following the figure.

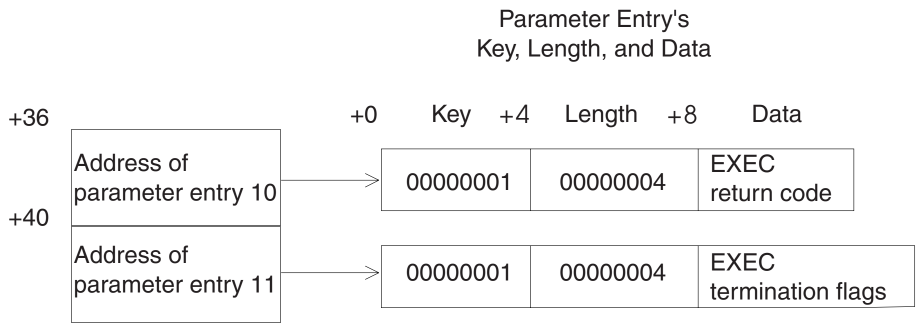


Figure 45. Exit-Dependent data for the EXEC command termination exit

EXEC Return Code (Parameter Entry 10)
This parameter entry is the return code from the EXEC command processor (either CLIST phase 1 processing or the completion of REXX processing). For information on the return codes from the EXEC command, see *z/OS TSO/E Command Reference*.

EXEC Termination Flags (Parameter Entry 11)
This parameter entry is a fullword indicating whether the EXEC command processed a CLIST, a REXX exec, or neither. Possible values for the EXEC termination flags are as follows:

Value (Hex)	Description
X'00000000'	The EXEC command did not process either a CLIST or a REXX exec. Possible reasons are: • EXEC processing encountered an error. For example, the EXEC command did not find the CLIST or REXX exec to be executed. • The EXEC initialization exit indicated that EXEC processing is to be bypassed.
X'00000001'	The EXEC command processed CLIST phase 1.
X'00000002'	The EXEC command processed a REXX exec.

Return specifications

The contents of the registers on return from the initialization and termination exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the initialization and termination exits

Table 46 on page 280 shows the return codes that the initialization and termination exits support.

Table 46. Return codes for the EXEC initialization and termination exits

Return code (decimal)	Description
0	Exit processing was successful. EXEC processing continues.
12, 16	An error occurred in the exit. The EXEC command processor terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the EXEC command processor. For more information, see the notes following this table and “Exit reason code” on page 38. If your exit sets a return code of either 12 or 16, you should consider displaying an informational message to the user. You can use the PUTLINE service routine to issue an informational message. See *z/OS TSO/E Programming Services* for more information.

Note:
1. If an exit returns an undefined return code, the EXEC command processor terminates without issuing an error message to the user.
2. If an initialization or termination exit sets a reason code that has a key value of X'03', this reason code is used as the return code from the EXEC command. However, if both exits indicate that the reason code is to be used as the return code from the EXEC command, the reason code from the termination exit overrides that from the initialization exit.
3. When requesting that the exit reason code be used as the return code from EXEC, you must insure that the reason code does not duplicate existing EXEC return codes.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exits must be reentrant, refreshable, and reusable.

If the processing done in the initialization exit requires clean-up to be performed, you must write a termination exit. For example, if the initialization exit obtains storage to return a new command buffer to the EXEC command processor, you must provide a termination exit to free this storage.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

Environment

- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY)

Installing the exits

You must name the exits as follows:

Initialization
IKJCT43I

Termination
IKJCT43T

Link-edit each exit as a separate load module. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see “Installing the standard-format exits” on page 39.

Possible uses

Some ways you can use the initialization exit include:
- Providing default values for symbolic variables that are used in CLISTs
- Changing the default values for operands of the EXEC command. For example, you can cause the PROMPT option to be the default for the explicit form of EXEC and therefore propagate the PROMPT operand to nested EXEC commands.
- Invoking an alternate interpreter. You can allow TSO/E users to issue the EXEC command to execute programs written in interpretive languages other than the CLIST language. To invoke an alternate interpreter, follow these steps:
  1. Examine the command buffer to determine the type of interpretive language.
  2. Invoke the interpreter.
  3. Set an exit return code of 16 to cause the EXEC command processor to terminate.

Some ways you can use the termination exit include:
- Performing clean-up processing. For example, you can free storage that was obtained in the initialization exit.
- Changing the default CONTROL options for CLISTs. These options are used on the CLIST CONTROL statement to define processing options. Follow these steps to change the default CONTROL options:
  1. Check the return code from the EXEC command processor to determine whether the CLIST has been placed on the stack and is executable. This return code is passed to the termination exit in the parameter list described in “Parameter descriptions for the termination exit” on page 278.
  2. Use the CLIST variable access routine (IKJCT441) to change the CONTROL options by setting symbolic variables. Change an option by setting the corresponding symbolic variable to the string `ON` or `OFF`. For example, to cause CONTROL NOPROMPT to be in effect, set &SYSPROMPT to `OFF`. Table 47 shows the CONTROL options and the corresponding variables that you can set:

Table 47. Symbolic variables used to set CLIST CONTROL options

CONTROL option	Symbolic variable
PROMPT/NOPROMPT	&SYSPROMPT
SYMLIST/NOSYMLIST	&SYSSYMLIST
CONLIST/NOCONLIST	&SYSCONLIST
LIST/NOLIST	&SYSLIST
CAPS/NOCAPS/ASIS	&SYSASIS
MSG/NOMSG	&SYSMSG
FLUSH/NOFLUSH	&SYSFLUSH

For information about using IKJCT441 to change CLIST variables, see *z/OS TSO/E Programming Services*. For more information about CLIST CONTROL options, see *z/OS TSO/E CLISTs*.

Chapter 32. Customizing the FREE command

This section describes ways to customize the FREE command.

Writing exits for the FREE command

Users issue the FREE command to:
- Release (deallocate) previously allocated data sets
- Change the output class of system output (SYSOUT) data sets
- Delete attribute lists
- Delete output descriptors that were defined by the OUTDES command
- Change a data set disposition that was specified with the ALLOCATE command.

For information about using FREE, see *z/OS TSO/E User's Guide*. For information about FREE and its operands, see *z/OS TSO/E Command Reference*.

TSO/E provides an initialization exit and a termination exit for the FREE command. You can use the exits to customize FREE processing for your users. The initialization exit receives control before the FREE command processor invokes the parse service routine to parse the command. The termination exit receives control just before the FREE command processor completes processing. If the initialization exit returns successfully to the FREE command processor and FREE processing itself abends, the FREE command processor invokes the termination exit before it terminates.

You can use the initialization exit to change the operands that users specify on the command or correct user errors when they issue the command. You can use the termination exit to perform clean-up or special processing prior to FREE completion. Depending on the processing your initialization exit performs, you may not need a corresponding termination exit.

The following highlights some ways you can use the FREE exits. For more information about how you can use the exits, see "Possible uses" on page 286.
- Correct a user's errors on the FREE command
- Change the operands a user specifies on the command
- Provide *pseudo-operands* that are equivalent to two or more FREE operands

TSO/E-supplied exits

TSO/E does not provide default exit routines for the FREE exits.

Entry specifications

For the FREE exits, the contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions for the initialization exit

The FREE initialization exit receives the address of the standard exit parameter list. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32. The initialization exit does not receive any exit-dependent data.

Parameter descriptions for the termination exit

The FREE termination exit receives the address of the standard exit parameter list. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32. The termination exit does not receive any exit-dependent data.

If the initialization exit returns a new command buffer or an exit-to-exit communication word to the FREE command processor, FREE passes the values of these parameter entries to the termination exit. For more information about the parameter entries, see “TSO/E standard exit parameter list” on page 32.

Return specifications

The contents of the registers on return for both FREE exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the initialization and termination exits

Table 48 shows the return codes that the FREE initialization and termination exits support.

Table 48. Return codes for the FREE initialization and termination exits

Return code (decimal)	Description
0	Exit processing was successful. FREE processing continues.
12	Exit processing was unsuccessful. FREE issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the FREE command processor. For more information about the exit reason code, see “Exit reason code” on page 38.
16	Exit processing was unsuccessful. FREE terminates processing. The FREE command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the FREE command processor terminates without displaying a message to the user.

The termination exit receives control just before the completion of FREE processing. Therefore, the FREE command processor may have already successfully executed regardless of the return code the termination exit returns.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exits must be reentrant, refreshable, reusable, and not APF-authorized.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

In some cases, you may not need to write a termination exit. This depends on:

- Whether the processing that the initialization exit performs requires a termination exit to perform clean-up activities
- How you use the exits to customize FREE processing

If the initialization exit obtains a system resource, you must write a termination exit to free the resource. For example, the initialization exit may obtain storage to return a new command buffer to the FREE command processor. In this case, you must provide a termination exit to free the storage for the new command buffer.

The initialization exit can change FREE operands using the command buffer. The exit checks the command buffer it receives and determines whether to change any operands. To change the operands, the exit must:

- Obtain storage for a new command buffer
- Build the new command buffer
- Update the key, length, and data fields for the new command buffer (parameter entry 2)
- Set return code 0 and return control to FREE

For more information about the parameter entries, see “TSO/E standard exit parameter list” on page 32. For more information about the format of the command buffer, see “Command buffer” on page 35.

Environment

- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY)

Installing the exits

You must name the exits as follows:

Initialization
IKJEFD21

Termination
IKJEFD22

Link-edit each exit as a separate load module. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in:

- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see "Installing the standard-format exits" on page 39.

Possible uses

Some possible uses of the FREE exits are described below:

- Change the operands that the user specifies on the command

  You can use the initialization exit to change the operands that users specify on the FREE command. The initialization exit receives the address of the command buffer. It can change the operands the user specifies on the FREE command by using a new command buffer. For example, the initialization exit can scan the command buffer and correct any errors on the command.
- Provide installation-defined pseudo-operands

  You can provide *pseudo-operands* for your installation's users that are equivalent to two or more FREE operands. Providing pseudo-operands makes it easier for users to issue the FREE command. Users need not remember several FREE operands. They can specify the pseudo-operand.

  For example, you could associate a pseudo-operand named FREEIT with three FREE operands. The initialization exit can scan the command buffer. If the exit finds the pseudo-operand FREEIT, it can replace FREEIT with the actual FREE operands and return a new command buffer.

To check the command buffer and change its contents, the initialization exit can:

- Scan the command buffer and decide, based on your own criteria, to change the command the user issued
- Obtain storage for a new command buffer
- Build the new command buffer
- Update the key, length, and data fields for the new command buffer as follows:

  Key X'02'

  Length
  the length of the new command buffer

  Data the address of the new command buffer
- Set a return code of 0 and return control to the FREE command processor.

The exit must not change the command buffer it receives. It must create a new command buffer and return the address of the new command buffer to FREE. For more information about the command buffer and the new command buffer, see "TSO/E standard exit parameter list" on page 32. For information about the format of the command buffer, see "Command buffer" on page 35.

You must also write a termination exit to free the storage the initialization exit obtains for the new command buffer. When the FREE command processor invokes the termination exit, it passes the address of the new command buffer to the termination exit. The termination exit frees the storage for the new command buffer.

Chapter 33. Customizing the PARMLIB command

The PARMLIB command lets you list and update TSO/E specifications that are in effect on the system. Those TSO/E specifications include tables of authorized commands and programs, and default values for some TSO/E commands.

The CHECK function of the PARMLIB command lets you check the syntax of any IKJTSOxx member of SYS1.PARMLIB, including active members.

Before people at your installation can use the PARMLIB command, you must add PARMLIB to the table of authorized commands. Chapter 11, "Specifying authorized commands/programs, and commands not supported in the background," on page 151 describes how to maintain and update the table.

You should also limit individual users from using the PARMLIB command. You can limit users in one of the following ways:

- Using PARMLIB exit routine IKJPRMX1. When a user issues PARMLIB, IKJPRMX1 can check the user ID and issue a return code to let the user continue, to cancel the PARMLIB command, or to invoke authority checking through RACF.
- Using RACF. You can use the RACF RDEFINE command to define PARMLIB as a RACF resource belonging to the TSOAUTH RACF class. Then give selected users access to the PARMLIB resource using the RACF PERMIT command. Users who will use the CHECK or LIST operands will require READ access to the TSOAUTH-class PARMLIB profile; in order to use the UPDATE operand, UPDATE access to the profile will be needed. Note that users do **not** require a TSO segment to gain access to the PARMLIB profile in the TSOAUTH class.

In addition to controlling access to the PARMLIB command, you can customize the PARMLIB command by writing exits to tailor or monitor its processing. TSO/E provides two exits for the PARMLIB command:

- Initialization (IKJPRMX1)
- Termination (IKJPRMX2)

Using the PARMLIB exits, you can:

- Verify that the user has authority to issue the PARMLIB command.
- Provide a new command buffer to change the operands a user specifies on the command or restrict the use of certain operands. For example, you can correct a user's errors or restrict users from using specific operands.

Writing exits for the PARMLIB command

TSO/E provides an initialization exit and a termination exit for the PARMLIB command. You can use the exits to customize PARMLIB processing for your users. The initialization exit receives control before the PARMLIB command processor invokes the parse service routine to parse the command. The termination exit receives control just before the PARMLIB command processor completes processing. If the initialization exit returns successfully to PARMLIB and PARMLIB processing itself abends, the PARMLIB command processor invokes the termination exit before it terminates.

You can use the initialization exit to check that users have authority to use the PARMLIB command, change the operands that users specify on the command, and change the values of the operands.

You can use the termination exit to perform clean-up or special processing prior to PARMLIB completion. Depending on the processing your initialization exit performs, you may not need a corresponding termination exit.

The following highlights some ways you can use the PARMLIB exits. For more information about how you can use the exits, see “Possible uses” on page 291.
- Verify that the user is authorized to use the PARMLIB command
- Specify that RACF authority checking is to be performed
- Correct a user's errors on the PARMLIB command
- Restrict the operands a user specifies on the command
- Determine how long it takes PARMLIB to execute

TSO/E-supplied exits

TSO/E does not provide default exit routines for the PARMLIB initialization and termination exits.

Entry specifications

For the PARMLIB initialization and termination exits, the contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions for the initialization exit

The PARMLIB initialization exit receives the address of the standard exit parameter list. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32.

Figure 46 on page 289 shows the exit-dependent data that the initialization exit receives beginning at offset +36 (decimal) in the parameter list.

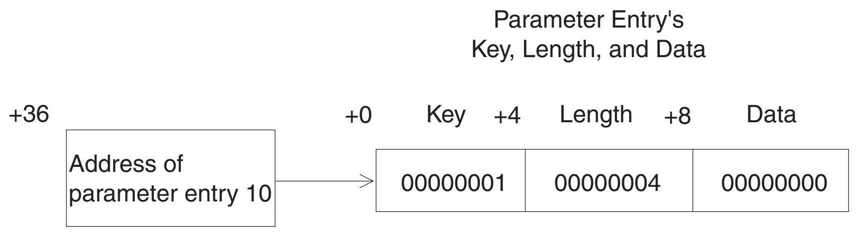


Figure 46. Exit-dependent data for the PARMLIB initialization exit

`Authority value (Parameter Entry 10)`

This parameter (X'0' on entry) lets IKJPRMX1 return a value to indicate the user's authority to use the PARMLIB command. On return from IKJPRMX1, the possible values are:

Value (hex)	Meaning
0	Use RACF to verify the user's authority
4	The user is authorized to use PARMLIB
8	The user is not authorized to use PARMLIB

Parameter descriptions for the termination exit

The PARMLIB termination exit receives the address of the standard exit parameter list. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32. The termination exit does not receive any exit-dependent data.

If the initialization exit returns a new command buffer or an exit-to-exit communication word to the PARMLIB command processor, PARMLIB passes the values of these parameter entries to the termination exit. For more information about the parameter entries, see “TSO/E standard exit parameter list” on page 32.

Return specifications

The contents of the registers on return for both PARMLIB exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the initialization and termination exits

Table 49 shows the return codes that the PARMLIB initialization and termination exits support.

Table 49. Return codes for the PARMLIB initialization and termination exits

Return code (decimal)	Description
0	Exit processing was successful. PARMLIB processing continues.
12	Exit processing was unsuccessful. PARMLIB issues an error message to the user and then terminates processing. The error message identifies the exit that requested termination and a reason code. For more information about the exit reason code, see “Exit reason code” on page 38.

Table 49. Return codes for the PARMLIB initialization and termination exits (continued)

Return code (decimal)	Description
16	Exit processing was unsuccessful. PARMLIB terminates processing without issuing an error message to the user. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the PARMLIB command processor terminates without displaying a message to the user.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exit must be reentrant, refreshable, reusable, and reside in an APF-authorized library. It is inadvisable for the exit to have APF authorization unless it is designed also to be called as the entry point for a job step program.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

In some cases, you may not need to write a termination exit. This depends on:
- Whether the processing that the initialization exit performs requires a termination exit to perform clean-up activities
- How you use the exits to customize PARMLIB processing

If the initialization exit obtains a system resource, you must write a termination exit to free the resource. For example, if the initialization exit obtains storage to return a new command buffer to the PARMLIB command processor, you must provide a termination exit to free the storage. If the initialization exit obtains storage, for example, for a new command buffer, it must obtain the storage from subpool 1.

The initialization exit can change PARMLIB operands using the command buffer. The exit checks the command buffer it receives and determines whether to change any operands. To change the operands, the exit must:
- Obtain storage from subpool 1 for a new command buffer
- Build the new command buffer
- Update the key, length, and data fields for the new command buffer (parameter entry 2)
- Set return code 0 and return control to PARMLIB.

For more information about the parameter entries, see “TSO/E standard exit parameter list” on page 32. For more information about the format of the command buffer, see “Command buffer” on page 35.

Environment

- State: Problem program, APF-authorized
- Key: 8
- AMODE(31), RMODE(ANY), ASCMODE(PRIMARY)

Restrictions and limitations

The PARMLIB exits must reside in an APF-authorized library.

Installing the exits

You must name the exits as follows:

Initialization
IKJPRMX1

Termination
IKJPRMX2

Link-edit each exit as a separate load module. The exits must reside in an APF-authorized library. You can link-edit the exits in any authorized load library, such as a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see “Installing the standard-format exits” on page 39.

Possible uses

Some possible uses of the PARMLIB exits are described below:
- Restrict use of the PARMLIB command to certain users

  The initialization exit can check the user ID and decide, based on your own criteria, to continue processing the PARMLIB command or not. The exit can return a value in parameter entry 10 to indicate that the user is authorized, not authorized, or that the user's authority should be verified through RACF. If the user is not authorized, PARMLIB issues a message, otherwise processing continues.
- Change the operands that the user specifies on the command

  You can use the initialization exit to change the operands that users specify on the PARMLIB command. The initialization exit receives the address of the command buffer. It can change the operands the user specifies on the PARMLIB command by using a new command buffer. For example, the initialization exit can scan the command buffer to:
  - Look for conflicts with the operands a user specifies and correct any errors. For example, the user cannot specify both the UPDATE and LIST operands on the PARMLIB command.
  - Prevent users from specifying certain operands or certain values for operands. For example, you could allow the user to issue the LIST operand but not the UPDATE operand.

  To check the command buffer and change its contents, the initialization exit can:
  - Scan the command buffer and decide, based on your own criteria, to change the command the user issued
  - Obtain storage for a new command buffer
  - Build the new command buffer
  - Update the key, length, and data fields for the new command buffer as follows:

    Key X'02'

    Length
    the length of the new command buffer

    Data the address of the new command buffer

  – Set a return code of 0 and return control to the PARMLIB command processor.

  The exit must not change the command buffer it receives. It must create a new command buffer and return the address of the new command buffer to PARMLIB.

  For more information about the command buffer and the new command buffer, see "TSO/E standard exit parameter list" on page 32. For information about the format of the command buffer, see "Command buffer" on page 35.

- Monitor how long it takes the PARMLIB command to complete processing.

  You can use the initialization and termination exits to monitor the approximate time it takes the PARMLIB command processor to complete processing. When the initialization exit receives control, it can:

  – Invoke the TIME macro

  – Use the exit-to-exit communication word to return the time to the PARMLIB command processor. The exit updates the "Key", "Length", and "Data" fields for the exit-to-exit communication word as follows:

    Key X'01'

    Length
    the length of the data (time)

    Data the data (time)

  – Set a return code of 0 and return to the PARMLIB command processor.

  When the termination exit gets control, it receives the time from the initialization exit in the exit-to-exit communication word. Before the termination exit returns control to PARMLIB, it can invoke the TIME macro. The exit can calculate the time difference between the time from the initialization exit (in the exit-to-exit communication word) and the time it receives from issuing the TIME macro. The result is the approximate time it took the PARMLIB command to complete its processing. The termination exit can include the processing time in a data set. You can then periodically print the data set and review the time calculations.

Chapter 34. Customizing the SUBMIT command and job output processing

Users can issue the TSO/E SUBMIT command to submit batch jobs for processing by JES. They can issue the TSO/E OUTPUT command to process the output of jobs they submit, the TSO/E STATUS command to display the status of any job in the system, and the TSO/E CANCEL command to stop the processing of jobs they submit.

By using JES initialization statements, you can change the default processing performed for jobs submitted by TSO/E users, as well as jobs submitted via other methods, such as external readers or ISPF/PDF. You may want to review, and if necessary, change the default processing.

You can also use the TSO/E SUBMIT exit, the TSO/E OUTPUT, STATUS, and CANCEL exit, JES exits, SMF exits, and RACF resource classes to customize jobs and their output before and after they have been submitted. For example, you can:
- Override the IBM-provided default job processing characteristics
- Monitor, change, or supplement the JCL users use to submit a job
- Cancel a job
- Tailor the way the TSO/E CANCEL, OUTPUT, and STATUS commands work
- Delete, release, or reroute job output
- Control who can submit and cancel jobs by job name
- Protect against unauthorized spool access of the SYSOUT data sets for the TSO/E OUTPUT command
- Allow users to submit jobs on behalf of another user (job owner) without knowing that user's password

Setting defaults for jobs submitted by TSO/E users

You may want to review default processing performed for TSO/E jobs, and, if necessary, change the processing to accommodate jobs submitted by TSO/E users. You can create a separate job class for TSO/E jobs, define additional internal readers (for JES2 only), or specify that TSO/E jobs and their output be held for processing. Most of the defaults are set by using JES initialization statements.

Associating job classes with jobs

Because TSO/E jobs are submitted randomly, and may cause poorer turnaround for other batch jobs, you might want to use a separate job class for TSO/E jobs. If you are using JES2, use the INTRDR initialization statement to specify a job class to be used for all TSO/E jobs. For JES3, you can use the IATUX28 exit to override the class users specify in their JCL and direct TSO/E jobs to a separate class.

To specify a job class to be used for jobs submitted by individual users, use either the ACCOUNT command or the RACF ADDUSER or ALTUSER command depending on whether the user information is defined in the UADS or RACF data base. The job class you specify using these commands overrides the class specified on the JES initialization statement.

For more information about these initialization statements and commands, see the following books:

- For the JES2 initialization statement, see *z/OS JES2 Initialization and Tuning Reference*.
- For the JES3 exit, see *z/OS JES3 Customization*.
- For the ACCOUNT command, see *z/OS TSO/E System Programming Command Reference*.
- For the RACF ADDUSER and ALTUSER commands, see *z/OS Security Server RACF Command Language Reference*.

Specifying the number of jobs that can be read simultaneously (JES2 only)

You may need to adjust the number of JES2 internal readers dedicated to handling jobs submitted by TSO/E users. When users issue the TSO/E SUBMIT command to submit a job for batch processing, a JES internal reader dedicated to TSO/E jobs reads the job onto the spool. The number of internal readers available determines how many TSO/E jobs can be simultaneously read onto the spool, and can affect how quickly jobs are processed. As a starting point, you should define one internal reader for every five users logged onto TSO/E. To specify the number of internal readers, use the JES2 INTRDR initialization statement. For more information about using the JES2 INTRDR initialization statement, see *z/OS JES2 Initialization and Tuning Reference*.

Specifying whether jobs are delayed for processing

By default, jobs submitted by TSO/E users are not held for later processing by an operator. You may want to hold TSO/E jobs for the following reasons:

- Usually there is a trade-off between the performance of batch and on-line processing, so you may want to perform your batch processing off-shift, when it does not degrade the performance of on-line users
- Some jobs require tapes, so you may want to hold those jobs until the tapes are available and have been mounted. If you do not hold the jobs, they tie up resources until the tapes are available.

You can use JES2 or JES3 initialization statements to specify that only certain job classes are held. However, you can specify that *all* jobs submitted by TSO/E users are held by using the JES2 INTRDR initialization statement. To specify a held class used for jobs submitted by individual users if they omit the job class in their JCL, use either the ACCOUNT command or the RACF ADDUSER or ALTUSER command depending on whether the user is defined in the UADS or RACF data base.

Holding output data for processing

After a job has been processed, its JCL and any output data sets the job produces (called SYSOUT data sets) are placed on the output queue for processing by an output device. By default, SYSOUT data sets are not held. You can specify that SYSOUT data sets be held for processing by TSO/E users. Holding data sets for processing allows TSO/E users to use the TSO/E OUTPUT command to view output at their terminals. They can then decide whether to print or purge the output. For more information about the OUTPUT command, see *z/OS TSO/E Command Reference*.

If you use JES2, use the OUTCLASS initialization statement to specify which output job classes are held for processing. If you use JES3, use the SYSOUT initialization statement. For more information about the initialization statements, see:

- *z/OS JES2 Initialization and Tuning Reference*
- *z/OS JES3 Initialization and Tuning Reference*

Customizing how users submit jobs and process the output

After you have changed the defaults for how TSO/E jobs are processed, you can use exits and RACF resource classes to customize the way TSO/E users submit jobs and process the output. For example, you can customize JCL statements, cancel jobs or job output, or allow users to process output from jobs other than their own.

In addition to using the TSO/E SUBMIT and the TSO/E OUTPUT, STATUS and CANCEL exits to customize the way users submit jobs and process the output, you can use JES and SMF. With RACF installed, you can also use the RACF resource classes, JESSPOOL, JESJOBS, and SURROGAT.

In general, you can use JES and SMF exits to perform the same processing as the TSO/E SUBMIT, and OUTPUT, STATUS, CANCEL exit routines.

With RACF installed, you can use the RACF resource class, JESSPOOL, to protect against unauthorized spool access of the SYSOUT data sets for the TSO/E OUTPUT command, and JESJOBS to control who can submit and cancel jobs by job name. For more information about the RACF resource classes, JESSPOOL and JESJOBS, see *z/OS Security Server RACF Security Administrator's Guide*.

With RACF installed, users can be defined to the RACF SURROGAT class. Jobs submitted by surrogate users can be cancelled and/or viewed by surrogate users without knowing the user's password. For more information about submitting a job as a surrogate user, see *z/OS Security Server RACF General User's Guide*. For more information about setting up the RACF SURROGAT class, see *z/OS Security Server RACF Security Administrator's Guide*.

This topic will help you decide which type of exit to use and explains how to use the TSO/E SUBMIT exit (IKJEFF10) and the TSO/E OUTPUT, STATUS, CANCEL exit (IKJEFF53). For information about specific JES and SMF exits you can use, see Chapter 50, "Overview of facilities for customizing TSO/E," on page 699.

The type of exit you use—TSO/E, JES, or SMF—depends on:
- Whether you want to tailor only the jobs TSO/E users submit, or all jobs, including TSO/E jobs
- The information you must access. For example, to access the job queues, use a JES exit.
- When in the job processing cycle you want to get control.

To customize only the jobs TSO/E users submit, use either the TSO/E SUBMIT exit or the TSO/E OUTPUT, STATUS, CANCEL exit. Those exits run in the user's address space. It is more convenient to access user-related information, such as logon information in the RACF data base, from that address space.

To customize the way *all* jobs and their output are processed, use JES or SMF exits. For example, you can use JES or SMF exits to tailor the processing the system does for jobs submitted via ISPF/PDF, external readers, or TSO/E users.

The JES exits run in either the JES2 or JES3 address space, or in the submitted job's address space. Use JES exits to access JES information, such as the job queues. Most SMF exits run in the submitted job's address space. To access accounting information, use an SMF exit.

To avoid using more resources than necessary, you should perform a task as early in the processing cycle as possible. For example, because the TSO/E SUBMIT exit gets control before any JES or SMF exits, it is more efficient to cancel a job using the TSO/E SUBMIT exit instead of using a JES or SMF exit. Most of the JES exits receive control before data sets and devices have been allocated. Most SMF exits receive control after data sets and devices have been allocated, and just before and during job execution.

You can use a combination of exit types to perform a task. For example, you can use the TSO/E SUBMIT exit to access information associated with a user's address space and store the information in the job's JCL as a control statement. You can then use a JES exit to process the information. You should be careful, however, that an exit does not nullify the processing another exit performed.

For more information about JES2, JES3, and SMF exits, see:
- *z/OS JES2 Installation Exits*
- *z/OS JES3 Customization*
- *z/OS MVS Installation Exits*

Writing an exit for the SUBMIT command

Users issue the SUBMIT command to submit one or more batch jobs for background processing by JES. On the command, users can specify one or more data set names or member names that define an input stream consisting of JCL statements and input data. For information about using the SUBMIT command, see *z/OS TSO/E User's Guide*. For information about the syntax of the SUBMIT command, see *z/OS TSO/E Command Reference*.

The SUBMIT command processor reads the JCL statements and writes the data set or data sets that the user specifies into an internal reader of the job entry subsystem. When the SUBMIT command processor reads or generates the first JOB statement, it invokes the exit. Note that if TSO/E builds the JOB statement, the PASSWORD and USER parameters are not included on the statement.

By default, only JOB statements and continuations of the JOB statement are passed to the exit. An embedded comment in the JOB statement is, however, passed as part of the JOB statement. By setting different control switches, the exit can indicate to the SUBMIT command processor that it wants to check additional types of JCL statements as the statements are read from the input data sets. For example, the exit can also check EXEC and DD statements.

You can use the SUBMIT exit to check the user ID and job name and accept, reject, or modify the JCL statements that a user submits. The exit can check the JCL statement and either leave it unchanged, change it, or delete it. The exit can continue a statement or add new statements. It can also have the SUBMIT command processor display a message at the user's terminal, request a response from the user, or cancel a SUBMIT request.

The following highlights some ways you can use the SUBMIT exit. For more information, see “Possible uses” on page 303.

- Cancel a SUBMIT request
- Process statements in addition to the JOB statement
- Delete the current JCL statement
- Add a new statement after the current statement
- Change the current statement
- Supply a password on the JOB statement

TSO/E-supplied exit

If you do not write an exit, an exit routine that TSO/E provides receives control. The default exit routine does not check JCL. It is invoked once for each SUBMIT command (JOB statement). The exit does not perform any processing other than:
- Turning off all switches that cause it to receive control
- Setting the return code to zero
- Returning to the SUBMIT command processor

Entry specifications

The contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of a pointer to an eight word parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions

The following describes the eight word parameter list.

Word 1
The address of the current statement that the SUBMIT command processor passes to the exit.

To delete the current statement, the exit must set word 1 to zero.

To process the statement, the exit must pass back the same address. It can either leave the statement unchanged or change the statement. The exit sets a return code of 0 or 4 depending on the processing it is performing.

If word 1 is zero when the exit gets control, this indicates that the exit must supply the next JCL statement, that is, the return code from the previous call of the exit was 4. The next JCL statement that the exit supplies must be either a continuation of the previous statement or a new statement.

Word 2
The address of a message to be displayed on the user's terminal. The exit must supply the message text when it sets a return code of 8 or 12. The exit must obtain the message buffer and should free it when it receives control again.

The format of the message is “LLtext”, where LL is a 2-byte field that contains the length of the message text area (including the LL field). The maximum text length is 242 bytes. Characters past the maximum length are truncated.

Note: For message IKJ56280A, the maximum length of 242 bytes includes the ‘+’ for a second-level message.

If word 2 is not zero when the exit gets control, this means the return code from the last call to the exit was either 8 or 12. If word 2 is zero, no message is present, which either means this is the first call to the exit, or the last return code was 0 or 4.

Word 3

The address of the response from the user if the exit's last return code was 12. The format of the response is “LLtext”, where LL is a 2-byte field that contains the length of the reply, including the LL field.

The SUBMIT command processor frees the buffer when word 3 is not zero after it calls the exit.

Word 4

The address of the user ID. The user ID can be up to eight characters long. It is left-justified and padded with blanks. If the submitter of the job was a TSO/E batch job, the user ID may be blank.

The value of the user ID is obtained from the PSCBUSER field of the PSCB (Protected Step Control Block), and may sometimes be blank in certain TSO/E batch environments. In particular, if SUBMIT is issued by a TSO/E batch job, and the installation does not use RACF or an equivalent security product to store TSO user logon information, the user ID field will contain blanks.

For more information about the PSCB fields when processing in the background, see the PROFILE Command in the *z/OS TSO/E Command Reference*. For information about the PSCB mapping, see *z/OS TSO/E System Diagnosis: Data Areas*.

Word 5

The address of the control switches, which are contained in a fullword. The exit can use byte 0. The SUBMIT command processor uses bytes 1, 2, and 3. The 4 bytes are described below. For more information about how the bytes are used, see “Summary of using the bytes of word 5” on page 300.

Byte 0

The exit sets these bits to indicate to the SUBMIT command processor the statements it wants to check in addition to the JOB statement. By default, the exit receives control for JOB statements only. The exit may change the bit settings to receive control when SUBMIT reads other statements. For example, if the exit sets bit 3 on, it receives control when the SUBMIT command processor finds operator commands or PROC or PEND statements in the JCL.

Byte 0 contains the following bits:

Bit	**Description**
0	Call exit for JOB statement
1	Call exit for EXEC statement
2	Call exit for DD statements
3	Call exit for commands

4	Call exit for null statement (`//` in columns 1 and 2 and blanks in the remaining columns)
5	Call exit for JES2 control statements -- `/*X` in columns 1-3 (`/*` in columns 1 and 2 and a non-blank character in column 3)
6	Call exit for JES3 control statements or comment statements (`//*` in columns 1-3)
7	Call exit for JES3 control statements (`//*` in columns 1-3 and a non-blank character in column 4)

`Byte 1`

If byte 1 is not zero, it contains a hexadecimal value that indicates the column in which the operand field begins. For example, if the operand field begins in column 16, byte 1 contains X'10'. The SUBMIT command processor supplies a value in byte 1 for all statement types.

`Byte 2`

The SUBMIT command processor sets these bits to identify the current statement to the exit. Byte 2 contains the following bits:

Bit	Description
0	JOB statement
1	EXEC statement
2	DD statement
3	Command
4	Null statement
5	Operand to be continued
6	Statement to be continued
7	Statement is a continuation

`Byte 3`

The SUBMIT command processor sets these bits to identify the current statement to the exit. Byte 3 contains the following bits:

Bit	Description
0	JES2 control statement -- `/*X` in columns 1-3 (`/*` in columns 1 and 2 and a non-blank character in column 3)
1	JES3 control statement or a comment statement -- `//*` in columns 1-3
2	JES3 control statement -- `//*` in columns 1-3 and a non- blank character in column 4
3	SUBMIT-generated job statement. The bit is on for the first statement and all continuations.

`Word 6`

Reserved for the exit's use. The first time the SUBMIT command processor calls the exit, it initializes word 6 to zero. The exit can use the word for counters or switches to keep track of its own processing. The SUBMIT command processor does not change the value between calls.

Word 7
The address of reconstructed logon job accounting information that the exit supplies for the user. The SUBMIT command processor inserts this information into generated JOB statements.

Word 8
The address of a halfword that contains the length of the job accounting information.

Summary of using the bytes of word 5

The SUBMIT command processor uses byte 1 of word 5 to identify the column in which the operand field begins. It also uses byte 3 of word 5 to identify:
- JES2 control statements (bit 0)
- JES3 control statements and comment statements (bit 1)
- JES3 control statements (bit 2)
- SUBMIT-generated job statements (bit 3).

The SUBMIT command processor interprets the following statement as a JES2 control statement. It sets bit 0 of byte 3 on and determines that the operand field begins in column 15. Therefore, it sets byte 1 to X'0F'.

```
/*JES2CTLSTMT OPERAND
```

The SUBMIT command processor interprets the following statements as either JES3 control statements or comment statements.

```
//* OPERAND
//*COMMENT OPERAND
//*JES3CTLSTMT OPERAND
```

The command processor sets bit 1 of byte 3 on. Byte 1 contains the following hexadecimal values. These values indicate the beginning of the operand field for each of these three statements:
- X'05'
- X'0C'
- X'10'

If the current statement has // in columns 1 and 2, but the SUBMIT command processor does not recognize the statement as a JOB, EXEC, or DD statement, it assumes that it is an operator command entered into the input stream. In this case, the command processor sets bit 3 in byte 2 on.

You can continue any type of statement except the following:
- Comment statements
- JES2 control statements
- JES3 control statements

You can indicate that a statement is to be continued by having:
- A comma as the last character of the operand, or
- A non-blank character in column 72.

If a statement has a comma as the last character of the operand, the SUBMIT command processor sets bits 5 and 6 of byte 2 on. In this case, the operand itself in the statement is to be continued.

If a statement has a non-blank character in column 72, and the last character of the operand is *not* a comma, the SUBMIT command processor sets bit 6 of byte 2 on. In this case, the statement itself is to be continued.

Limitation: Unlike the JCL rules, the SUBMIT exit does not allow for the continuation of parameter fields enclosed in quotes. If required, split up the enclosed parameter field into several fields, thereby enclosing each in quotes. For example:
requires to code as follows:

```
//NAMN1 OUTPUT JESDS=ALL,
//      ADDRESS=('text that describes a very long delivery address
        and does not fit on to a single line')
⋮
```

```
//NAMN1 OUTPUT JESDS=ALL,
//      ADDRESS=('text that describes a very long delivery address',
        'and does not fit on to a single line')
⋮
```

The SUBMIT command processor does not pass the exit the preceding JCL statements if they are in a DD DATA (or DD *, for /*X statements) input data stream.

Return specifications

The contents of the registers on return must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes

Return code (decimal)	Description
0	Continue. The SUBMIT command processor processes the current statement and reads the next one. The exit uses return code 0 when it has finished processing the current statement.
4	Invoke the exit again to obtain another statement. The SUBMIT command processor processes the current statement that the exit passes back. It invokes the exit again so the exit can insert a statement. The inserted statement can be: • A continuation of the previous statement (the statement that the exit passed back when it set return code 4), or • A new JCL statement.

Return code (decimal)	Description
8	The SUBMIT command processor displays message IKJ56283I and invokes the exit again. The exit must obtain the message text area and supply the message text. The exit stores the address of the message in word 2 of the parameter list, sets the return code (8), and returns control. The SUBMIT command processor displays the message and then invokes the exit again. After the exit receives control again, it should free the message text area.
12	The SUBMIT command processor displays prompting message IKJ56280A, obtains a response from the user, and invokes the exit again. The exit must obtain the message text area and supply the message text. The exit stores the address of the message in word 2 of the parameter list, sets the return code (12), and returns control. The SUBMIT command processor displays the message, obtains the user's response, and invokes the exit again passing the user's response in word 3 of the parameter list. After the exit receives control again, it should free the message text area. The SUBMIT command processor obtains and frees the reply text area. If the user specifies NOPROMPT on a PROFILE command or uses a CLIST without the PROMPT keyword, a return code of 12 causes the SUBMIT command processor to issue a message and end processing.
16	End processing of the SUBMIT command. If the exit cancels processing, the SUBMIT command processor does not issue a message to the user. Therefore, before the exit sets a return code of 16, it should first use return code 8 to display an appropriate message to the user.

If the exit sets an undefined return code, the SUBMIT command processor issues an error message and ends processing.

Programming considerations

The exit must follow standard linkage conventions. It must save the registers on entry and restore the registers when it returns. The exit must be reusable, reentrant, and refreshable.

Note: If the exit is in LNKLST rather than LPA, the exit does not need to be refreshable.

Exit processing is determined by setting various control switches and return codes. For more information about the control switches and parameter list, see “Entry specifications” on page 297. For information about the return codes, see “Return specifications” on page 301.

By default, the exit gets control only when the first JOB statement is read or generated. The exit can process other types of JCL statements by setting the different control switch bits in byte 0 of word 5 of the parameter list.

The SUBMIT command processor sets control switch bits in bytes 2 and 3 of word 5 to indicate to the exit the type of current statement it is passing. The exit can check these bits to determine the type of statement and whether the statement is a continuation of an operand or a continuation of a statement itself.

Word 6 of the parameter list is for the exit's use. The exit can use the bits in word 6 to keep track of its processing. SUBMIT does not change word 6 between calls to the exit. The exit can set a bit to indicate a certain condition and then check that bit when it receives control again. In this way, the exit can determine what processing it did during the last invocation and what processing it is to perform based on the bit settings.

The exit can also use the return codes to determine how statements are processed and when the exit receives control again. For example, if the exit has finished processing the current statement, it sets a return code of 0. The SUBMIT command processor then processes that statement and continues with the next one. If the exit wants to receive control again to either continue the current statement or add a new statement, it sets a return code of 4. Use return code 8 to display a message and return code 12 to display a message and get a response.

To format the parameter list and assign symbolic names to return codes, use the IKJEFFIE mapping macro instruction. IKJEFFIE creates two assembler DSECTs named IEDSECTD and IESUBCTD. For the format of each DSECT, see *z/OS TSO/E System Diagnosis: Data Areas*.

For the SUBMIT command, issue:

```
IKJEFFIE IETYPE=SUBMIT
```

After establishing addressability with each DSECT, you can refer to the DSECT fields by name.

Environment

- State: Supervisor
- Key: 8
- AMODE(24), RMODE(24)

Restrictions and limitations

A user may use a revised version of the exit that is in a logon procedure's authorized step library. Otherwise, the SUBMIT command processor uses only the system link list to locate and invoke the exit.

Installing the exit

You must name the exit IKJEFF10. You must link-edit the exit in SYS1.LINKLIB as an independent module.

Possible uses

Some possible uses of the SUBMIT exit are described below. The exit can perform several different tasks by using different control switches and return codes. For example, in one invocation, the exit may delete the current statement and also indicate to the SUBMIT command processor that it wants to add a new statement.

- Cancel a SUBMIT request

For example, if a user provides incorrect information on a JCL statement, you can cancel the job before it starts to execute. To cancel a job after it starts to execute, use an SMF exit. For more information about SMF exits you can use, see *z/OS MVS Installation Exits*.

The SUBMIT exit can check the user ID, job name, and individual JCL statements. It can decide, based on your own criteria, to cancel a SUBMIT request. If the exit cancels processing, the SUBMIT command processor does not issue a message. Therefore, before the exit cancels a request, it should display an appropriate message to the user.

To cancel SUBMIT processing, the exit should:

– Obtain storage for a message and supply the message text

– Set word 2 to the address of this message. Set a control bit on in word 6, set a return code of 8, and return.

  The SUBMIT command processor displays the message and then invokes the exit again.

– When the exit gets control again, the bit in word 6 is still on. The exit turns off the bit in word 6, sets a return code of 16, and returns.

- Process statements in addition to the JOB statement

  By default, the exit gets control only for a JOB statement. The exit can set various control switches in word 5 to indicate to the SUBMIT command processor that it wants to process additional types of statements, such as:
  – EXEC statements
  – DD statements
  – Commands
  – JES2 and JES3 control statements.

  To process other types of statements, set the appropriate bit(s) of byte 0 in word 5. For example, if you want to process DD statements in addition to the JOB statement, set bit 2 of byte 0 in word 5.

  When the exit gets control, it checks bytes 2 and 3 of word 5 to identify the type of statement it is getting.

- Delete the current JCL statement

  To delete the current JCL statement, the exit:

  – Sets word 1 to zero

  – Sets a return code based on the type of processing it is doing

  – Optionally, sets a bit on in word 6 (by setting this bit, the exit can determine the processing that occurred when it receives control again)

  – Returns to the SUBMIT command processor

- Add a new statement after the current statement

  To add a new statement, the exit:

  – Sets return code 4, turns on a bit in word 6, and returns.

  – When the exit gets control again, word 1 is zero and the bit in word 6 is still on. The exit supplies the new statement, sets word 1 to the address of the new statement, turns off the bit in word 6, sets a return code of 0, and returns.

- Change the current statement

  The exit can check the JCL statement and either change or add a parameter. Some examples of the changes or additions the exit can make on a JOB statement include:
  – Account number and accounting information
  – Programmer's name

- Job class (CLASS=)
- Priority (PRTY=)
- Special job processing such as placing a job on hold (TYPRUN=)

To add or change the account number or accounting information on the JOB statement, the exit:

- Replaces the information in the current statement
- Sets a return code based on the type of processing it is performing and returns

Note: Word 7 and word 8 of the parameter list contain the address of the reconstructed logon job accounting information and the address of a halfword containing the length of that information, respectively. Those addresses are set up by the SUBMIT exit interface routine (IKJEFF09) and can be used by the SUBMIT exit (IKJEFF10). The information addressed by word 7 has already been processed, so changing the contents of the data pointed to by word 7 in the exit will have no effect on the accounting information on the job statement.

To change parameters or add other parameters on the JOB statement or other JCL statements, the exit:

- Replaces the information in the current statement
- Sets a return code based on the type of processing it is performing and returns

- Continue a statement

  If the exit adds or changes information on the JCL statement, it may have to continue the statement. To continue a statement, the exit:

  - Changes the current statement to indicate continuation. The exit can add a comma after the last character of the last operand, or add a non-blank character in column 72.
  - Sets a return code of 4, sets a bit on in word 6, and returns.
  - When the exit receives control again, word 1 is zero and the bit in word 6 is still on. The exit supplies the address of the continuation statement in word 1 and turns off the bit in word 6. The exit sets a return code based on the processing it is performing and returns.

- Customize the JOB statement

  You can either supply job information the user did not supply, or override the information a user specified. For example, you may want to minimize the information users must specify on the JOB statement, thus keeping inexperienced users from having to use JCL. You may want to supply accounting information or a job class.

  You may also want to include a password. Generally, a password is not needed on the JOB statement. For example, passwords are not required if users submit jobs to execute at the same node on which they are defined.

  The password is required on the JOB statement if:

  - Your RACF users are defined with the same user ID and password on more than one node in a network, and
  - They use the JES control statement `//*ROUTE XEQ` to route jobs to another node.

  The SUBMIT exit can add the password to the JOB statement by retrieving the password from the terminal status block (TSB). However, by default, the passwords of users defined to RACF are not stored in the TSB. To have the logon processor store the passwords in the TSB, use the logon pre-prompt exit (IKJEFLD or IKJEFLD1). Then, the SUBMIT exit can retrieve the password from the TSBPSWD field and change the JOB statement when needed.

Using the logon pre-prompt exit, set the “Store password in TSB” bit on (bit 3 of byte 2 of the control switches) to have the logon processor store the passwords in the TSB for RACF users. For more information, see “Writing a logon pre-prompt exit (IKJEFLD/IKJEFLD1)” on page 90.

- Determine the type of the current statement.

  When the exit gets control, word 1 contains the address of the current statement. The SUBMIT command processor also sets control switches in word 5 to indicate the type of statement it is passing. The different bits of bytes 2 and 3 (word 5) indicate whether the statement is a JOB, EXEC, or DD statement, a command, a comment statement, or a JES2 or JES3 control statement. Several bits also indicate whether the statement is:
  – Null (bit 4 of byte 2)
  – An operand that is to be continued (bit 5 of byte 2)
  – A statement that is to be continued (bit 6 of byte 2)
  – A statement that is a continuation statement (bit 7 of byte 2)

  In general, the exit bases its processing on the type of statement that it gets when it receives control. For example, bit 7 (of byte 2 in word 5) indicates that the current statement is a continuation of the previous statement that the exit processed before it returned control to the SUBMIT command processor.

Writing an exit for the OUTPUT, STATUS, and CANCEL commands

Users issue the OUTPUT, STATUS, and CANCEL commands to process batch jobs.

OUTPUT
Using OUTPUT users can process the output of jobs that they have submitted. A user can:
- Direct the output to the terminal or a data set
- Delete output
- Change output classes
- Route output to a remote workstation
- Release held output for printing

STATUS
Users can issue the STATUS command to display the status of any job in the system.

CANCEL
Users can issue the CANCEL command to stop the processing of jobs that they have submitted.

For information about using the OUTPUT, STATUS, and CANCEL commands, see *z/OS TSO/E User's Guide*. For information about the syntax of the OUTPUT, STATUS, and CANCEL commands, see *z/OS TSO/E Command Reference*.

You can write an exit for OUTPUT, STATUS, and CANCEL to tailor the way the commands operate. For example, you can allow users to cancel other users' jobs, process the output from other users' jobs, or restrict users from obtaining the status of other users' jobs. For more information about how you can use the OUTPUT, STATUS, and CANCEL exit, see “Possible uses” on page 312.

The exit is common to all three command processors. If you do not write an exit, the command processors invoke a default exit routine that TSO/E provides. For more information about the default exit, see “TSO/E-supplied exit” on page 307.

If you have RACF installed, and you are using the RACF resource class JESJOBS and/or JESSPOOL, you may want to review the processing of the IBM supplied exit, IKJEFF53, with the sample exit in member IKJEFF5X of SYS1.SAMPLIB to see if your installation requires the checking of jobname restrictions. For more information, see "TSO/E sample exit" on page 308.

The exit can receive control many times during the processing of the OUTPUT, STATUS, or CANCEL command. When and how many times the exit gets control depends on the number of job names that the user specifies on the command and the return code the exit sets when it returns control to the command processor.

The exit initially receives control whenever a user issues the OUTPUT or CANCEL command, or the STATUS command with operands. When the exit gets control for the first time, it processes the first job name that the user specified on the command and then returns control to the command processor. If the user specified more than one job name, the exit receives control again for each job name that the user specified.

The exit may get control again with the same job name depending on the return code it sets when it returns control to the command processor. For example, the exit can set return code 8 if it wants the command processor to display a message to the user. The exit sets the return code and returns control to the command processor. The command processor displays the message and then invokes the exit again with the same job name. For more information about the return codes, see "Return specifications" on page 310.

If a user issues the STATUS command with no operands, the exit does not receive control. In this case, the STATUS command processor displays only the status of jobs whose job names consist of the user's user ID and one identifying character. For more information about job names and the JOB statement, see *z/OS TSO/E User's Guide*.

TSO/E-supplied exit

If you do not write an exit, a default exit routine that TSO/E provides (IKJEFF53) receives control. The exit routine is common to all three commands.

If a user issues the STATUS command with no operands, the TSO/E exit routine does not receive control. In this case, the STATUS command displays the status of all jobs in the system whose job names consist of the user's user ID and one identifying alphanumeric character or the characters #, @, or $. If a user issues the STATUS command with operands, the exit receives control. The exit lets the user obtain the status of any job in the system.

The TSO/E exit receives control whenever a user issues the CANCEL or OUTPUT command. It restricts a user from:
- Using the CANCEL command to cancel any other user's job
- Using the OUTPUT command to process the output of any other user's job

The exit checks the user ID and the job name the user entered on the CANCEL or OUTPUT command. For CANCEL, the job name must be the user ID plus one character. For OUTPUT, the job name must be the user ID or must start with the user ID. If a user enters an incorrect job name on the CANCEL or OUTPUT command, the exit returns an informational message to the command processor. The command processor displays the message and then invokes the exit again. The exit then cancels the request for that job name. If the user entered more than one job name on the command, the exit receives control again to process each job name in order.

For more information about job names and their correspondence with the user's user ID, see *z/OS TSO/E User's Guide*.

TSO/E sample exit

If you have RACF installed, and your installation plans to use the RACF resource classes JESJOBS and/or JESSPOOL, a sample exit is supplied in SYS1.SAMPLIB that you may review. This sample exit allows the JESJOBS and/or JESSPOOL classes to control the jobname restrictions when these classes are active. If these classes are inactive, the jobname restrictions are controlled by the exit. The jobname restrictions for the TSO/E CANCEL command is the user ID plus one character. The jobname restrictions for the TSO/E OUTPUT command is the user ID. If you are using an installation-written exit, you may want to look at the exit in SYS1.SAMPLIB to determine if you need the same checking of the JESJOBS and JESSPOOL resource classes in your exit.

Note:
1. JESJOBS and JESSPOOL control TSO/E OUTPUT and CANCEL commands.
2. Entry specifications, parameter descriptions, return specifications, programming considerations, installation procedures, environment, and restrictions and limitations are the same for this sample exit.

Entry specifications

The contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of a ten word parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions

On entry, register 1 contains the address of the following ten word parameter list:

`Word 1`
The address of the job name.

`Word 2`
The address of a halfword that contains the length of the job name.

`Word 3`
The address of the user ID.

`Word 4`
The address of one byte that contains the length of the user ID.

`Word 5`
The address of a message to be displayed on the terminal. The exit must

supply the message when it sets a return code of 4 or 8. The format of the message is “LLtext”, where LL is a 2-byte field that contains the length of the entire message (including the LL field). The maximum text length is 127 bytes. The exit must obtain the message text area and should free it when it receives control again.

Word 6
The address of a response from the user if the exit's previous return code was 4. The format of the response is “LLtext”, where LL is a 2-byte field that contains the length of the entire reply (including the LL field). The command processor that calls the exit obtains and frees the reply text area.

Word 7
The address of the one byte command code that indicates which command processor invoked the exit. The command codes are:

0	STATUS command processor
4	CANCEL command processor
8	OUTPUT command processor

Word 8
The address of the jobid if the user specified jobid on the command. If the user did not specify jobid, this word contains zero.

Word 9
The address of a halfword that contains the jobid length. If the user did not specify jobid, the length field contains zero.

Word 10
(For the OUTPUT command only)

The address of a list of pointers and bits that reflects the syntax that the user entered on the OUTPUT command. The total length of this list is five fullwords. The high-order bit of word 10 must be on to indicate the end of the parameter list. The five fullwords are as follows. For more information about the parse descriptor entry (PDE), see *z/OS TSO/E Programming Services*.

Word 1
Pointer to the first PDE for the CLASS operand on the chain of PDEs. If the user did not specify CLASS on the OUTPUT command, this word contains zero.

Word 2
Pointer to the print-data-set-name PDE. If the user did not enter the data set name, this word contains zero.

Word 3
Pointer to the PDE for the class name on the NEWCLASS operand. If the user did not specify a new class, this word contains zero.

Word 4
Pointer to the PDE for the remote station ID. If the user did not enter a destination, this word contains zero.

Word 5
Only the first 12 bits (high-order) are used to reflect the user-entered syntax as follows:

X'8000'
PAUSE (if off, assume NOPAUSE or not applicable)

X'4000'
HOLD (if off, assume NOHOLD or not applicable)

X'2000'
HERE

X'1000'
BEGIN

X'0800'
NEXT

X'0400'
DELETE

X'0200'
PRINT

X'0100'
NEWCLASS

X'0080'
KEEP (if off, assume NOKEEP or not applicable)

X'0040'
DEST

X'0020'
Reserved

X'0010'
Reserved

Return specifications

The contents of the registers on return must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes

Return code (decimal)	Description
0	Valid job name, continue processing. If there are more job names to process, the command processor invokes the exit again with the next job name.
4	Display prompting message IKJ56208A, obtain a response, and invoke the exit again with the same job name. The exit supplies the message text. The exit must obtain the message text area and should free it when it gets control again. The appropriate command processor obtains and frees the reply text area. If the user specifies NOPROMPT on a PROFILE command or uses a CLIST without the PROMPT keyword, a return code of 4 causes the appropriate command processor to issue a message and end processing.
8	Display message IKJ56208I and invoke the exit again with the same job name. The exit supplies the message text. The exit must obtain the message text area and should free it when it receives control again.
12	Incorrect job name. Cancel the request for this job. The appropriate command processor continues and checks for any other job names on the command. If another job name is available, the command processor invokes the exit again with the new job name.
16	Terminate the CANCEL, OUTPUT, or STATUS command.

If the exit uses any other return codes, the command processor issues an error message and ends processing.

Before you use either return code 12 or 16 to cancel processing, consider displaying an informational message to the user. The exit can use a TSO/E service, such as PUTLINE, or use return code 8 to display a message. The exit can also use return code 4 to display a message and obtain a response.

Programming considerations

The exit must follow standard linkage conventions. It must save the registers on entry and restore the registers when it returns. The exit must be reusable and reentrant.

If you use the sample exit (IKJEFF53) in SYS1.SAMPLIB or write an exit that contains the same checking of the RACF resource classes (JESJOBS and/or JESSPOOL), the exit will return without doing any jobname checking for the:

- CANCEL command, if the RACF JESJOBS class is active
- OUTPUT command, if the RACF JESSPOOL class is active

Note: If the RACF JESJOBS and JESSPOOL classes are inactive, jobname checking will be done.

The exit can check the command code in word 7 of the parameter list to determine which command processor invoked the exit. It can check the user ID, job name, and jobid to determine whether the command should execute based on your installation's processing requirements. Word 10 of the parameter list contains the address of a list of pointers and bits that reflect the syntax the user entered on the OUTPUT command. The exit can check word 10 to tailor processing of the OUTPUT command.

The exit can also return a message to the command processor, which the command processor displays to the user. The exit can return an informational message (return code of 8) or a message that requires a response (return code of 4). The exit must obtain the message text area and should free it.

The address of the user response is passed in word 6 of the parameter list. The command processor that invokes the exit obtains and frees the reply text area for the user response.

The exit can set different return codes to handle different types of processing. Before using return code 12, you should consider using return code 8 first to display a message to the user. You can also use return code 4 to display a message and prompt the user for a response. When the exit receives control again (with the same job name), it can then set a return code of 12. The exit itself must keep track of the processing it is performing and the return codes it sets.

The exit can use any of the TSO/E service routines in its processing. For a description of the service routines, see *z/OS TSO/E Programming Services*.

You can use the IKJEFFIE mapping macro instruction to format the parameter list. For CANCEL or STATUS, IKJEFFIE creates an assembler DSECT named IEDSECTD. For OUTPUT, IKJEFFIE creates two assembler DSECTs, IEDSECTD and IEOUTPLD. For the format of each DSECT, see *z/OS TSO/E System Diagnosis: Data Areas*.

For the CANCEL or STATUS command, issue:

```
IKJEFFIE IETYPE=CANST
```

For the OUTPUT command, or if the exit gets control for all three commands, issue:

```
IKJEFFIE IETYPE=OUTPUT
```

After you establish addressability with each DSECT, you can refer to the DSECT fields by name.

Environment

- State: Supervisor
- Key: 8
- AMODE(24), RMODE(24)
- APF-authorized

Restrictions and limitations

A user may use a revised version of the exit that is in a logon procedure's authorized step library. Otherwise, the OUTPUT, STATUS, and CANCEL command processors use only the system link list to locate and invoke the exit.

Installing the exit

You must name the exit IKJEFF53.

You must link-edit the exit into SYS1.LINKLIB as an independent module.

Possible uses

This topic describes some ways you can use the OUTPUT, STATUS, and CANCEL exit. The TSO/E-supplied default exit and the sample exit in SYS1.SAMPLIB, perform many of the tasks listed below. However, if you write your own exit routine, you may want your exit to perform some of the same tasks that the default and sample exits perform, such as restricting users from processing other users' jobs.

- Restrict users from using the STATUS command to obtain the status of other users' jobs

  You may want to restrict most of your users from displaying the status of other users' jobs. You may also want to allow certain users, such as a system programmer or operator, to obtain the status of any job, or a subset of jobs. To tailor the use of the STATUS command, the exit could:
  - Check word 7 to determine if the command code is 0 (STATUS command)
  - Check the user ID, job name, and jobid to determine, based on your own criteria, whether you want the user to obtain the status of the job

    If the user can obtain the job's status, set a return code of 0, and return.

    If the user should not obtain the job's status, set an appropriate return code based on the type of processing you want to perform. For example, the exit could supply a message that prompts the user for a different job name and/or jobid. The exit obtains storage for the message and supplies the message text. The exit supplies the address of the message in word 5 of the parameter list, sets a return code of 4, and returns to the command processor. The command processor displays the message, obtains the user's response, and invokes the exit again passing the user's response in word 6. The exit can then perform processing based on the user's response.

  Instead of prompting the user for information, the exit could simply display an informational message and then cancel the processing for that job. In this case, the exit obtains storage for the message and supplies the message text. The exit supplies the address of the message in word 5 of the parameter list, sets a return code of 8, and returns to the command processor. The command processor displays the message and invokes the exit again. The exit can then set a return code of 12 to cancel the processing of that particular job and return control to the command processor.

- Let certain users cancel other users' jobs

  For example, you may want to allow an operator to cancel any job. The exit could:

  - Check word 7 to determine if the command code is 4 (CANCEL command)
  - Check the user ID to determine if the user can cancel the job

    If the user can cancel the job, the exit sets a return code of 0, and returns.

    If the user cannot cancel the job, the exit can first use return code 8 to display an informational message to the user. When the exit receives control again, it can set a return code of 12 to cancel the request for the job.

- Restrict a user from processing the output of another user's job

  You may want users to process only the output of their own jobs that they have submitted. The exit could:

  - Check word 7 to determine if the command code is 8 (OUTPUT command)
  - Check the user ID and the job name to determine, based on your own criteria, whether the user can process the job output.

    If the user can process the output, the exit sets a return code of 0 and returns.

    If the user should not process the output, the exit can set an appropriate return code based on the processing you want to perform. For example, the exit can supply an informational message and then cancel the processing for that job.

- Restrict a user from changing a job's output class.

  You may want to prevent certain users from using specific output classes. You can use the exit to check if a user is changing a job's output class. The exit can:

  - Check word 7 to determine if the command code is 8 (OUTPUT command)
  - Check the third word pointed to by word 10 and determine if the user specified a new class on the OUTPUT command
  - Check the user ID and determine, based on your own criteria, whether the user can use the new class

    If the user can use the new class or did not specify a new class, set a return code of 0, and return.

    If the user cannot use that output class, the exit can either cancel processing for that job or change the output class the user specified. To change the output class, the exit can first display an informational message to the user. The exit obtains storage for the message and supplies the message text. The exit supplies the address of the message in word 5 of the parameter list, sets a return code of 8, and returns to the command processor. The command processor displays the message and invokes the exit again with the same job name. The exit can then update the new class (word 3 that word 10 points to) and return with a return code of 0.

Chapter 35. Customizing how users print data sets

Users have several options to print data sets. These methods include:

- Issuing the PRINTDS command to print sequential data sets, members of a partitioned data set (PDS), or an entire PDS. Using PRINTDS, users can also print data sets that contain Document Composition Facility (DCF) data.
- Issuing the ALLOCATE command to allocate a system output (SYSOUT) data set.
- Invoking an application that uses the printer support service. After invoking the application, users can select a printer and printing options.

You do not have to perform any tasks to make the PRINTDS and ALLOCATE commands available to users. You can, however, customize the PRINTDS and ALLOCATE commands to tailor how users print data sets.

Using Security Labels on Printed Output

If you have RACF installed, and your installation is using security labels, the security label of the data may be printed on all pages of the printed output. Users can override page labeling, if they have the correct RACF access authority to do so, by using the DPAGELBL and NODPAGELBL keywords of the TSO/E OUTDES command. For more information about security controls in printed output, see *z/OS Security Server RACF Security Administrator's Guide*.

Defining Output Descriptors for the PRINTDS and ALLOCATE Commands

An output descriptor specifies processing options for a system output (SYSOUT) data set. You can define output descriptors to eliminate the need for users to specify output-related operands on the ALLOCATE and PRINTDS commands. If an output descriptor has been defined, users need to specify only the OUTDES operand and the name of the output descriptor.

You can define output descriptors by naming and coding OUTPUT JCL statements. On the OUTPUT JCL statements, you can specify parameters that define printing options, such as:

- COPIES
- DEST
- FORMDEF
- FORMS

You must include the OUTPUT JCL statements in the logon procedure. For more information about using OUTPUT JCL statements to define output descriptors, see "Defining OUTPUT JCL statements" on page 317.

You and your installation's users can optionally use the TSO/E OUTDES command to create and reuse dynamic output descriptors. TSO/E provides two exits for the OUTDES command that allow you to customize the use of OUTDES. For more information about the OUTDES exits, see "Writing exits for the OUTDES command" on page 319. For information about using the OUTDES command and its operands, see *z/OS TSO/E Command Reference*.

Writing Exits for the PRINTDS Command

For PRINTDS, TSO/E provides two exits:
- Initialization (IKJEFY60)
- Termination (IKJEFY64)

Using the PRINTDS exits, you can:
- Restrict certain users from using the PRINTDS command
- Change the operands a user specifies on the command. For example, you can correct a user's errors or restrict users from using certain operands.
- Change the default values for operands that have a fixed default
- Provide installation-defined *pseudo-operands*. If users specify the same PRINTDS operands to print particular data sets, you can define a pseudo-operand that is equivalent to several PRINTDS operands. Users can then specify the pseudo-operand instead of each of the corresponding PRINTDS operands. The PRINTDS initialization exit can replace the pseudo-operand with the actual PRINTDS operands.

For more information about the PRINTDS exits, see “Writing exits for the PRINTDS command” on page 323.

Writing Exits for the ALLOCATE Command

TSO/E provides initialization and termination exits for the ALLOCATE command. For information about these exits, see “Writing exits for the ALLOCATE command” on page 223.

Using the Printer Support Service

In addition to the ALLOCATE and PRINTDS commands, you can use the printer support service to set up printer definitions and then create applications that invoke printer support CLISTs, which are supplied by TSO/E, to print the data sets. Users can invoke the application, select a printer and printing options, and print data sets. Printer support provides a standard interface between your application programs and printers. You do not have to write an application to access a specific printer.

To use printer support, you must have the Information Center Facility installed. Using a set of panels, Information Center Facility administrators define your installation's printers and different output characteristics and fonts. When you define printers, you can define one or more print formats for a particular printer. Each print format and physical printer combination is called a *print definition*. The Information Center Facility provides a default print definition you can copy and modify. You can define all of your installation's printers or define only the printers you will use in your interactive applications. You can also use panels to test a printer before users can access it from your applications. For more information about the default print definition and defining print definitions, see *z/OS TSO/E Administration*.

After you define the print definitions, you can write applications that invoke three printer support CLISTs that TSO/E provides (ICQCPC00, ICQCPC10, and ICQCPC15). The application can be a CLIST or program that runs in an ISPF environment. You can invoke the printer selection CLIST (ICQCPC00) to display a list of all or a subset of the printers you defined. Users can select a printer and formatting options. The application can also format data and display the formatted data to the user for verification. The application can then have the ICQCPC00 CLIST invoke the print CLISTs ICQCPC10 or ICQCPC15, or it can invoke one of the CLISTs directly to print the data.

By using the printer support service, users do not have to code JCL statements or issue the PRINTDS or ALLOCATE commands to print data sets. They can simply invoke the application, and select a printer and printing options.

There are several kinds of applications that you can write using the printer support service. For example, you may want to use printer support to print mail. Your application can display a list of printers that print small amounts of data in a short time. After the user selects a printer, the application sends the data to the selected printer. You can also provide an application that formats and prints documents. The application can display a list of printers and fonts. It can format the data and display the formatted text to the user for verification. After the user verifies the format, the application can invoke either of the print CLISTs to print the text.

For more information about using the printer support CLISTs in your applications, see *z/OS TSO/E Programming Services*.

Defining OUTPUT JCL statements

You use OUTPUT JCL statements to specify processing and formatting options for system output (SYSOUT) data sets and to define output descriptors. You can use OUTPUT JCL statements in different ways to customize how users print data sets.

You can code OUTPUT JCL statements and then refer to these statements in SYSOUT DD statements to process the output of a SYSOUT data set in different ways. For example, you can use the following OUTPUT JCL and DD statements to process the output locally and also to send the SYSOUT data set to a remote location.

```
//OUTPUT1  OUTPUT    DEST=ROUTE2.SCR
//OUTPUT2  OUTPUT    CONTROL=SINGLE
//MYDATA   DD        SYSOUT=C,OUTPUT=(*.OUTPUT1,*.OUTPUT2)
```

By placing OUTPUT JCL statements in logon procedures, you can define output descriptors for your installation's users. An output descriptor associates printer locations and printing options with a single name — the name of the output descriptor. By defining output descriptors, users need not remember or specify different output-related operands when they issue the ALLOCATE or PRINTDS command. Instead, they can specify the OUTDES operand and the name of an output descriptor. For more information about the syntax of the ALLOCATE and PRINTDS operands, see *z/OS TSO/E Command Reference*.

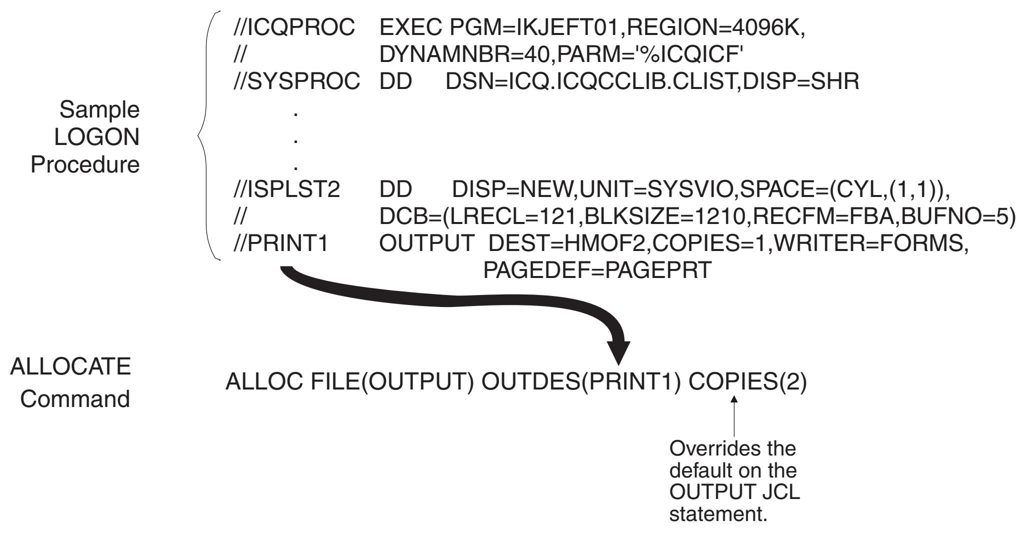
Using OUTPUT JCL statements gives you flexibility in defining printer destinations and formatting options, and controlling the processing. On the OUTPUT JCL statements, you can specify options such as:

- CONTROL - specifies the data set records begin with carriage control characters or the output should be single, double, or triple spaced
- FORMDEF - identifies a library member containing statements that the Print Services Facility™ (PSF) uses to print on the IBM 3800 Printing Subsystem Model 3

- INDEX - sets the left margin for output on an IBM 3211 printer with the indexing feature
- PAGEDEF - identifies a library member containing statements that the Print Services Facility (PSF) uses to print on the IBM 3800 Printing Subsystem Model 3

For more information about the OUTPUT JCL statement, see *z/OS MVS JCL Reference*.

Figure 47 shows a sample logon procedure that defines the output descriptor PRINT1. The figure also shows how users can issue the ALLOCATE command with the OUTDES operand and specify the PRINT1 output descriptor. The COPIES(2) operand on the ALLOCATE command overrides the default defined for PRINT1.

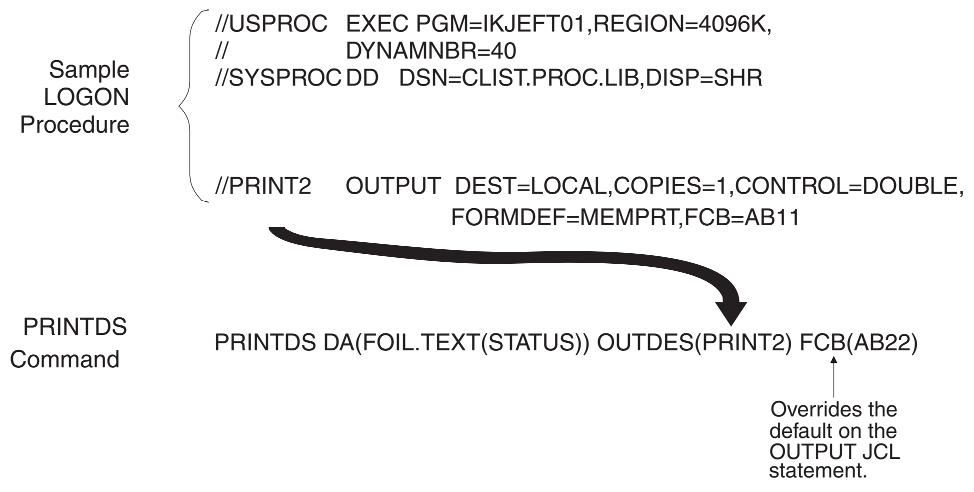


```
//ICQPROC   EXEC PGM=IKJEFT01,REGION=4096K,
//          DYNAMNBR=40,PARM='%ICQICF'
//SYSPROC   DD     DSN=ICQ.ICQCCLIB.CLIST,DISP=SHR
        .
        .
        .
//ISPLST2   DD     DISP=NEW,UNIT=SYSVIO,SPACE=(CYL,(1,1)),
//          DCB=(LRECL=121,BLKSIZE=1210,RECFM=FBA,BUFNO=5)
//PRINT1    OUTPUT  DEST=HMOF2,COPIES=1,WRITER=FORMS,
                    PAGEDEF=PAGEPRT

ALLOC FILE(OUTPUT) OUTDES(PRINT1) COPIES(2)
```

Figure 47. Using the output descriptor in the ALLOCATE command

Figure 48 on page 319 shows a sample logon procedure that defines the output descriptor PRINT2. The figure also shows how users can issue the PRINTDS command and specify the PRINT2 output descriptor. The FCB operand on the PRINTDS command overrides the default defined for PRINT2.



```
//USPROC   EXEC PGM=IKJEFT01,REGION=4096K,
//              DYNAMNBR=40
//SYSPROC  DD   DSN=CLIST.PROC.LIB,DISP=SHR

//PRINT2   OUTPUT  DEST=LOCAL,COPIES=1,CONTROL=DOUBLE,
                   FORMDEF=MEMPRT,FCB=AB11

PRINTDS DA(FOIL.TEXT(STATUS)) OUTDES(PRINT2) FCB(AB22)
```

Figure 48. Using the output descriptor in the PRINTDS command

Writing exits for the OUTDES command

The following sections contain information to help you write exits for the OUTDES command.

Functional description

Users can issue the OUTDES command to create or reuse a dynamic output descriptor. Users can then reference the name of the output descriptor when they issue the ALLOCATE or PRINTDS commands. By using output descriptors, users need not specify different output-related operands on the ALLOCATE or PRINTDS commands. For information about OUTDES and its operands, see *z/OS TSO/E Command Reference*.

TSO/E provides an initialization exit and a termination exit for the OUTDES command. You can use the exits to customize OUTDES processing for your users. The initialization exit receives control before the OUTDES command processor invokes the parse service routine to parse the command. The termination exit receives control just before the OUTDES command processor completes processing. If the initialization exit returns successfully to the OUTDES command processor and OUTDES processing itself abends, the OUTDES command processor invokes the termination exit before it terminates.

You can use the initialization exit to change the operands that users specify on the command or correct user errors when they issue the command. You can use the termination exit to perform clean-up or special processing prior to OUTDES completion. Depending on the processing your initialization exit performs, you may not need a corresponding termination exit.

The following highlights some ways you can use the OUTDES exits. For more information about how you can use the exits, see “Possible uses” on page 322.
- Correct a user's errors on the OUTDES command
- Change the operands a user specifies on the command
- Provide *pseudo-operands* that are equivalent to two or more OUTDES operands.

TSO/E-supplied exits

TSO/E does not provide default exit routines for the OUTDES initialization and termination exits.

Entry specifications

For the OUTDES initialization and termination exits, the contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions for the initialization exit

The OUTDES initialization exit receives the address of the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. The initialization exit does not receive any exit-dependent data.

Parameter descriptions for the termination exit

The OUTDES termination exit receives the address of the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. The termination exit does not receive any exit-dependent data.

If the initialization exit returns a new command buffer or an exit-to-exit communication word to the OUTDES command processor, OUTDES passes the values of these parameter entries to the termination exit. For more information about the parameter entries, see "TSO/E standard exit parameter list" on page 32.

Return specifications

The contents of the registers on return for both OUTDES exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the initialization and termination exits

Table 50 on page 321 shows the return codes that the OUTDES initialization and termination exits support.

Table 50. Return codes for the OUTDES initialization and termination exits

Return code (decimal)	Description
0	Exit processing was successful. OUTDES processing continues.
12	Exit processing was unsuccessful. OUTDES issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the OUTDES command processor. For more information about the exit reason code, see "Exit reason code" on page 38.
16	Exit processing was unsuccessful. OUTDES terminates processing. The OUTDES command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the OUTDES command processor terminates without displaying a message to the user.

The termination exit receives control just before the completion of OUTDES processing. Therefore, the OUTDES command processor may have already successfully created or reused the output descriptor regardless of the return code the termination exit returns.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exits must be reentrant, refreshable, reusable, and not APF-authorized.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

In some cases, you may not need to write a termination exit. This depends on:

- Whether the processing that the initialization exit performs requires a termination exit to perform clean-up activities
- How you use the exits to customize OUTDES processing.

If the initialization exit obtains a system resource, you must write a termination exit to free the resource. For example, the initialization exit may obtain storage to return a new command buffer to the OUTDES command processor. In this case, you must provide a termination exit to free the storage for the new command buffer.

The initialization exit can change OUTDES operands using the command buffer. The exit checks the command buffer it receives and determines whether to change any operands. To change the operands, the exit must:

- Obtain storage for a new command buffer
- Build the new command buffer
- Update the key, length, and data fields for the new command buffer (parameter entry 2)
- Set return code 0 and return control to OUTDES.

For more information about the parameter entries, see “TSO/E standard exit parameter list” on page 32. For more information about the format of the command buffer, see “Command buffer” on page 35.

Environment

- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY), ASCMODE(PRIMARY)

Installing the exits

You must name the exits as follows:

Initialization
IKJEFY11

Termination
IKJEFY12

Link-edit each exit as a separate load module. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in:

- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see “Installing the standard-format exits” on page 39.

Possible uses

Some possible uses of the OUTDES exits are described below:

- Change the operands that the user specifies on the command

  You can use the initialization exit to change the operands that users specify on the OUTDES command. The initialization exit receives the address of the command buffer. It can change the operands the user specifies on the OUTDES command by using a new command buffer. For example, the initialization exit can scan the command buffer and correct any errors on the command.
- Provide installation-defined pseudo-operands

  You can provide *pseudo-operands* for your installation's users that are equivalent to two or more OUTDES operands. Providing pseudo-operands makes it easier for users to issue the OUTDES command. Users need not remember several OUTDES operands. They can specify the pseudo-operand.

  For example, you could associate a pseudo-operand named OUT1 with four OUTDES operands. The initialization exit can scan the command buffer. If the exit finds the pseudo-operand OUT1, it can replace OUT1 with the actual OUTDES operands and return a new command buffer.

To check the command buffer and change its contents, the initialization exit can:

- Scan the command buffer and decide, based on your own criteria, to change the command the user issued
- Obtain storage for a new command buffer
- Build the new command buffer
- Update the key, length, and data fields for the new command buffer as follows:

  Key X'02'

Length
the length of the new command buffer

Data the address of the new command buffer

- Set a return code of 0 and return control to the OUTDES command processor.

The exit must not change the command buffer it receives. It must create a new command buffer and return the address of the new command buffer to OUTDES. For more information about the command buffer and the new command buffer, see “TSO/E standard exit parameter list” on page 32. For information about the format of the command buffer, see “Command buffer” on page 35.

You must also write a termination exit to free the storage the initialization exit obtains for the new command buffer. When the OUTDES command processor invokes the termination exit, it passes the address of the new command buffer to the termination exit. The termination exit frees the storage for the new command buffer.

Writing exits for the PRINTDS command

Users issue the PRINTDS command to print sequential data sets, members of a partitioned data set (PDS), or an entire PDS. They can print data sets that contain Document Composition Facility (DCF) data. Users can also route the command's output to a remote workstation or a user at a specific remote workstation, or to a data set. Using PRINTDS, users can specify a variety of formatting options. For more information about using PRINTDS, see *z/OS TSO/E User’s Guide*. For information about PRINTDS and its operands, see *z/OS TSO/E Command Reference*.

TSO/E provides an initialization exit and a termination exit for the PRINTDS command. You can use the exits to customize PRINTDS processing for your users. The initialization exit receives control before the PRINTDS command processor invokes the parse service routine to parse the command. The termination exit receives control just before the PRINTDS command processor completes processing. If the initialization exit returns successfully to PRINTDS and PRINTDS processing itself abends, the PRINTDS command processor invokes the termination exit before it terminates.

You can use the initialization exit to restrict certain users from using the PRINTDS command, change the operands that users specify on the command, and change the values of PRINTDS operands that have fixed default values. For more information about the operands the initialization exit can change, see “Programming considerations” on page 327.

You can use the termination exit to perform clean-up or special processing prior to PRINTDS completion. Depending on the processing your initialization exit performs, you may not need a corresponding termination exit.

The following highlights some ways you can use the PRINTDS exits. For more information about how you can use the exits, see “Possible uses” on page 330.

- Restrict certain users from using the PRINTDS command
- Change the default values for operands that have a fixed default
- Correct a user's errors on the PRINTDS command
- Change the operands a user specifies on the command
- Provide *pseudo-operands* that are equivalent to two or more PRINTDS operands
- Determine how long it takes PRINTDS to execute

TSO/E-supplied exits

TSO/E does not provide default exit routines for the PRINTDS initialization and termination exits.

Entry specifications

For the PRINTDS initialization and termination exits, the contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions for the initialization exit

The PRINTDS initialization exit receives the address of the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32.

Figure 49 shows the exit-dependent data that the initialization exit receives beginning at offset +36 (decimal) in the parameter list.

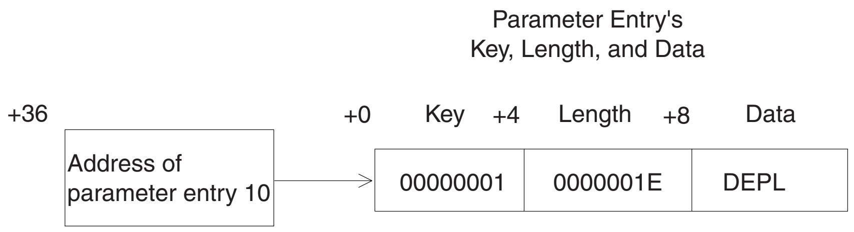


Figure 49. Exit-dependent data for the PRINTDS initialization exit

`DEPL (Parameter Entry 10)`
This parameter entry contains the default exit parameter list (DEPL). Table 51 shows the format of the DEPL.

For more information about the PRINTDS command and its operands, see *z/OS TSO/E Command Reference*.

Table 51. Format of the default exit parameter list

Offset (decimal)	Length	Contents or meaning
+0	4	Reserved.

Table 51. Format of the default exit parameter list (continued)

Offset (decimal)	Length	Contents or meaning
+4	4	The length of the default exit parameter list (DEPL). The value is X'1E'. The length is the length of the data starting at offset +8 in the DEPL. The length does not include the first two words: • Offset +0 - reserved • Offset +4 - length (X'1E') However, the length does include the four reserved bytes beginning at offset +34.
+8	2	The value for the BIND operand in binary. The default is 0. The possible values are 0 - 255.
+10	2	The value for the TMARGIN operand in binary. The default is 0. The possible values are 0 - 4094. However: `PAGELEN - TMARGIN - BMARGIN ≥ 6`
+12	2	The value for the BMARGIN operand in binary. The default is 0. The possible values are 0 - 4094. However: `PAGELEN - TMARGIN - BMARGIN ≥ 6`
+14	2	The value for the PAGELEN operand in binary. The default is 60. The possible values are 6 - 4094. However: `PAGELEN - TMARGIN - BMARGIN ≥ 6`
+16	1	The value for the CLASS operand in character format. The default is A. The possible values are A - Z or 0 - 9.
+17	1	Reserved.
+18	2	The value for the BURST/NOBURST operand in binary. The possible values are: 1 - BURST 2 - NOBURST The default is 2 (NOBURST).
+20	2	The value for the COPIES operand in binary. The default is 1. The possible values are 1 - 255.
+22	2	The value for the HOLD/NOHOLD operand in binary. The possible values are: 1 - HOLD 2 - NOHOLD The default is 2 (NOHOLD).
+24	2	The value for the MEMBER/DIRECTORY/ALL operand in binary. The possible values are: 1 - MEMBER 2 - DIRECTORY 3 - ALL The default is 3 (ALL).

Table 51. Format of the default exit parameter list (continued)

Offset (decimal)	Length	Contents or meaning
+26	2	The value for the NUM/SNUM/NONUM operand in binary. The possible values are: 1 - NUM 2 - SNUM 3 - NONUM The default is 3 (NONUM).
+28	2	The NUM or SNUM column location that specifies the column in which the line number field begins. If NUM or SNUM is specified, the value must be 1-32,760 in binary. If NONUM is specified, the column location is ignored.
+30	2	The NUM or SNUM field length. If NUM or SNUM is specified, the field length must be 1 - 8 in binary. If NONUM is specified, the field length is ignored.
+32	2	The value for the TITLE/NOTITLE operand in binary. The possible values are: 1 - TITLE 2 - NOTITLE The default is 1 (TITLE). **Note:** For more information about the TITLE/NOTITLE values, see "Restrictions and limitations" on page 329.
+34	4	Reserved.

Parameter descriptions for the termination exit

The PRINTDS termination exit receives the address of the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. The termination exit does not receive any exit-dependent data.

If the initialization exit returns a new command buffer or an exit-to-exit communication word to the PRINTDS command processor, PRINTDS passes the values of these parameter entries to the termination exit. For more information about the parameter entries, see "TSO/E standard exit parameter list" on page 32.

Return specifications

The contents of the registers on return for both PRINTDS exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the initialization and termination exits

Table 52 on page 327 shows the return codes that the PRINTDS initialization and termination exits support.

Table 52. Return codes for the PRINTDS initialization and termination exits

Return code (decimal)	Description
0	Exit processing was successful. PRINTDS processing continues. When the initialization exit returns control, the PRINTDS command processor checks the values of the operands that the exit returns in the DEPL. If any of the values are not valid, the PRINTDS command processor displays an error message to the user. The error message indicates which operands were not valid. PRINTDS then terminates processing. For information about the possible values the exit routine can return in the DEPL, see Table 53 on page 329.
12	Exit processing was unsuccessful. PRINTDS issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the PRINTDS command processor. For more information about the exit reason code, see “Exit reason code” on page 38.
16	Exit processing was unsuccessful. PRINTDS terminates processing. The PRINTDS command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the PRINTDS command processor terminates without displaying a message to the user.

The termination exit receives control just before the completion of PRINTDS processing. Therefore, the input data set may have already successfully printed regardless of the return code the termination exit returns.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exits must be reentrant, refreshable, reusable, and not APF-authorized.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

In some cases, you may not need to write a termination exit. This depends on:

- Whether the processing that the initialization exit performs requires a termination exit to perform clean-up activities
- How you use the exits to customize PRINTDS processing.

If the initialization exit obtains a system resource, you must write a termination exit to free the resource. For example, the initialization exit may obtain storage to return a new command buffer to the PRINTDS command processor. In this case, you must provide a termination exit to free the storage for the new command buffer. If the initialization exit obtains storage, for example, for a new command buffer, it must obtain the storage from subpool 1.

To monitor how long it takes the PRINTDS command to execute, you need an initialization and a termination exit. Use the exits to calculate the time difference between when the initialization exit gets control and when the termination exit gets control.

The PRINTDS command has three classes of operands:

1. Operands that have a fixed default value if the user does not specify the operand (for example, BIND and PAGELEN)
2. Operands that PRINTDS sets dynamically based on the data set's attributes and other operands the user specifies (for example, DCF/NODCF and TRC/NOTRC)
3. Operands that do not have a default value (for example, FLASH and CHARS).

For more information about the PRINTDS command and its operands, see *z/OS TSO/E Command Reference*.

The initialization exit can change PRINTDS operands using the command buffer. The exit checks the command buffer it receives and determines whether to change any operands. To change the operands, the exit must:

- Obtain storage for a new command buffer
- Build the new command buffer
- Update the key, length, and data fields for the new command buffer (parameter entry 2)
- Set return code 0 and return control to PRINTDS

For more information about the parameter entries, see “TSO/E standard exit parameter list” on page 32. For more information about the format of the command buffer, see “Command buffer” on page 35.

In addition to changing the command buffer, the initialization exit lets you easily change the first class of PRINTDS operands, which are operands that have a fixed default value. If the user does not explicitly specify these operands on the PRINTDS command, PRINTDS uses the fixed default value. The fixed default values do not change based on the data set's attributes or other operands the user specifies.

The initialization exit receives the fixed default values as exit-dependent data in the *default exit parameter list* (DEPL). To change these values, the initialization exit updates the value in the DEPL, sets a return code of 0, and returns control to the PRINTDS command processor.

If the initialization exit returns a value for an operand in the DEPL and that operand is also specified in the command buffer, PRINTDS uses the value from the command buffer rather than the value in the DEPL. For example, suppose the user issues the following PRINTDS command:

```
PRINTDS DATASET(MY.DATA) CLASS(3)
```

If the initialization exit sets the CLASS value in the DEPL to ‘Z’, PRINTDS ignores the value in the DEPL and uses the value for the CLASS operand in the command buffer, which is ‘3’.

Table 53 on page 329 shows the PRINTDS operands with a fixed default value that the initialization exit receives in the DEPL. The table shows the fixed default value and the possible values for each operand. The default value is the value that PRINTDS uses if the user does not explicitly specify the operand on the PRINTDS command. PRINTDS also provides the value of the TITLE/NOTITLE operand in the DEPL even though PRINTDS will change the value if it conflicts with the data set's attributes. For more information about TITLE/NOTITLE, see “Restrictions and limitations.”

For more information about the format of the DEPL, see “Parameter descriptions for the initialization exit” on page 324.

Table 53. PRINTDS operands in the default exit parameter list (DEPL)

Operand	Fixed default value	Possible values
BIND	0 (binary)	0 - 255
TMARGIN	0 (binary)	0 - 4094
BMARGIN	0 (binary)	0 - 4094
PAGELEN	60 (binary)	6 - 4095
CLASS/SYSOUT (note 1)	A (character)	A - Z and 0 - 9
BURST/NOBURST (note 1)	• NOBURST • 2 (binary)	• 1 - BURST • 2 - NOBURST
COPIES (note 1)	1 (binary)	1 - 255
HOLD/NOHOLD (note 1)	NOHOLD 2 (binary)	• 1 - HOLD • 2 - NOHOLD
MEMBER/DIRECTORY/ALL	• ALL • 3 (binary)	• 1 - MEMBER • 2 - DIRECTORY • 3 - ALL
NUM/SNUM/NONUM	• NONUM • 3 (binary)	• 1 - NUM • 2 - SNUM • 3 - NONUM
TITLE/NOTITLE (note 2)	TITLE, if possible 1 (binary) Otherwise, NOTITLE 2 (binary)	• 1 - TITLE • 2 - NOTITLE

Note:
1. These operands apply only when users print a SYSOUT data set.
2. For more information about TITLE/NOTITLE values, see “Restrictions and limitations.”

The PRINTDS command processor invokes Dynamic Allocation to allocate and de-allocate data sets during its processing. Allocation-attribute operands such as COPIES, CLASS, and HOLD/NOHOLD are passed as text units to Dynamic Allocation, which provides the allocation input validation exit routine (IEFDB401) that you can also use. For more information about IEFDB401, see *z/OS MVS Installation Exits*.

Environment

- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY)

Restrictions and limitations

PRINTDS provides the value of the TITLE/NOTITLE operand to the initialization exit in the DEPL. However, if the value the initialization exit provides in the DEPL conflicts with the data set's attributes, PRINTDS ignores the value.

If TITLE is the default value, PRINTDS prints title lines on each output page, only if both of the following are true:

- The CCHAR, DCF, or TRC operands are not specified
- The input data set does not contain carriage control characters

Otherwise, PRINTDS uses NOTITLE.

If NOTITLE is the default, PRINTDS does not print title lines for sequential data sets or members of partitioned data sets. However, PRINTDS always prints the directory of a PDS with title lines. For more information about the TITLE and NOTITLE operands, see *z/OS TSO/E Command Reference*.

Installing the exits

You must name the exits as follows:

Initialization
IKJEFY60

Termination
IKJEFY64

Link-edit each exit as a separate load module. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see “Installing the standard-format exits” on page 39.

Possible uses

Some possible uses of the PRINTDS exits are described below:

- Restrict certain users from using the PRINTDS command

  The initialization exit can check the user ID and decide, based on your own criteria, to cancel the PRINTDS command. The exit can first display an informational message to the user, for example, using PUTLINE. The exit can then set a return code of 16 and return to the PRINTDS command processor.
- Change the operands that have a fixed default value

  The PRINTDS initialization exit receives the default exit parameter list (DEPL). The DEPL contains the default values of PRINTDS operands that have a fixed default value. The exit updates the appropriate values in the DEPL, sets a return code of 0, and returns to the PRINTDS command processor.
- Change the operands that the user specifies on the command

  You can use the initialization exit to change the operands that users specify on the PRINTDS command. The initialization exit receives the DEPL, which it can use to change the fixed default values of specific operands.

  The initialization exit also receives the address of the command buffer. It can change the operands the user specifies on the PRINTDS command by using a new command buffer. For example, the initialization exit can scan the command buffer to:
  - Look for conflicts with the operands a user specifies and correct any errors. For example, the user cannot specify both the TITLE and CCHAR operands on the PRINTDS command.
  - Prevent users from specifying certain operands or certain values for operands.

  To check the command buffer and change its contents, the initialization exit can:

  - Scan the command buffer and decide, based on your own criteria, to change the command the user issued
  - Obtain storage for a new command buffer
  - Build the new command buffer
  - Update the key, length, and data fields for the new command buffer as follows:

    Key X'02'

    Length
    the length of the new command buffer

    Data the address of the new command buffer
  - Set a return code of 0 and return control to the PRINTDS command processor.

  The exit must not change the command buffer it receives. It must create a new command buffer and return the address of the new command buffer to PRINTDS. For more information about the command buffer and the new command buffer, see "TSO/E standard exit parameter list" on page 32. For information about the format of the command buffer, see "Command buffer" on page 35.

  You must also write a termination exit to free the storage the initialization exit obtains for the new command buffer. When the PRINTDS command processor invokes the termination exit, it passes the address of the new command buffer to the termination exit. The termination exit frees the storage for the new command buffer.

- Provide installation-defined pseudo-operands

  If users at your installation print data sets with the same types of characteristics, you can define *pseudo-operands* that are equivalent to two or more PRINTDS operands. Providing pseudo-operands makes it easier for users to issue the PRINTDS command. Users need not remember several PRINTDS operands. They can specify the pseudo-operand.

  For example, you could associate a pseudo-operand named PRTINV with three PRINTDS operands. The initialization exit can scan the command buffer. If the exit finds the pseudo-operand PRTINV, it can:

  - Obtain storage for a new command buffer
  - Build a new command buffer and replace the pseudo-operand with the appropriate PRINTDS operands in the new command buffer
  - Update the "Key", "Length", and "Data" fields for the new command buffer as follows:

    Key X'02'

    Length
    the length of the new command buffer

    Data the address of the new command buffer
  - Set a return code of 0 and return control to the PRINTDS command processor.

  You must also provide a termination exit. The termination exit must free the storage that the initialization exit obtained for the new command buffer.

- Monitor how long it takes the PRINTDS command to complete processing.

  You can use the initialization and termination exits to monitor the approximate time it takes the PRINTDS command processor to complete processing. When the initialization exit receives control, it can:

  - Invoke the TIME macro

– Use the exit-to-exit communication word to return the time to the PRINTDS command processor. The exit updates the “Key”, “Length”, and “Data” fields for the exit-to-exit communication word as follows:

  Key X'01'

  Length
  the length of the data (time)

  Data the data (time)

– Set a return code of 0 and return to the PRINTDS command processor.

When the termination exit gets control, it receives the time from the initialization exit in the exit-to-exit communication word. Before the termination exit returns control to PRINTDS, it can invoke the TIME macro. The exit can calculate the time difference between the time from the initialization exit (in the exit-to-exit communication word) and the time it receives from issuing the TIME macro. The result is the approximate time it took the PRINTDS command to complete its processing. The termination exit can include the processing time in a data set. You can then periodically print the data set and review the time calculations.

Chapter 36. Customizing how users send and retrieve messages

Users issue the SEND command to send messages to other users, the operator, or to a specific operator or operator console. Users who are authorized to use the OPERATOR command can issue the SEND subcommand of the OPERATOR command to send messages. Users issue the LISTBC command to retrieve messages that other users have sent.

Before users at your installation can use SEND and LISTBC, you must add the commands to the table of authorized commands. For more information, see Chapter 11, “Specifying authorized commands/programs, and commands not supported in the background,” on page 151.

You give users authorization to use the OPERATOR command when you define them to either the user attributes data set (UADS) or the RACF data base. For more information about the UADS and RACF data base, see Part 4, “Maintaining the UADS, RACF data base, and broadcast data set,” on page 193.

By default, the SEND command processor and the OPERATOR SEND subcommand processor may store a message in the mail section of the broadcast data set depending on the operands the user specifies. For example, a user can specify that SEND or OPERATOR SEND store the message if a target user is not logged on. Authorized users of the OPERATOR command can also issue the SEND subcommand to send notices. Notices are messages that are intended for all users on the system. The OPERATOR SEND subcommand processor stores notices in the notices section of the broadcast data set.

By default, when a user issues the LISTBC command to retrieve a stored message, LISTBC retrieves the message from the mail section of the broadcast data set. For more information about SEND and LISTBC, see *z/OS TSO/E Command Reference*. For more information about the SEND subcommand of OPERATOR, see *z/OS TSO/E System Programming Command Reference*.

You can customize how users send, store, and retrieve messages (mail) at your installation. You can define installation defaults in SYS1.PARMLIB member IKJTSOxx to tailor SEND, OPERATOR SEND, and LISTBC processing. Using IKJTSOxx, you can prevent all users from using the SEND command. You can also prevent users who are authorized to use the OPERATOR command from using the SEND subcommand. You can also write exits to restrict certain users from using SEND, OPERATOR SEND, or LISTBC.

You can define console names for the MCS consoles at your installation using SYS1.PARMLIB member CONSOLxx. Users can then specify the console name instead of the console ID when sending messages to an operator console. Console names are generally easier to remember and use than console IDs. For more information about defining console names, refer to *z/OS MVS Planning: Operations*.

Note: The names for extended MCS consoles are dynamically defined during console activation. Installations do not need to define names for extended MCS consoles. If you have RACF installed, your installation can control which users can send messages to other users. To control the use of the SEND command in this way, use the RACF security resource class SMESSAGE. For example, a user may send messages to another user only if permitted to the receiving user's resource within the SMESSAGE class. For information about setting up the SMESSAGE resource class, see *z/OS Security Server RACF Security Administrator's Guide*.

Because SEND, OPERATOR SEND, and LISTBC access the broadcast data set to store and retrieve messages, you may experience contention for the data set. Instead of storing messages (mail) in the broadcast data set, you can use individual user logs to store the messages. A user log is a data set that the SEND command or subcommand processor uses to store messages in rather than storing the messages in the broadcast data set. Using user logs may reduce possible contention for the broadcast data set.

Protection of messages from users at different security labels is another reason for using individual user logs. If you have RACF installed, and your installation is using security labels, you can customize the TSO/E SEND and LISTBC commands by setting some of the operand values of the SEND PARMLIB parameter to determine security checking of the individual user logs. This is explained in the operand descriptions in this chapter.

If you use user logs, LISTBC retrieves messages from the user log. However, OPERATOR SEND continues to store notices in the notices section of the broadcast data set and LISTBC retrieves notices from the broadcast data set.

You can use various SEND, OPERATOR SEND, and LISTBC exits to customize command processing and the use of user logs. Using the exits, you can:
- Restrict certain users from using the command
- Change the operands users specify on the command
- Encrypt and decrypt messages
- Format messages or add information to messages that are stored
- Allocate user logs instead of having the LISTBC command processor allocate them

Note: The EDIT and TEST commands have a SEND subcommand that operates the same as the SEND command. The TEST command also has a LISTBC subcommand that operates the same as the LISTBC command. The customization tasks for the SEND and LISTBC commands that this chapter describes also affect the SEND subcommands of EDIT and TEST and the LISTBC subcommand of TEST.

Unless otherwise indicated, when this chapter refers to the SEND command or SEND messages, the information applies to both SEND and OPERATOR SEND.

This chapter describes the different ways you can customize SEND, OPERATOR SEND, and LISTBC processing by:
- Defining installation defaults in SYS1.PARMLIB member IKJTSOxx
- Storing SEND messages in individual user logs
- Writing SEND, OPERATOR SEND, and LISTBC exits to tailor how users send and retrieve messages. Using the exits, you can change the installation defaults you define in IKJTSOxx and tailor the use of user logs.

Defining installation defaults for SEND, OPERATOR SEND, and LISTBC

You can use SYS1.PARMLIB member IKJTSOxx to define installation defaults for SEND, OPERATOR SEND, and LISTBC processing. In IKJTSOxx, you can:
- Specify whether users can issue the SEND command at your installation

- Specify whether users who are authorized to use the OPERATOR command can issue the SEND subcommand to send messages or create notices
- Specify whether SEND can save messages in user logs
- Specify whether SEND stores messages in the broadcast data set or in user logs, and identify the qualifiers for the user log data set name
- Allow LISTBC to retrieve messages from the user log andthe broadcast data set, from the broadcast data set only, or from the user log only
- Specify whether SEND stores messages in the broadcast data set, if your installation is using user logs and the target user does not have a user log
- Specify whether the individual user logs are security protected from the user
- Specify the name of the broadcast data set and associated processing options

TSO/E provides a sample IKJTSO00 member in SYS1.SAMPLIB. To define installation defaults in IKJTSOxx, do the following:
- If you have not already done so, copy sample member IKJTSO00 from SYS1.SAMPLIB to SYS1.PARMLIB. You may have already copied IKJTSO00 to define other installation defaults.
- You can create alternative members in SYS1.PARMLIB using the IKJTSOxx naming convention.
- Edit the member in SYS1.PARMLIB and locate the SEND PARMLIB parameter.
- Specify the operands for your installation's requirements

You can make the SEND defaults in IKJTSOxx take effect immediately using the PARMLIB command. You can also use the PARMLIB command to list the current SEND defaults and to check the syntax of any IKJTSOxx member. For more information about using the PARMLIB command, see *z/OS TSO/E System Programming Command Reference*.

The operands for the SEND PARMLIB parameter are:
- USERSEND
- OPERSEND
- SAVE
- LOGNAME
- CHKBROD
- USEBROD
- MSGPROTECT
- SYSPLEXSHR
- OPERSEWAIT
- USERLOGSIZE
- BROADCAST

For more information about these operands, see *z/OS MVS Initialization and Tuning Reference*.

When you IPL your system, a *SEND PARMLIB control block* (IKJEESCB) is created.The control block contains the SEND and LISTBC installation defaults you specify in IKJTSOxx. TSO/E uses the control block for SEND and LISTBC processing.

The SEND, OPERATOR SEND, and LISTBC exits receive the address of a **copy** of the SEND PARMLIB control block. Using the exits, you can change the values in the copy of the control block to tailor SEND and LISTBC processing for individual users. However, you cannot use the exits to change the installation defaults in SYS1.PARMLIB member IKJTSOxx. To change the installation defaults, you must:

- Edit member IKJTSOxx in SYS1.PARMLIB
- Change the values of the operands in the SEND PARMLIB parameter
- Use the PARMLIB command with the UPDATE operand to make the changes take effect immediately.

If you do not define installation defaults in SYS1.PARMLIB member IKJTSOxx, the SEND, OPERATOR SEND, and LISTBC commands operate as provided by TSO/E. That is, SEND and OPERATOR SEND store messages in the broadcast data set and LISTBC retrieves stored messages from the broadcast data set.

If you write exits for the commands or subcommand, the exits receive a copy of a default control block that has the following values:

USERSEND
ON

OPERSEND
ON

SAVE ON

LOGNAME
* (which means the current broadcast data set)

CHKBROD
OFF

USEBROD
ON

MSGPROTECT
OFF

SYSPLEXSHR
OFF

OPERSEWAIT

BROADCAST
(see “BROADCAST keyword for IKJTSOxx”)

For more information about using the exits, see “Writing exits for the SEND, OPERATOR SEND, and LISTBC commands” on page 341.

BROADCAST keyword for IKJTSOxx

Use the BROADCAST keyword of the IKJTSOxx member of parmlib to specify the name of a broadcast data set and associated processing options. You can switch broadcast data sets dynamically, without an IPL, by using the SET IKJTSO=xx system command or PARMLIB UPDATE.

For more information about specifying the name of a broadcast data set, see *z/OS MVS Initialization and Tuning Reference*. For more information about the SET IKJTSO=xx command, see *z/OS MVS System Commands*. For more information about PARMLIB UPDATE, see *z/OS TSO/E System Programming Command Reference*.

The default control block contains the following values for the BROADCAST keyword:

data-set-name
SYS1.BRODCAST

volume-name
no volume

time-out
five seconds

switch-prompt
PROMPT

Storing SEND messages

You can store all SEND messages in the broadcast data set, or you can use individual user logs to store the messages. IBM suggests implementing user logs to avoid the single point of failure that exists if all users rely on the broadcast data set for their individual mail. If you use individual user logs, you can use them for some of your users and use the broadcast data set for other users by writing exits for the SEND, OPERATOR SEND, and LISTBC commands.

User logs provide large processor constraint relief by eliminating possible contention for the broadcast data set. If your installation has large processors, such as the IBM 3090, the system could experience contention when users store or retrieve large numbers of messages at one time from the broadcast data set. By using user logs, you might decrease possible contention for the broadcast data set.

With RACF installed, your installation may be using security labels (SECLABELs). User logs can be set up to protect messages from being viewed by a user who is not logged on at the proper security label. The MSGPROTECT operand of the SEND PARMLIB parameter is used to indicate whether the individual user log data set is security protected from the user. For more information about the MSGPROTECT operand see *z/OS MVS Initialization and Tuning Reference*. For information about setting up security labels, see *z/OS Security Server RACF Security Administrator's Guide*.

If you use user logs, each user has an individual user log data set. The data set can be either a sequential data set or a member of a partitioned data set (PDS). SEND stores messages in the user log data set and LISTBC retrieves messages from the user log. When SEND stores the messages in the user logs, SEND truncates the trailing blanks. The command syntax for the SEND and LISTBC commands and the OPERATOR SEND subcommand is unaffected by user logs.

If you use user logs, OPERATOR SEND stores notices in the broadcast data set and LISTBC retrieves the notices from the broadcast data set. Notices are messages that the SEND subcommand of OPERATOR stores for all users on the system. The use of user logs does not affect the storing and retrieving of notices.

The following topics describe how to store messages in the broadcast data set or in user logs.

Storing messages in the broadcast data set only

If you want to store all SEND messages in the broadcast data set, you do not have to perform any tasks. By default, SEND stores messages in the broadcast data set and LISTBC retrieves them from the broadcast data set.

Storing messages in separate user logs

Using user logs, each user has an individual user log data set. The user log data set can be either a sequential data set or a member of a partitioned data set.

To implement user logs, use the LOGNAME operand of the SEND PARMLIB parameter in IKJTSOxx. For LOGNAME, specify the qualifier(s) for the user log data set name. You can also specify a member name in parentheses. The other qualifier of each user log data set is the user's user ID.

Note: User log data set names should not conflict with system data set names.

The naming convention used for the user log data set depends on the MSGPROTECT operand setting:

- If the MSGPROTECT operand of the SEND PARMLIB parameter is set to OFF, the first-level qualifier of each user log data set is the user's user ID. Therefore, for each user, the user log has a data set name of `userid.LOGNAME`, where *userid* is the user's user ID, and LOGNAME is the qualifier you specify on the LOGNAME operand.
- If the MSGPROTECT operand is set to ON, the user's user ID is the last qualifier of each user log data set. Therefore, for each user, the user log has a data set name of `LOGNAME.userid`.

If you specify user logs as members of a PDS, make sure the logical record length (LRECL) of the PDSis adequate to prevent messages from being truncated. This is particularly important if you use the exits to append information to messages that are stored. However, the LRECL of the PDS must not exceed 150.

A user log requires a minimum of one track for each user. You can save space by defining user logs as members of a partitioned data set. However, TSO/E does not compress a PDS. If SEND needs to store a message, SEND rewrites the member in the PDS. If your users receive a considerable amount of mail, you may eventually run out of allocated space in the PDS. In this case, you may want to use sequential data sets for some of your user's user logs.

LISTBC allocates user logs using the following data set attributes:

RECFM
fixed-blocked

LRECL
150

BLKSIZE
1500

Primary tracks
1

Secondary tracks
2

Note: The above space attributes can be overridden using the USERLOGSIZE operand under the SEND statement in PARMLIB member IKJTSOxx. For more information, see *z/OS MVS Initialization and Tuning Reference*.

For members of a PDS, LISTBC uses a value of 20 for the number of directory blocks.

You can use the LISTBC pre-allocate exit to allocate your own user logs with attributes that are more suitable for your different users. For more information about the exits, see "Writing exits for the SEND, OPERATOR SEND, and LISTBC commands" on page 341.

For information about implementing user logs, see "Converting from using the broadcast data set to user logs."

You can also use individual user logs for some of your users and continue to use the broadcast data set as the message repository for other users. For more information, see "Storing messages in user logs and the broadcast data set" on page 341.

Converting from using the broadcast data set to user logs

To implement the use of user logs, first copy sample member IKJTSO00 from SYS1.SAMPLIB to SYS1.PARMLIB, if you have not already done so. You can create alternative members using the IKJTSOxx naming convention. In IKJTSOxx, modify the SEND PARMLIB parameter as follows:

LOGNAME
user log data set name

CHKBROD
ON

Note:

1. After you convert from using the broadcast data set to user logs, the return codes from the SEND command are different. For information about these return codes, see *z/OS TSO/E Command Reference*.
2. If your installation is implementing the use of user logs with security protection, see also "Security protected user logs" on page 340.

With these values, the following SEND and LISTBC processing occurs:

- If a target user has not yet issued the LISTBC command or logged on and specified MAIL, and SEND needs to store a message for the user, the user's user log has not yet been allocated. Therefore, SEND stores the message in the broadcast data set.
- When the target user either issues the LISTBC command or logs on and specifies MAIL, LISTBC determines that the user log has not been allocated. The LISTBC command processor allocates the user log. The user log data set name depends on the MSGPROTECT operand setting.

  For more information about MSGPROTECT, see *z/OS MVS Initialization and Tuning Reference*.

  LISTBC allocates both sequential data sets and members of a PDS with the following data set attributes:

  LRECL
  150

  BLKSIZE
  1500

  Primary tracks
  1

  Secondary tracks
  2

  Note: If MSGPROTECT is set to ON, see the data set attributes shown in "Security protected user logs" on page 340.

  For members of a PDS, LISTBC uses a value of 20 for the number of directory blocks.

- LISTBC then checks the broadcast data set for any messages for the user because it has just created the user log data set.

  Note: If you modify the USEBROD operand to OFF, messages are not stored in the broadcast data set.
- The next time SEND needs to store a message for the target user, the user log data set exists. SEND stores the message in the user log.

After a period of time, you can decide to no longer check the broadcast data set for stored messages. At this point, your users' user logs should have been allocated and you can set CHKBROD to OFF. If you define new users to your system, have them either issue the LISTBC command or request MAIL when they first log on, to ensure a user log is created.

At this point, you can use the TSO SYNC command to clean up the broadcast data set if you want.

Note: This will delete all mails from the broadcast data set.

LISTBC allocates the user log with specific attributes. You may decide that the default attributes are unsuitable for some or all of your installation's users. You can use the LISTBC pre-allocate exit to allocate the user log data sets instead of having the LISTBC command processor allocate them. For more information about the pre-allocate exit, see “Writing exits for the SEND, OPERATOR SEND, and LISTBC commands” on page 341.

Security protected user logs: If RACF is installed, your installation is using security labels, and the following operands are modified, the security label of the message is checked to determine if the user is authorized to view the message. Modify the following in the SEND PARMLIB parameter in addition to the operands described in “Converting from using the broadcast data set to user logs” on page 339:

USEBROD
OFF

MSGPROTECT
ON

With these additional values, SEND and LISTBC processing is the same as described in “Converting from using the broadcast data set to user logs” on page 339 with the following exceptions:

- Messages are stored only in the individual user log, not in the broadcast data set.
- The sender's current security label is stored with the message.
- To allocate the user log data set, the target user must issue the LISTBC command or logon specifying MAIL.
- LISTBC allocates both sequential data sets and members of a PDS with the following data set attributes:

  LRECL
  232

  BLKSIZE
  2320

Primary tracks
1

Secondary tracks
2

- Before allowing the user to view the message, the user's security label and the message's security label are checked. If the user, attempting to view the message, is not logged on at an appropriate security label, then the user is not allowed to view the message. The message is placed back into the user log and can possibly be viewed when the user is logged on at the proper security label. If the user can never log on at the proper security label (not authorized for the security label of the message), the message is deleted.

For more information about security labels, see *z/OS Security Server RACF Security Administrator's Guide*.

Storing messages in user logs and the broadcast data set

You can have SEND store messages for some of your users in individual user logs and use the broadcast data set as the message repository for other users. First, to implement the use of user logs, follow the instructions described in "Converting from using the broadcast data set to user logs" on page 339. To use the broadcast data set for some of your users, write an initialization exit for the SEND, OPERATOR SEND, and LISTBC commands. The initialization exit can change the name of the data set that the command (subcommand) processors use to store and retrieve messages. For more information about the exits, see "Writing exits for the SEND, OPERATOR SEND, and LISTBC commands."

Writing exits for the SEND, OPERATOR SEND, and LISTBC commands

TSO/E provides many exits you can use to customize how the SEND and LISTBC commands and the OPERATOR SEND subcommand handle the sending, storing, and retrieving of messages. The exits for SEND and OPERATOR SEND are:

- Initialization - set up the environment for later exits
- Pre-display - perform processing before SEND displays the message to the target user
- Pre-save - perform processing before SEND opens the user log to store a message
- Failure - perform processing whenever SEND detects an I/O error on the user log
- Termination - perform clean-up processing prior to SEND completion

The exits for LISTBC are:

- Initialization - set up the environment for later exits
- Pre-list - perform processing before LISTBC associates (allocates) the user log data set name to a ddname and opens the user log data set
- Pre-allocate - create the user log data set
- Pre-read - perform processing before LISTBC reads a message
- Pre-display - perform processing before LISTBC displays the message to the user
- Failure - perform processing whenever LISTBC detects an I/O error on the user log
- Termination - perform clean-up processing prior to LISTBC completion

The SEND and LISTBC commands and the OPERATOR SEND subcommand do not share exits. For example, there are three initialization exits; one for SEND, one for LISTBC, and one for OPERATOR SEND.

You can use the different exits to customize SEND and LISTBC processing for your users. If you use only the broadcast data set to store messages, you can use only the initialization and termination exits to customize SEND and LISTBC processing. If you use user logs to store messages, you can use the other exits in addition to the initialization and termination exits to customize the use of user logs for individual users.

The following highlights some ways you can use the initialization exits:
- Restrict certain users from using a command
- Change the operands a user specifies on the command
- Change installation defaults for SEND and LISTBC processing that you either specified in IKJTSOxx or that are defaulted by the system if you did not update IKJTSOxx
- Ensure that users receive notices and mail when they log on to TSO/E

If you are using user logs, you can use the different initialization exits and the other SEND, OPERATOR SEND, and LISTBC exits to:
- Change the name of the data set SEND uses to store a message
- Perform your own file I/O to store messages in or retrieve messages from the user log
- Allocate a user log data set instead of having LISTBC allocate the user log
- Add information to a message that SEND stores
- Compress messages that SEND stores and decompress the message before LISTBC displays the message to the target user
- Format messages that are displayed to support special features of output devices

You can use the termination exits to perform clean-up or special processing prior to the command processor's completion. Depending on the processing the other exit routines perform, you may not need a termination exit for the command or subcommand. For more information about how you can use the exits, see “Possible uses” on page 362.

You use the SEND PARMLIB parameter in IKJTSOxx to specify installation defaults for SEND and LISTBC processing and to implement the use of user logs on your system. For more information about defining installation defaults and setting up user logs, see:
- “Defining installation defaults for SEND, OPERATOR SEND, and LISTBC” on page 334
- “Storing SEND messages” on page 337

When you IPL your system, a *SEND PARMLIB control block* (IKJEESCB) is created. The control block contains the installation defaults you specify in IKJTSOxx. If you do not define installation defaults in IKJTSOxx, a default control block is created that contains default values for SEND, OPERATOR SEND, and LISTBC. Each SEND, OPERATOR SEND, and LISTBC exit receives the address of a **copy** of the control block IKJEESCB.

In the control block, the EESCB_LOGNAME field contains the name of the data set you are using to store SEND messages. The name can be the broadcast data set or the name of a user log.

Note: The names referenced for the fields in the control block (for example, EESCB_LOGNAME) are obtained from the IKJEESCB DSECT mapping as provided in SYS1.AMODGEN.

The initialization and termination exits receive control regardless of the value in the EESCB_LOGNAME field. However, SEND, LISTBC, and OPERATOR SEND invoke the other exits only if EESCB_LOGNAME is the name of a user log.

The following topics describe when each SEND, OPERATOR SEND, and LISTBC exit receives control. For more information about how the exits' return codes affect SEND and LISTBC processing, see “Return specifications” on page 358.

Overview of when the SEND and OPERATOR SEND exits receive control

If a user issues the SEND command to send a message to only the console or the operator, SEND invokes only the initialization and termination exits. It does not invoke the SEND pre-display, pre-save, or failure exits.

- The **initialization exit** receives control before SEND invokes the parse service routine to parse the command. The exit gets control if you are using user logs or the broadcast data set to store messages, that is, the value of EESCB_LOGNAME is either * (indicating the name of the current broadcast data set) or a user log name.

  If SEND cannot display or store the message after the initialization exit returns control, SEND invokes the termination exit and then completes processing. If the initialization exit returns with a return code of 12 or 16, SEND does not display or store the message. SEND invokes the termination exit and then completes processing.
- If EESCB_LOGNAME in the copy of the control block is a user log name, the **pre-display exit** receives control. The exit receives control for each target user, unless the SEND command contains the SAVE operand. In this case, SEND does not invoke the pre-display exit.

  The exit receives control for each target user before SEND determines whether it can display the message to the target user. If the ALL operand is specified on the SEND subcommand of the OPERATOR command, the pre-display exit gets control only once. It does not get control for each target user.
- The **pre-save exit** receives control if both of the following are true:
  – EESCB_LOGNAME in the copy of the control block is a user log name
  – SEND will store the message for the target user(s).

  The exit receives control for each target user before SEND opens the user log to store the message. If the ALL operand is specified on the SEND subcommand of the OPERATOR command, the pre-save exit gets control only once. It does not get control for each target user.
- The **failure exit** receives control if EESCB_LOGNAME in the copy of the control block is a user log name. The exit receives control whenever SEND detects an I/O error on the user log when it is opening or closing the data set, or storing a message.
- The **termination exit** receives control just before SEND completes processing. The exit gets control if you are using user logs or the broadcast data set to store

messages, that is, the value of EESCB_LOGNAME is either * (indicating the name of the current broadcast data set) or a user log name.

Overview of when the LISTBC exits receive control

The LISTBC exits receive control when users issue the LISTBC command and when users log on to TSO/E (during logon, LISTBC processes mail and notices).

- The **initialization exit** receives control before LISTBC invokes the parse service routine to parse the command. The exit gets control if you are using user logs or the broadcast data set to store messages, that is, the value of EESCB_LOGNAME is either * (indicating the name of the current broadcast data set) or a user log name.

  When a user logs on, the initialization exit receives control before LISTBC opens the broadcast data set or the user log to retrieve messages.

  After the initialization exit returns control to LISTBC, LISTBC determines whether it should process notices. If so, LISTBC retrieves any notices from the broadcast data set.

  LISTBC then determines whether it should process mail. If not, LISTBC invokes the termination exit and then completes processing. If so, LISTBC determines whether it retrieves mail from the broadcast data set or the user log. The LISTBC command processor then continues processing and invokes the appropriate exits as described below.
- The **pre-list exit** receives control only if EESCB_LOGNAME in the copy of the control block is a user log name. The exit receives control before LISTBC associates (allocates) the user log data set name to a ddname and opens the user log data set.
- The **pre-allocate exit** receives control if both of the following are true:
  - EESCB_LOGNAME in the copy of the control block is a user log name
  - The user's user log data set does not yet exist

  LISTBC tries to associate (allocate) the user log data set name to a ddname. If the allocation fails because the user log does not exist, LISTBC invokes the pre-allocate exit. If the allocation is successful, the user log exists and the pre-allocate exit does not receive control.
- The **pre-read exit** receives control only if EESCB_LOGNAME in the copy of the control block is a user log name. The exit receives control:
  - After LISTBC opens the user log data set
  - Before LISTBC reads a message

  The pre-read exit gets control for each message in the user log.
- The **pre-display exit** receives control only if EESCB_LOGNAME in the copy of the control block is a user log name. The exit receives control before LISTBC displays the message to the user.

  The pre-display exit gets control for each message in the user log. After LISTBC reads and displays all of the messages, it closes the data set.
- The **failure exit** receives control only if EESCB_LOGNAME in the copy of the control block is a user log name. The exit receives control whenever LISTBC detects an I/O error on the user log when it is opening or closing the data set, or retrieving a message.
- The **termination exit** receives control just before LISTBC completes processing. The exit gets control if you are using user logs or the broadcast data set to store messages, that is, the value of EESCB_LOGNAME is either * (indicating the name of the current broadcast data set) or a user log name.

TSO/E-supplied exits

TSO/E does not provide default exit routines for any of the SEND, OPERATOR SEND, and LISTBC exits.

Entry specifications

The contents of the registers on entry for all of the SEND, OPERATOR SEND, and LISTBC exits are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions - overview

The following topics describe the parameters that each SEND, OPERATOR SEND, and LISTBC exit receives on entry. For more information about the standard exit parameter list and the parameter entries, see "TSO/E standard exit parameter list" on page 32.

Each exit receives the address of a **copy** of the SEND PARMLIB control block (IKJEESCB) as exit-dependent data. The copy of the control block contains the installation values you defined in SYS1.PARMLIB member IKJTSOxx or default values if you did not update IKJTSOxx. The exits can change the values in the copy of the control block to customize command processing for individual users. For more information about defining or changing installation defaults on a system-wide basis, see "Defining installation defaults for SEND, OPERATOR SEND, and LISTBC" on page 334.

Table 54 shows the format of the SEND PARMLIBcontrol block (IKJEESCB). You can use the IKJEESCB mapping macro, which is provided inSYS1.AMODGEN, to map the SEND PARMLIB control block. This macro has the following syntax.

```
[label]  IKJEESCB [ DSECT{=YES}]
                  [      { NO }]
```

Table 54. SEND PARMLIB control block format

Offset dec (Hex)	Number of bytes	Field name	Contents or meaning
0(0)	8	EESCB_IDENTIFIER	The control block identifier, C'IKJEESCB'
8(8)	1	EESCB_VERSION	The control block version number
9(9)	3	EESCB_RESERVED1	Reserved

Table 54. SEND PARMLIB control block format (continued)

Offset dec (Hex)	Number of bytes	Field name	Contents or meaning
12(C)	4	EESCB_FLAGS_1	The high-order byte of this field contains flags. For the meanings of these flags, see Table 55.
16(10)	44	EESCB_DATASET	User log data set name (LOGNAME)
60(3C)	8	EESCB_MEMBER	Data set member name
68(44)	36	EESCB_RESERVED2	Reserved

All character data in the control block is left-justified and padded with blanks. Fields containing flags and data set name information relate directly to the operands you specify on the SEND PARMLIB parameter in IKJTSOxx.

Table 55 describes the bit settings for EESCB_FLAGS_1. EESCB_CHKBROD, EESCB_USEBROD, and EESCB_MSGPROTECT are used only if you use user logs to store messages. They do not apply if you use only the broadcast data set to store messages.

Table 55. Meaning of bit settings in SEND PARMLIB control block

Name	Bit setting	Meaning	
EESCB_OPERSEND	`x... ....`	**0**	Authorized users of OPERATOR cannot use the SEND subcommand.
		1	Authorized users of OPERATOR can use the SEND subcommand.
EESCB_USERSEND	`.x.. ....`	**0**	Users cannot use the SEND command.
		1	Users can use the SEND command.
EESCB_SAVE	`..x. ....`	**0**	SEND cannot store messages in user logs.
		1	SEND can store messages in user logs.
EESCB_CHKBROD	`...x ....`	**0**	LISTBC checks only the user log for messages.
		1	LISTBC checks the broadcast data set and the user log for messages.
EESCB_USEBROD	`.... x...`	**0**	Mail is not stored in the broadcast data set, if the user does not have an individual user log. The user does not receive the message.
		1	Mail is stored in the broadcast data set, if the user does not have an individual user log.

Table 55. Meaning of bit settings in SEND PARMLIB control block (continued)

Name	Bit setting	Meaning	
EESCB_MSGPROTECT	`.... .x..`	**0**	User logs are not protected from the user and all mail can be viewed by the user. User log data set naming convention is 'userid.logname'.
		1	User logs are protected from the user and mail can be viewed only if the user is logged on at the proper security label. User log data set naming convention is 'logname.userid'.
EESCB_SYSPLEXSHR	`.... ..x.`	**0**	The broadcast data set is not shared exclusively between systems in the sysplex.
		1	The broadcast data set is shared exclusively between systems in the sysplex.
EESCB_SYSPLEXSHR_XCF	`.... ...x`	**0**	EESCB_SYSPLEXSHR was updated as a result of a PARMLIB UPDATE on this system.
		1	EESCB_SYSPLEXSHR was updated as a result of a PARMLIB UPDATE on another system.
Note: The following is the start of a new word.			
EESCB_OPERSEWAIT	`x... ....`	**0**	OPERATOR SEND without WAIT/NOWAIT specified on the command will be issued with NOWAIT.
		1	OPERATOR SEND without WAIT/NOWAIT specified on the command will be issued with WAIT.

Parameter descriptions for the SEND exits

All of the SEND exits receive the address of the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. The following topics describe the exit-dependent data that each SEND exit receives.

Initialization exit

Figure 50 on page 348 shows the exit-dependent data that the initialization exit receives beginning at offset +36 (decimal) in the parameter list. The parameter entry is described following the figure.

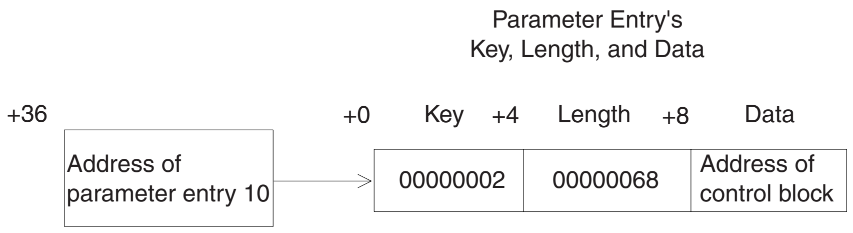


Figure 50. Exit-Dependent Data for the SEND Initialization exit

Control Block (Parameter Entry 10)
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

Pre-display exit

Figure 51 shows the exit-dependent data that the pre-display exit receives beginning at offset +36 (decimal) in the parameter list. Each parameter entry is described following the figure.

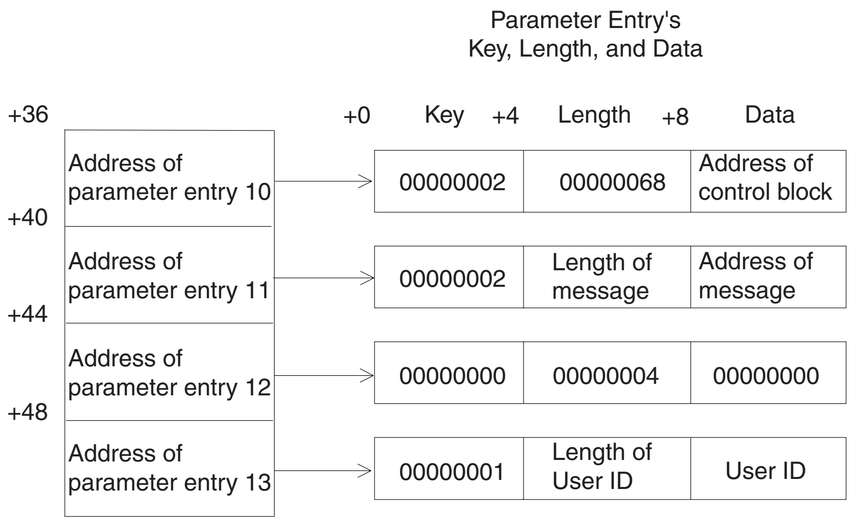


Figure 51. Exit-Dependent data for the SEND pre-display exit

Control Block (Parameter Entry 10)
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

SEND Message (Parameter Entry 11)
This parameter entry contains the address of the message that the user is sending.

New SEND Message (Parameter Entry 12)
Use this parameter entry to return a new SEND message to the SEND command processor.

User ID (Parameter Entry 13)
This parameter entry contains the user ID of the target user.

Pre-save exit

Figure 52 on page 349 shows the exit-dependent data that the pre-save exit receives beginning at offset +36 (decimal) in the parameter list. Each parameter entry is described following the figure.

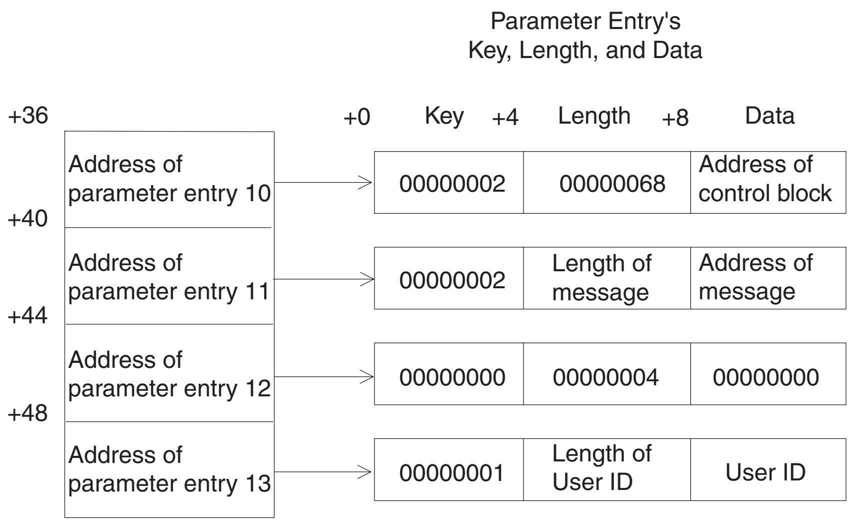


Figure 52. Exit-Dependent data for the SEND pre-save exit

Control Block (Parameter Entry 10)
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

SEND Message (Parameter Entry 11)
This parameter entry contains the address of the message that the user is sending.

New SEND Message (Parameter Entry 12)
Use this parameter entry to return a new SEND message to the SEND command processor.

User ID (Parameter Entry 13)
This parameter entry contains the user ID of the target user.

Failure exit

Figure 53 shows the exit-dependent data that the failure exit receives beginning at offset +36 (decimal) in the parameter list. Each parameter entry is described following the figure.

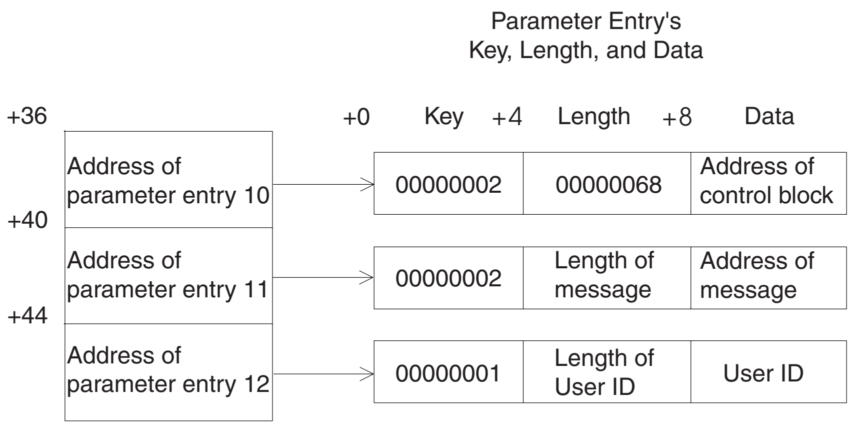


Figure 53. Exit-Dependent data for the SEND failure exit

Control Block (Parameter Entry 10)
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

SEND Message (Parameter Entry 11)
This parameter entry contains the address of the message that the user is sending.

User ID (Parameter Entry 12)
This parameter entry contains the user ID of the target user.

Termination exit

Figure 54 shows the exit-dependent data that the termination exit receives beginning at offset +36 (decimal) in the parameter list. The parameter entry is described following the figure.

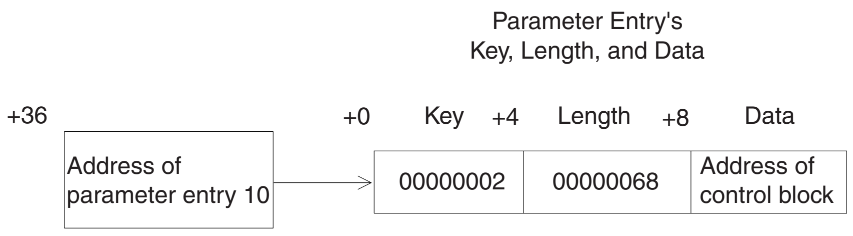


Figure 54. Exit-Dependent data for the SEND termination exit

Control Block (Parameter Entry 10)
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

Parameter descriptions for the OPERATOR SEND exits

All of the OPERATOR SEND exits receive the address of the standard exit parameter list. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32. However, no data is passed for the following parameter entries:
- Parameter entry 3 - user profile table (UPT)
- Parameter entry 4 - environmental control table (ECT)
- Parameter entry 5 - protected step control block (PSCB)

For these three parameter entries, the key, length, and data fields are:

Key X'00'

Length
X'04'

Data X'00'

Parameter entry 1 contains the address of the command buffer. Figure 2 on page 35 shows the format of the command buffer, however, the command buffer for the SEND subcommand of the OPERATOR command is different. As Figure 2 on page 35 illustrates, the command buffer contains a four-byte header field followed by a text field that contains the command and operands the user specified. The command buffer for the SEND subcommand of the OPERATOR commanddoes not contain the four-byte header field. It contains only the text field. In addition, the text field contains only the operands the user specified on the OPERATOR SEND subcommand. It does not contain the subcommand.

The following topics describe the exit-dependent data that each OPERATOR SEND exit receives.

Initialization exit

Figure 55 shows the exit-dependent data that the initialization exit receives beginning at offset +36 (decimal) in the parameter list. The parameter entry is described following the figure.

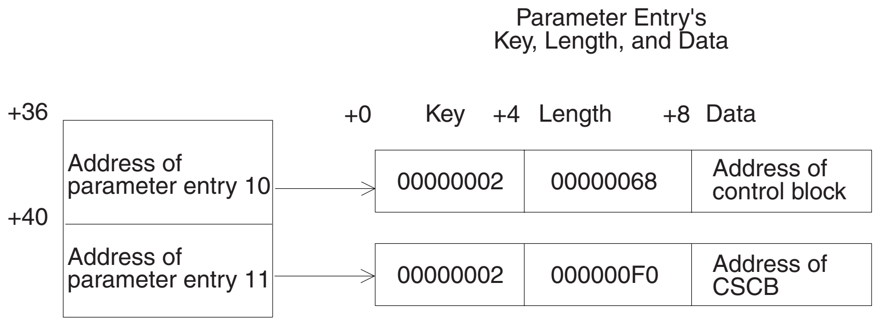


Figure 55. Exit-Dependent data for the OPERATOR SEND initialization exit

`Control Block (Parameter Entry 10)`
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

`CSCB (Parameter Entry 11)`
This parameter entry contains the address of the command scheduling control block (CSCB) to allow installations to identify the origin console.

Pre-display exit

Figure 56 shows the exit-dependent data that the pre-display exit receives beginning at offset +36 (decimal) in the parameter list. Each parameter entry is described following the figure.

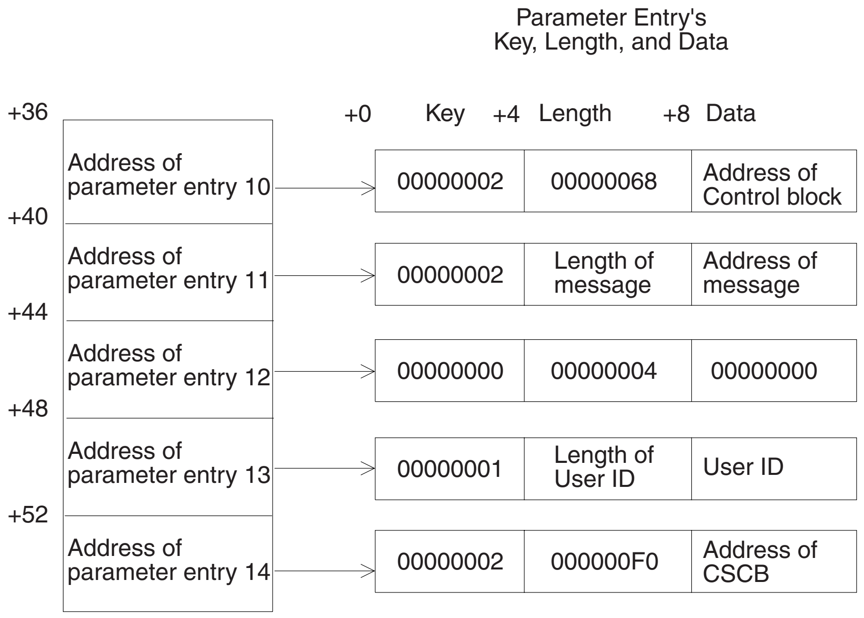


Figure 56. Exit-Dependent data for the OPERATOR SEND pre-display exit

Control Block (Parameter Entry 10)
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

SEND Message (Parameter Entry 11)
This parameter entry contains the address of the message that the user is sending.

New SEND Message (Parameter Entry 12)
Use this parameter entry to return a new SEND message to the SEND subcommand processor.

User ID (Parameter Entry 13)
This parameter entry contains the user ID of the target user. If the sending user specified the ALL operand on the SEND subcommand, this parameter entry contains blanks.

CSCB (Parameter Entry 14)
This parameter entry contains the address of the command scheduling control block (CSCB) to allow installations to identify the origin console.

Pre-save exit

Figure 57 shows the exit-dependent data that the pre-save exit receives beginning at offset +36 (decimal) in the parameter list. Each parameter entry is described following the figure.

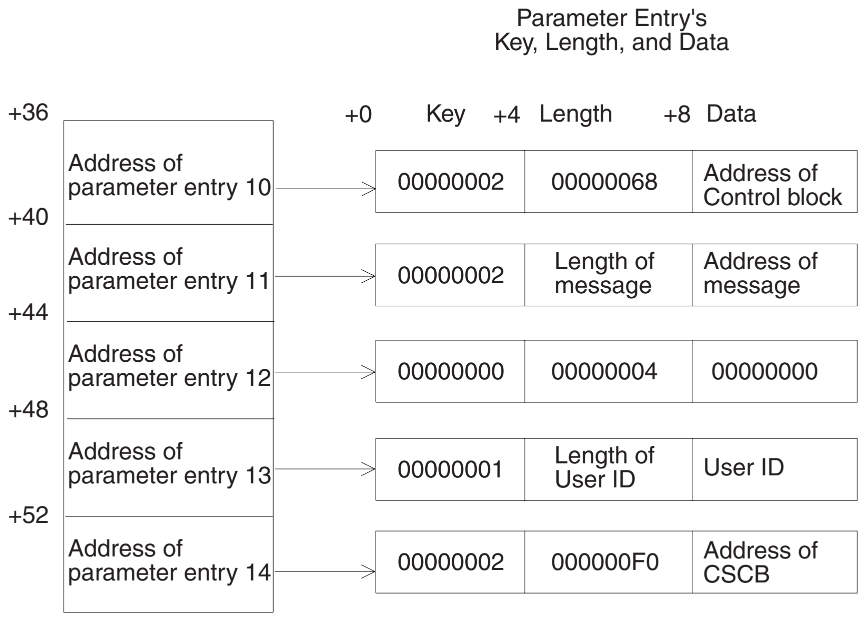


Figure 57. Exit-Dependent data for the OPERATOR SEND pre-save exit

Control Block (Parameter Entry 10)
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

SEND Message (Parameter Entry 11)
This parameter entry contains the address of the message that the user is sending.

New SEND Message (Parameter Entry 12)
Use this parameter entry to return a new SEND message to the SEND subcommand processor.

User ID (Parameter Entry 13)
This parameter entry contains the user ID of the target user. If the sending user specified the ALL operand on the SEND subcommand, this parameter entry contains blanks.

CSCB (Parameter Entry 14)
This parameter entry contains the address of the command scheduling control block (CSCB) to allow installations to identify the origin console.

Failure exit

Figure 58 shows the exit-dependent data that the failure exit receives beginning at offset +36 (decimal) in the parameter list. Each parameter entry is described following the figure.

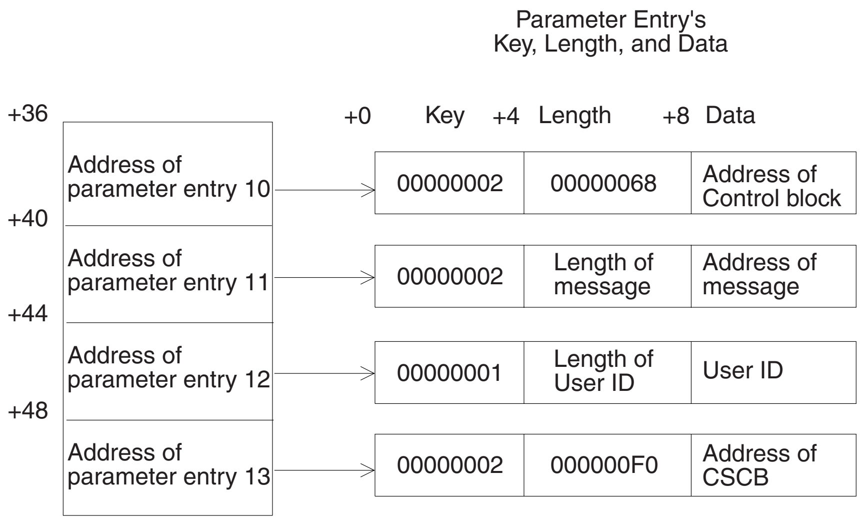


Figure 58. Exit-Dependent data for the OPERATOR SEND failure exit

Control Block (Parameter Entry 10)
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

SEND Message (Parameter Entry 11)
This parameter entry contains the address of the message that the user is sending.

User ID (Parameter Entry 12)
This parameter entry contains the user ID of the target user. If the sending user specified the ALL operand on the SEND subcommand, this parameter entry contains blanks.

CSCB (Parameter Entry 13)
This parameter entry contains the address of the command scheduling control block (CSCB) to allow installations to identify the origin console.

Termination exit

Figure 59 on page 354 shows the exit-dependent data that the termination exit receives beginning at offset +36 (decimal) in the parameter list. Each parameter entry is described following the figure.

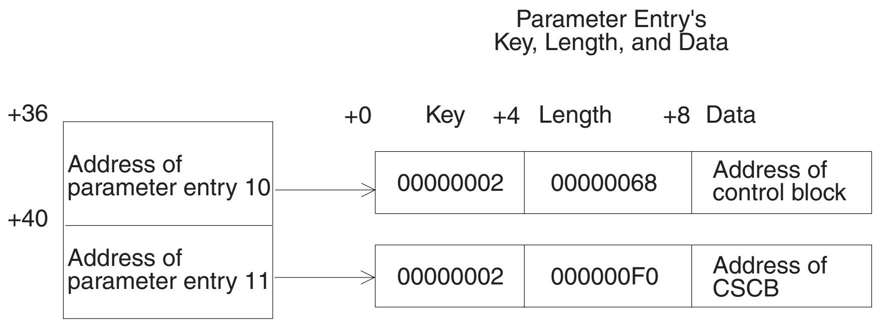


Figure 59. Exit-Dependent data for the OPERATOR SEND termination exit

Control Block (Parameter Entry 10)
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

CSCB (Parameter Entry 11)
This parameter entry contains the address of the command scheduling control block (CSCB) to allow installations to identify the origin console.

Parameter descriptions for the LISTBC exits

All of the LISTBC exits receive the address of the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. The following topics describe the exit-dependent data that each LISTBC exit receives.

Initialization exit

The LISTBC initialization exit receives different parameters depending on whether the exit receives control during logon processing or when a user issues the LISTBC command.

Parameters when initialization exit receives control during logon processing: When users log on, they can specify MAIL, NOTICES, or both, but they do not issue the LISTBC command. Therefore, when LISTBC invokes the initialization exit, there is no command buffer. Figure 60 shows the key, length, and data values for the command buffer (parameter entry 1) when the LISTBC initialization exit receives control during logon processing. The key of X'00' indicates that no data is passed in the parameter entry.

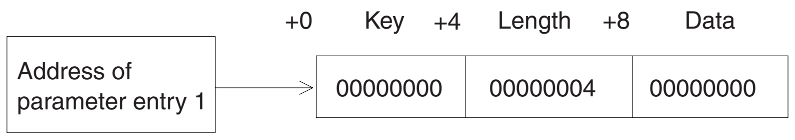


Figure 60. Command buffer parameter entry when LISTBC initialization exit gets control during logon

Figure 61 on page 355 shows the exit-dependent data that the LISTBC initialization exit receives when it gets control during logon processing. The exit-dependent data begins at offset +36 (decimal) in the parameter list. Each parameter entry is described following the figure.

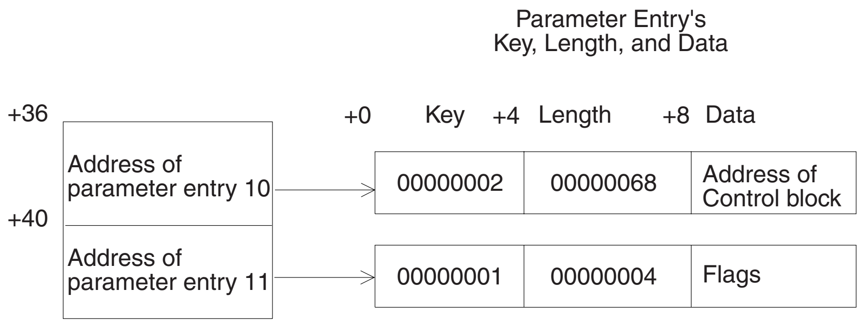


Figure 61. Exit-Dependent data for the LISTBC initialization exit during logon processing

`Control Block (Parameter Entry 10)`

This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

`Flags (Parameter Entry 11)`

When users log on to TSO/E, they can specify MAIL or NOMAIL and NOTICES or NONOTICES. This parameter entry contains *LISTBC indicator flags*. The flags indicate whether the user specified MAIL/NOMAIL and NOTICES/NONOTICES.

Although there is no command buffer you can use to modify processing, you can change the indicator flag bits to control and customize processing. For example, you can use the bits to ensure that users receive mail and notices when they log on. Table 56 shows the format of the LISTBC indicator flags, which consist of 32 bits.

Table 56. Format of the LISTBC indicator fags

Bit	Description
1	Indicates whether the user specified MAIL or NOMAIL. • 0 - the user specified NOMAIL • 1 - the user specified MAIL
2	Indicates whether the user specified NOTICES or NONOTICES. • 0 - the user specified NONOTICES • 1 - the user specified NOTICES
3-32	Reserved

Parameters when initialization exit receives control after user issues LISTBC: Figure 62 shows the exit-dependent data that the initialization exit receives beginning at offset +36 (decimal) in the parameter list. The parameter entry is described following the figure.

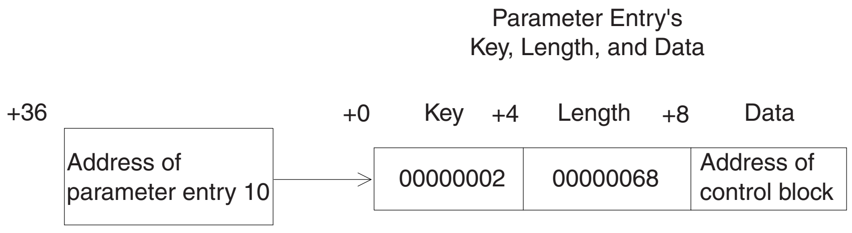


Figure 62. Exit-Dependent data for the LISTBC initialization exit when user issues LISTBC command

Control Block (Parameter Entry 10)
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

Pre-list exit

Figure 63 shows the exit-dependent data that the pre-list exit receives beginning at offset +36 (decimal) in the parameter list. The parameter entry is described following the figure.

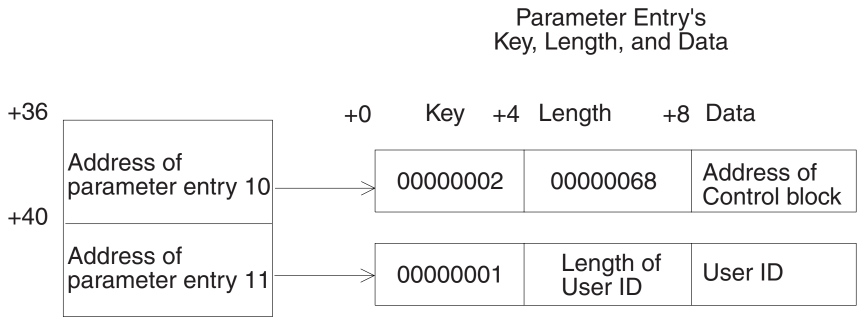


Figure 63. Exit-dependent data for the LISTBC pre-list exit

Control Block (Parameter Entry 10)
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

User ID (Parameter Entry 11)
Use this parameter entry to specify a user ID to be used as the high-level qualifier for the user log data set name.

Pre-allocate exit

Figure 64 shows the exit-dependent data that the pre-allocate exit receives beginning at offset +36 (decimal) in the parameter list. The parameter entry is described following the figure.

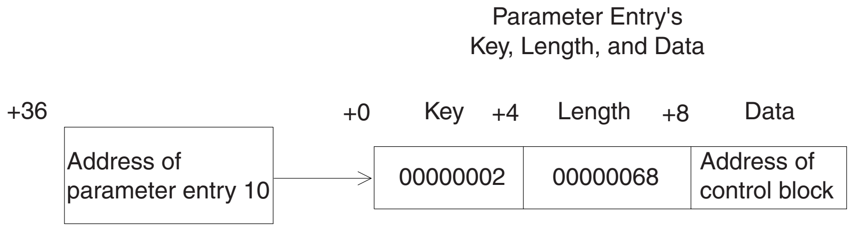


Figure 64. Exit-dependent data for the LISTBC pre-allocate exit

Control Block (Parameter Entry 10)
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

Pre-read exit

Figure 65 on page 357 shows the exit-dependent data that the pre-read exit receives beginning at offset +36 (decimal) in the parameter list. Each parameter entry is described following the figure.

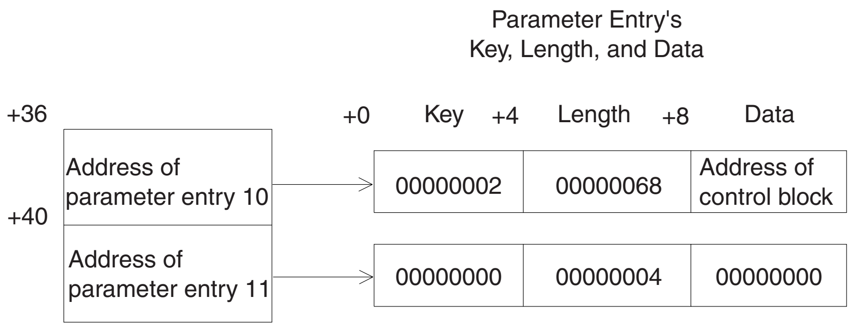


Figure 65. Exit-dependent data for the LISTBC pre-read exit

`Control Block (Parameter Entry 10)`
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

`LISTBC Message Buffer (Parameter Entry 11)`
Use this parameter entry with a return code of 4 to return a message to the LISTBC command processor. A return code of 4 indicates to LISTBC that it should not perform any file I/O and should use the message returned in the message buffer.

Pre-display exit

Figure 66 shows the exit-dependent data that the pre-display exit receives beginning at offset +36 (decimal) in the parameter list. Each parameter entry is described following the figure.

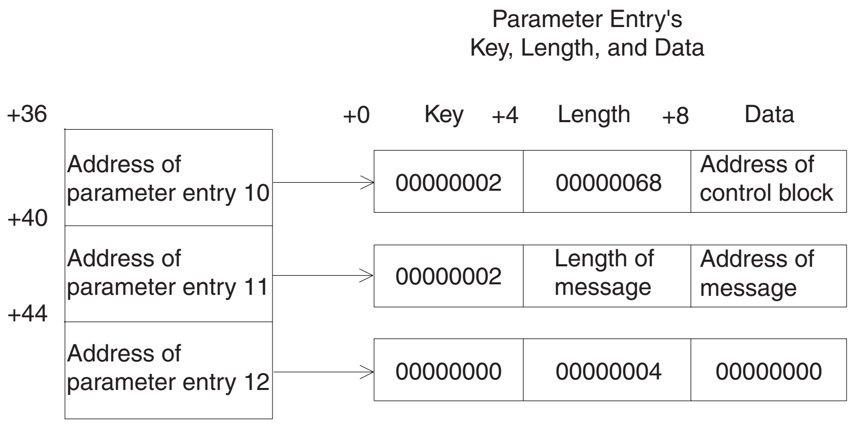


Figure 66. Exit-dependent data for the LISTBC pre-display exit

`Control Block (Parameter Entry 10)`
This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

`LISTBC Message (Parameter Entry 11)`
This parameter entry contains the address of the message that the LISTBC command processor retrieved from the user log.

`New LISTBC Message (Parameter Entry 12)`
Use this parameter entry to return a new message to the LISTBC command processor.

Failure exit

Figure 67 shows the exit-dependent data that the failure exit receives beginning at offset +36 (decimal) in the parameter list. The parameter entry is described following the figure.

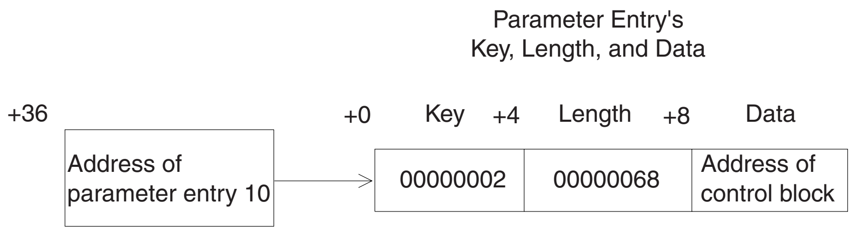


Figure 67. Exit-dependent data for the LISTBC failure exit

`Control Block (Parameter Entry 10)`
: This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

Termination exit

Figure 68 shows the exit-dependent data that the termination exit receives beginning at offset +36 (decimal) in the parameter list. The parameter entry is described following the figure.

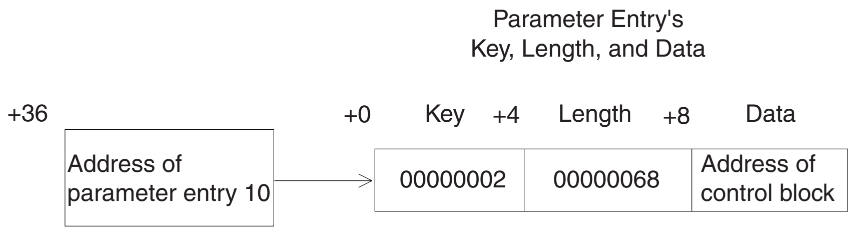


Figure 68. Exit-dependent data for the LISTBC termination exit

`Control Block (Parameter Entry 10)`
: This parameter entry contains the address of the copy of the SEND PARMLIB control block. For a description of the control block, see Table 54 on page 345.

Return specifications

The contents of the registers on return for all of the SEND, OPERATOR SEND, and LISTBC exits must be:

Registers 0–14
: Same as on entry

Register 15
: Return code

Return codes for the SEND exits

Table 57 shows the standard return codes that all of the SEND exits support.

Table 57. Standard return codes that All SEND exits support

Return code (decimal)	Description
0	Exit processing was successful. SEND processing continues.

Table 57. Standard return codes that All SEND exits support (continued)

Return code (decimal)	Description
12	Exit processing was unsuccessful. SEND issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the SEND command processor. For more information about the reason code, see “Exit reason code” on page 38.
16	Exit processing was unsuccessful. SEND terminates processing. The SEND command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display an informational message to the user, for example, using PUTLINE.

In addition to the standard return codes, several SEND exits support additional return codes. Table 58 lists these exits and return codes.

Table 58. Additional return codes for SEND exits

Exit	Return code (decimal)	Description
Pre-display	4	Do not display the SEND message.
Pre-save	4	Do not store the SEND message.

If your SEND exits return an unsupported return code, the SEND command processor terminates without displaying a message to the user.

Return codes for the OPERATOR SEND exits

Table 59 shows the standard return codes that all of the OPERATOR SEND exits support.

Table 59. Standard return codes that all OPERATOR SEND exits support

Return code (decimal)	Description
0	Exit processing was successful. OPERATOR SEND processing continues.
12	Exit processing was unsuccessful. OPERATOR SEND issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the OPERATOR SEND subcommand processor. For more information about the reason code, see “Exit reason code” on page 38.
16	Exit processing was unsuccessful. OPERATOR SEND terminates processing without displaying a message to the user.

In addition to the standard return codes, several OPERATOR SEND exits support additional return codes. Table 60 on page 360 lists these exits and return codes.

Table 60. Additional return codes for OPERATOR SEND exits

Exit	Return code (decimal)	Description
Pre-display	4	Do not display the SEND message.
Pre-save	4	Do not store the SEND message.

If your OPERATOR SEND exits return an unsupported return code, the SEND subcommand processor terminates without displaying a message to the user.

Return codes for the LISTBC exits

Table 61 shows the standard return codes that all of the LISTBC exits support.

Table 61. Standard return codes that all LISTBC exits support

Return code (decimal)	Description
0	Exit processing was successful. LISTBC processing continues.
12	Exit processing was unsuccessful. LISTBC issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the LISTBC command processor. For more information about the reason code, see "Exit reason code" on page 38.
16	Exit processing was unsuccessful. LISTBC terminates processing. The LISTBC command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display an informational message to the user, for example, using PUTLINE.

In addition to the standard return codes, several LISTBC exits support additional return codes. Table 62 lists these exits and return codes.

Table 62. Additional return codes for LISTBC exits

Exit	Return code (decimal)	Description
Pre-allocate	4	Do not allocate the user log data set.
Pre-read	4	Do not perform any file I/O. The exit returns the message to the LISTBC command processor in the message buffer.
	8	The end-of-file (EOF) has been reached.
Pre-display	4	Do not display the message that LISTBC retrieved.

If your LISTBC exits return an unsupported return code, the LISTBC command processor terminates without displaying a message to the user.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return.

The SEND, OPERATOR SEND, and LISTBC exits must be reentrant, refreshable, and reusable. The SEND and LISTBC exits are APF-authorized. The OPERATOR SEND exits are not APF-authorized, however, they execute in supervisor state with a key of 0.

The user SEND and the LISTBC exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

The need to write a termination exit depends on:

- Whether the processing that the other exits perform require a termination exit to perform clean-up activities
- How you use the other exits to customize SEND, OPERATOR SEND, and LISTBC processing

If one of the exits obtains a system resource, you must write a termination exit to free the resource before the command or subcommand processor completes its processing.

Environment

OPERATOR SEND Exits

- State: Supervisor
- Key: 0
- AMODE(31), RMODE(ANY)
- Not APF-authorized

SEND and LISTBC Exits

- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY)
- APF-authorized

Restrictions and limitations

You can use the different initialization and termination exits whether or not you use user logs to store messages. However, the other exits are specifically designed to be used only if your installation uses user logs. For more information about when the exits receive control, see “Overview of when the SEND and OPERATOR SEND exits receive control” on page 343 and “Overview of when the LISTBC exits receive control” on page 344.

Installing the exits

You must name the exits as follows:

SEND Exits

Initialization
IKJEESX0

Pre-display
IKJEESX1

Pre-save
IKJEESX2

Failure
IKJEESX3

Termination
IKJEESX4

OPERATOR SEND Exits

Initialization
IEEVSNX0

Pre-display
IEEVSNX1

Pre-save
IEEVSNX2

Failure
IEEVSNX3

Termination
IEEVSNX4

LISTBC Exits

Initialization
IKJEESX5

Pre-display
IKJEESX6

Pre-list
IKJEESX7

Pre-read
IKJEESX8

Pre-allocate
IKJEESX9

Failure
IKJEESXA

Termination
IKJEESXB

Link-edit each exit as a separate load module. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The SEND and LISTBC exits can reside in:
- The link pack area (LPA)
- LNKLST

The OPERATOR SEND exits can reside in:
- The link pack area (LPA)
- LNKLST

For more information about using the LPA, LNKLST, or STEPLIB, see “Installing the standard-format exits” on page 39.

Possible uses

The SEND, OPERATOR SEND, and LISTBC exits give you flexibility in customizing how your users send, store, and retrieve messages. You can use the various initialization and termination exits even if you store messages in and retrieve messages from the broadcast data set. If you use separate user logs, all of the exits are useful for customizing the use of these logs.

The following topics provide an overview of how you can use each of the exits and give some examples of using the exits to perform specific tasks.

Overview of using each exit

The different kinds of processing you can perform in each exit are described below.

SEND, OPERATOR SEND, and LISTBC initialization exits: You can use the various initialization exits even if you do not use user logs. If you use user logs, you use the initialization exits to initialize the environment for the other SEND, OPERATOR SEND, and LISTBC exits.

You can use an initialization exit to:

- Restrict certain users from using a command
- Change the operands a user specifies on the command
- Restrict certain users from storing messages
- Restrict users from sending messages to certain users
- Ensure that users receive NOTICES and MAIL when they log on to TSO/E
- Prevent users from receiving NOTICES, MAIL, or both when they log on to TSO/E
- Change the installation defaults to customize SEND, OPERATOR SEND, and LISTBC processing for individual users

You can use the initialization exits to change the installation defaults in the copy of the SEND PARMLIB control block. By doing this, you can customize the use of user logs and the broadcast data set to store messages.

SEND and OPERATOR SEND pre-display exits: The pre-display exit receives control before SEND displays the message to the target user. You can use a pre-display exit to format the message. For example, the exit can:

- Add diagnostic information to the message, such as the name of an ISPF panel
- Reformat the message to provide support for special features of certain output devices, such as graphics and recording attachments

The exit must not change the SEND message it receives (parameter entry 11). However, the exit can return a new SEND message to the SEND command (subcommand) processor. To return a new SEND message, the exit can:

- Obtain storage for a new SEND message
- Build the new SEND message
- Update the key, length, and data fields for the new SEND message (parameter entry 12) as follows:

  Key X'02'

  Length
  the length of the new SEND message

  Data the address of the new SEND message
- Set a return code of 0 and return to the SEND command (subcommand) processor.

The exit can display the message itself and set a return code of 4, which prevents the SEND command processor from displaying the message. It can also change the target user IDs so that SEND displays messages only for certain users.

Note: This exit does not change the target user IDs from one user to *ALL* users or from *ALL* users to one user. To change the target user IDs from *ALL* users or to *ALL* users, see the SEND subcommand of OPERATOR in *z/OS TSO/E System Programming Command Reference*.

SEND and OPERATOR SEND pre-save exits: The pre-save exit receives control before SEND stores the message in the user log. You can use the exit to change the name of the data set in which SEND stores the message by changing the EESCB_LOGNAME in the copy of the control block.

You can prevent SEND from storing the message by setting a return code of 4. For example, you may not want certain users to store messages. You may also want to perform your own file I/O. The exit can open the user log, store the message, close the user log, and then return control to SEND with a return code of 4.

You can use the SEND pre-save exits with the LISTBC pre-read exit to process the message. The SEND pre-save exit can change the message or add information to it and then the LISTBC pre-read exit can decode the added information. For example, the SEND pre-save exit can add sequence numbers to messages that SEND stores. The LISTBC pre-read exit can then sort the messages sequentially for retrieval. The SEND pre-save exit can compress the message before SEND stores it. The LISTBC pre-read exit can decompress the message before the message is displayed to the user.

You can also use the pre-save exit to:

- Change the target user IDs so that SEND stores messages only for certain users
- Return a different message to the SEND command (subcommand) processor using the “new SEND message” parameter entry (parameter entry 12)

LISTBC pre-list exit: The pre-list exit receives control before LISTBC associates (allocates) the user log data set name to a ddname and opens the user log. You can use the LISTBC pre-list exit to change the name of the user log.

You can also use the pre-list exit with the LISTBC pre-read exit to perform your own file I/O. The pre-list exit can open the log data set.

LISTBC pre-allocate exit: Use the pre-allocate exit to allocate the user log data set instead of having LISTBC perform the allocation. If LISTBC allocates the user log, it uses the following data set attributes for both sequential data sets and members of a PDS:

RECFM	fixed-blocked
LRECL	150
BLKSIZE	1500
Primary tracks	1
Secondary tracks	2

If the MSGPROTECT operand is set to ON, the following data set attributes are used:

LRECL	232
BLKSIZE	2320
Primary tracks	1
Secondary tracks	2

For members of a PDS, LISTBC uses a value of 20 for the number of directory blocks.

If you require different allocation values for the user log, use the pre-allocate exit to allocate the data set. SEND and LISTBC require that the record format specify fixed-length records. Note that the maximum LRECL you can define is 150. However, with MSGPROTECT set to ON the maximum LRECL you can define is 232.

LISTBC pre-read exit: The pre-read exit receives control after LISTBC opens the user log and before LISTBC reads the first message. You can use the exit to provide your own file I/O. The pre-read exit receives control for each message that exists in the data set for the user. When the exit receives control, it can:

- Read the message
- Return the message to LISTBC in the LISTBC message buffer (parameter entry 11). The exit updates the key, length, and data fields for the LISTBC message buffer as follows:

  Key X'02'

  Length
  the length of the message

  Data the address of the message
- Set a return code of 4 and return control to the LISTBC command processor

When the exit finishes reading all of the messages for the user, it can set a return code of 8 (EOF) and return control to LISTBC. The exit can also process a special departmental-type data set and then set a return code of 0 to allow LISTBC to continue processing the user log.

You can also use the LISTBC pre-read exit with the SEND and OPERATOR SEND pre-save exits to process the message. The SEND or OPERATOR SEND pre-save exit can change the message before SEND stores it. For example, the pre-save exit can:

- Add sequence numbers to the message
- Compress the message

The LISTBC pre-read exit can then:

- Sort the messages sequentially for retrieval
- Decompress the message before it is displayed to the user

LISTBC pre-display exit: The pre-display exit receives control before LISTBC displays the message to the user. You can use the exit to format the message before LISTBC displays it. For example, the exit can:

- Add information to the message
- Reformat the message to provide support for special features of certain output devices, such as graphics and recording attachments

The exit must not change the LISTBC message it receives (parameter entry 11). However, the exit can return a new message to the LISTBC command processor. To return a new message, the exit can:

- Obtain storage for a new message
- Build the new message
- Update the key, length, and data fields for the new LISTBC message (parameter entry 12) as follows:

  Key X'02'

Length
the length of the new LISTBC message

Data the address of the new LISTBC message

- Set a return code of 0 and return to the LISTBC command processor

The pre-display exit can prevent LISTBC from displaying the message by using a return code of 4. The exit can also display the message itself and then set a return code of 4 so that LISTBC does not display the message.

SEND, OPERATOR SEND, and LISTBC failure exits: The failure exits receive control whenever the command (subcommand) processor detects an I/O error on the user log data set when:
- Opening or closing the user log
- Reading a message from the user log
- Storing a message in the user log

You can use the failure exits to recover from an I/O error.

SEND, OPERATOR SEND, and LISTBC termination exits: The termination exits receive control just before the command (subcommand) completes processing. You can use the termination exits to perform any clean-up activities or special processing before command processing ends. If any of the other SEND, OPERATOR SEND, or LISTBC exits obtain storage for a system resource, you must provide a corresponding termination exit to free the storage.

You can also use a termination exit with the initialization exit to calculate the approximate amount of time it takes the command to complete its processing.

Examples of how to customize SEND, OPERATOR SEND, and LISTBC processing

The following describes some ways you can use the SEND, OPERATOR SEND, and LISTBC exits to customize command processing.

- Restrict certain users from using the SEND or LISTBC commands or the OPERATOR SEND subcommand

  The initialization exit can check the user ID and decide, based on your own criteria, to cancel the command. The exit can set a return code of 16 and return to the command or subcommand processor. It can also set the appropriate bit in the copy of the SEND PARMLIB control block to prevent an individual user from issuing the command. For example, you can use the USERSEND or OPERSEND bits in the control block to inactivate the SEND command or the OPERATOR SEND subcommand for that user.
- Change the operands the user specified on the command

  For various reasons, you may want to change the operands a user specified. For example, you may want to restrict certain users from using the SEND command to:
  - Send messages to certain users
  - Store messages

  For the OPERATOR SEND subcommand, you may want only certain users to list, delete, or store notices in the broadcast data set.

  To change the operands a user specified on the command, use the initialization exit. The exit can:
  - Scan the command buffer and decide, based on your own criteria, to change the command the user issued

- Obtain storage for a new command buffer
- Build the new command buffer
- Update the key, length, and data fields for the new command buffer as follows:

  Key X'02'

  Length
  the length of the new command buffer

  Data the address of the new command buffer
- Set a return code of 0 and return control to the command or subcommand processor

The exit must not change the command buffer it receives. It must build a new command buffer and return the address of the new command buffer to the invoking command. For more information about the command buffer, see "Command buffer" on page 35.

Note: The format of the command buffer for the SEND subcommand of the OPERATOR command is different from the command buffer for other command processors. For more information, see "Parameter descriptions for the OPERATOR SEND exits" on page 350.

You must also write a termination exit to free the storage the initialization exit obtains for the new command buffer.

- Ensure that users receive your installation's notices and their messages (mail) when they log on to TSO/E

  When users log on to TSO/E, LISTBC invokes the initialization exit. The exit receives *LISTBC indicator flags* that indicate whether the user specified NOTICES, NONOTICES, MAIL, or NOMAIL. The initialization exit can check the bit settings of the flags and change them so that LISTBC displays any notices and mail. For more information about the LISTBC indicator flags, see Table 56 on page 355.
- Allocate a user log data set

  If you are using user logs and a user either issues the LISTBC command or logs on and specifies MAIL, LISTBC determines whether the user's user log has been allocated. If the log data set has not been allocated, the LISTBC command processor allocates it. LISTBC allocates both sequential data sets and members of a PDS with the following data set attributes:

  LRECL
  150

  BLKSIZE
  1500

  Primary tracks
  1

  Secondary tracks
  2

  If the MSGPROTECT operand is set to ON, the following data set attributes are used:

  LRECL
  232

  BLKSIZE
  2320

Primary tracks
1

Secondary tracks
2

For members of a PDS, LISTBC uses a value of 20 for the number of directory blocks.

If the default attributes that LISTBC uses are unsuitable for some or all of your installation's users, use the LISTBC pre-allocate exit to allocate the user log data set. The pre-allocate exit can allocate the user log, set a return code of 4, and return to the LISTBC command processor. The return code of 4 indicates to LISTBC that it should not allocate the user log.

If you allocate a user log as a member of a PDS, consider the LRECL you use for the PDS. You must ensure that the LRECL is adequate to prevent messages from being truncated. This is particularly important if you use the exits to append information to messages that are stored. However, the LRECL of the PDS must not exceed 150. If the MSGPROTECT operand is ON, the LRECL of the PDS must not exceed 232.

Processing messages sent to the system operator

Your TSO/E execs and CLISTs can send messages to the system operator by issuing the SEND command.If you specify a message identifier at the beginning of the message text, you can use SYS1.PARMLIB(MPFLSTxx) to process these messages. To process a message sent by the SEND command, a plus sign (+) must be placed before the message identifier when you specify it in MPFLSTxx (for example, "+MSG001I"). For more information on controlling message processing through MPFLSTxx, see *z/OS MVS Initialization and Tuning Guide* and *z/OS MVS Initialization and Tuning Reference*.

Chapter 37. Customizing the TEST command

Users can invoke the TEST command to test unauthorized assembler programs. This includes testing APPC/MVS transaction programs written in assembler language. This chapter describes how you can customize TEST command processing for your users. For information about how to customize the testing of *authorized* assembler programs, see Chapter 38, "Customizing the TESTAUTH command," on page 383.

You can customize the TEST command in several ways:

- You can supply unauthorized installation-written subcommands of the TEST command, and unauthorized installation-written command processors to be invoked under TEST. You must define those subcommands and command processors to TEST, using CSECT IKJEGSCU or member IKJTSOxx in SYS1.PARMLIB.
- You can write exit routines to tailor the processing of the TEST command and its subcommands.

TSO/E provides the following exits for the TEST command and TEST subcommands:

- IKJEGMIE -- Initialization exit for the TEST command
- IKJEGMTE -- Termination exit for the TEST command
- IKJEGCIE -- Initialization exit for TEST subcommands
- IKJEGCTE -- Termination exit for TEST subcommands

Using the TEST exits, you can:

- Restrict certain users from using the TEST command or TEST subcommands.
- Change the operands a user specifies on the command or subcommands. For example, you can correct a user's errors or restrict users from using certain operands.

For more information about the TEST command exits, see "Writing exits for the TEST command" on page 372. For more information about the TEST subcommand exits, see "Writing exits for subcommands of the TEST command" on page 375.

Adding TEST subcommands and command processors

TSO/E supplies many subcommands for the TEST command, and many commands that can be invoked under TEST. If you need additional testing functions, you can write your own command processors or subcommand processors for use with TEST. You must first write the command or subcommand processor itself, and then define it to the TEST command.

z/OS TSO/E Programming Guide describes how to write a command processor or subcommand processor. To write a command processor to be invoked under TEST, follow the general procedures in that document; there are no special procedures. To write a subcommand of TEST, however, you must follow the special procedures described below in "Writing a subcommand of TEST" on page 370.

Writing a subcommand of TEST

The steps for writing a subcommand processor are listed in *z/OS TSO/E Programming Guide*. The steps for writing a TEST subcommand differ only in the first and last steps: accessing parameters and passing return codes.

Accessing parameters

Unlike other commands or subcommands, TEST subcommands do not access the command processor parameter list (CPPL). Instead, TEST subcommands begin by accessing the TEST communication table (TCOMTAB).

When a TEST subcommand receives control, register 9 points to the TCOMTAB. In the TCOMTAB, significant fields for TEST subcommands are:

- The INBUF field at decimal offset 120, which points to a subcommand buffer passed from TEST
- The TSTUPT field at decimal offset 92, which points to the UPT
- The TSTECT field at decimal offset 96, which points to the ECT
- The TPLPTR field at decimal offset 80, which points to the TEST parameter list (TPL). The TPL contains pointers to other information, including the PSCB.

Those fields of the TCOMTAB point to the same information that other subcommands would obtain from the CPPL.

Return codes from TEST subcommands

TEST subcommands must return control to the TEST command with register 15 set to a return code of zero. Any other return codes cause unpredictable results.

Defining a command or subcommand to TEST

After you have written a command processor to be invoked under TEST, or a subcommand of TEST, you must define it as such to the TEST command.

To define installation-written subcommands and command processors to TEST, you can use the CSECT IKJEGSCU or member IKJTSOxx of SYS1.PARMLIB. With IKJTSOxx, you avoid having to update, reassemble and link-edit the CSECT, and you can update or list the subcommands and command processors dynamically using the PARMLIB command.

The following topics describe how to use IKJTSOxx or CSECT IKJEGSCU.

Using SYS1.PARMLIB member IKJTSOxx

To define the TEST subcommands or command processors in IKJTSOxx, do the following:

- If you have not already done so, copy sample member IKJTSO00 from SYS1.SAMPLIB to SYS1.PARMLIB. You may have already copied IKJTSO00 to define other installation defaults.
- You can create alternative members in SYS1.PARMLIB using the IKJTSOxx naming convention.
- Edit the member in SYS1.PARMLIB to contain the appropriate TEST subcommands and command processors for your installation.

In IKJTSOxx, include the TEST statement with the following keyword parameters and their subfields:

TSOCMD
: specifies installation-written command processors that can be invoked under TEST at your installation. IBM-supplied command processors need not be included.

SUBCMD
: specifies installation-written TEST subcommands, with each subcommand represented by its name and entry point, separated by commas and enclosed in parentheses.

The following example shows how you can specify the TEST command information in SYS1.PARMLIB member IKJTSOxx.

```
TEST  TSOCMD(COMMAND1,        /* Installation command 1    */ +
             COMMAND2,        /* Installation command 2    */ +
             COMMAND3)        /* Installation command 3    */ +
                                                              +
      SUBCMD((SCM1,LOAD1),    /* Installation subcommand 1 */ +
            (SCM2,LOAD2),     /* Installation subcommand 2 */ +
            (SCM3,LOAD3))     /* Installation subcommand 3 */
```

The sample member IKJTSO00 in SYS1.SAMPLIB contains no TEST command information, because that information is unique for each installation. Use the example above as a guide for specifying TEST information for your installation.

You can use the PARMLIB command to dynamically list or update the installation-written subcommands and command processors defined to TEST. For information about using the PARMLIB command, see *z/OS TSO/E System Programming Command Reference*.

Using CSECT IKJEGSCU

If you choose not to use IKJTSOxx, you can use CSECT IKJEGSCU to define installation-written subcommands of TEST and command processors that can be invoked under TEST. To use IKJEGSCU, you must not have SYS1.PARMLIB member IKJTSOxx nor use the PARMLIB command. If you update IKJEGSCU, you must assemble it, link-edit it into load module TEST, and refresh the LLA before the updates take effect.

Figure 69 on page 372 shows the format of an entry in IKJEGSCU. IKJEGSCU allows one entry for each TEST subcommand and command processor that can be invoked under TEST. The entries must include:
- Length of the command or subcommand's name
- Name of the command or subcommand
- Length of an abbreviation for the name
- Abbreviation for the name
- Name of the load module
- ID

The format of an entry is shown below.

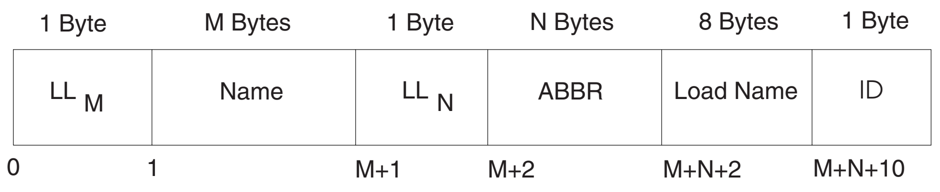


Figure 69. Format of entries in IKJEGSCU

The abbreviation for the subcommand name is optional. The other entries are required. Subcommands must have unique IDs in the range 34-127. command processors must have unique IDs in the range 145-255. Entries in the CSECT can be in any order, regardless of ID number.

Writing exits for the TEST command

Users issue the TEST command to test assembler programs in an unauthorized state. The TEST command lets the user trace a program's execution and diagnose possible errors. Users can issue the TEST command to debug active programs, or they can execute programs under the TEST command to test the programs before putting them into production. For more information about using TEST, see *z/OS TSO/E Programming Guide*. For information about TEST and its operands, see *z/OS TSO/E Command Reference*.

TSO/E provides an initialization exit and a termination exit for the TEST command. You can use the exits to customize TEST processing for your users. The initialization exit receives control before the TEST command invokes the parse service routine to parse the command. The termination exit receives control just before the TEST command processor completes processing.

If the initialization exit returns successfully to the TEST command processor and TEST processing itself abends, the TEST command processor invokes the termination exit before it terminates.

You can use the initialization exit to change the operands that users specify on the command or correct user errors when they issue the command. You can use the termination exit to perform clean-up or special processing prior to TEST command completion. Depending on the processing your initialization exit performs, you may not need a corresponding termination exit.

The following highlights some ways you can use the TEST exits. For more information about how you can use the exits, see “Possible uses” on page 375.

- Restrict certain users from using the TEST command
- Correct a user's errors on the TEST command
- Change the operands a user specifies on the command

TSO/E-supplied exits

TSO/E does not provide default exit routines for the TEST initialization and termination exits.

Entry specifications

For the TEST initialization and termination exits, the contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions for the initialization exit

The TEST initialization exit receives the address of the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. The initialization exit does not receive any exit-dependent data.

Parameter descriptions for the termination exit

The TEST termination exit receives the address of the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. The termination exit does not receive any exit-dependent data.

If the initialization exit returns a new command buffer or an exit-to-exit communication word to the TEST command processor, TEST passes the values of these parameter entries to the termination exit. For more information about the parameter entries, see "TSO/E standard exit parameter list" on page 32.

Return specifications

The contents of the registers on return for both TEST exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the initialization and termination exits

Table 63 shows the return codes that the TEST initialization and termination exits support.

Table 63. Return codes for the TEST initialization and termination exits

Return code (decimal)	Description
0	Exit processing was successful. TEST processing continues.
12	Exit processing was unsuccessful. TEST issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the TEST command processor. For more information about the exit reason code, see "Exit reason code" on page 38.

Table 63. Return codes for the TEST initialization and termination exits (continued)

Return code (decimal)	Description
16	Exit processing was unsuccessful. TEST terminates processing. The TEST command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the TEST command processor terminates without displaying a message to the user.

The termination exit receives control just before the completion of TEST processing. Therefore, the TEST command processor may have already successfully tested the unauthorized assembler program regardless of the return code the termination exit returns.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exits must be reentrant, refreshable, and reusable. The exits are not APF-authorized.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

In some cases, you may not need to write a termination exit. This depends on:

- Whether the processing that the initialization exit performs requires a termination exit to perform clean-up activities
- How you use the exits to customize TEST processing

If the initialization exit obtains a system resource, you must write a termination exit to free the resource. For example, if the initialization exit obtains storage to return a new command buffer to the TEST command processor, you must provide a termination exit to free the storage.

The initialization exit can change TEST operands using the command buffer. The exit checks the command buffer it receives and determines whether to change any operands. To change the operands, the exit must:

- Obtain storage for a new command buffer
- Build the new command buffer
- Update the key, length, and data fields for the new command buffer (parameter entry 2)
- Set return code 0 and return control to TEST

For more information about the parameter entries, see “TSO/E standard exit parameter list” on page 32. For more information about the format of the command buffer, see “Command buffer” on page 35.

Environment

- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY), ASCMODE(PRIMARY)
- Not APF-authorized

Installing the exits

You must name the exits as follows:

Initialization
IKJEGMIE

Termination
IKJEGMTE

Link-edit each exit as a separate load module. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see "Installing the standard-format exits" on page 39.

Possible uses

Some possible uses of the TEST exits are described below:
- Change the operands that the user specifies on the command

  You can use the initialization exit to change the operands that users specify on the TEST command. The initialization exit receives the address of the command buffer. It can change the operands the user specifies on the TEST command by using a new command buffer. For example, the initialization exit can scan the command buffer and correct any errors on the command.

To check the command buffer and change its contents, the initialization exit can:
- Scan the command buffer and decide, based on your own criteria, to change the command the user issued
- Obtain storage for a new command buffer
- Build the new command buffer
- Update the key, length, and data fields for the new command buffer as follows:

  Key X'02'

  Length
  the length of the new command buffer

  Data the address of the new command buffer
- Set a return code of 0 and return control to the TEST command processor

The exit must not change the command buffer it receives. It must create a new command buffer and return the address of the new command buffer to TEST.

For more information about the command buffer and the new command buffer, see "TSO/E standard exit parameter list" on page 32. For information about the format of the command buffer, see "Command buffer" on page 35.

Writing exits for subcommands of the TEST command

The TEST command includes a number of subcommands that users can issue while the TEST command is executing. For example, under the TEST command, users can issue the LIST subcommand to display the contents of a virtual storage area or registers, or the DELETE subcommand to delete a load module from virtual storage. For more information about using TEST with subcommands, see *z/OS TSO/E Programming Guide*. For information about TEST and its subcommands, see *z/OS TSO/E Command Reference*.

TSO/E provides an initialization exit and a termination exit for the subcommands of the TEST command. You can use the exits to customize TEST subcommand processing for your users. The initialization exit receives control before the TEST command invokes the subcommand. The termination exit receives control just before the TEST subcommand completes processing.

If the initialization exit returns successfully to the TEST subcommand and the TEST command processor itself abends, the TEST command processor invokes the subcommand termination exit before ending.

You can use the subcommand initialization exit to change the operands that users specify on the subcommand or correct user errors when they issue the subcommand. You can use the termination exit to perform clean-up or special processing prior to subcommand completion. Depending on the processing your subcommand initialization exit performs, you may not need a corresponding subcommand termination exit.

The following highlights some ways you can use the TEST subcommand exits. For more information about how you can use the exits, see “Possible uses” on page 380.

- Restrict certain users from using one or more TEST subcommands
- Correct a user's errors on a TEST subcommand
- Change the operands a user specifies on a subcommand
- Provide *pseudo-operands* that are equivalent to two or more operands of a TEST subcommand

TSO/E-supplied exits

TSO/E does not provide default exit routines for the TEST subcommand initialization and termination exits.

Entry specifications

For the TEST subcommand initialization and termination exits, the contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions for the initialization exit

The TEST subcommand initialization exit receives the address of the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32.

Figure 70 shows the exit-dependent data that the TEST subcommand initialization exit receives beginning at offset +36 (decimal) in the parameter list.

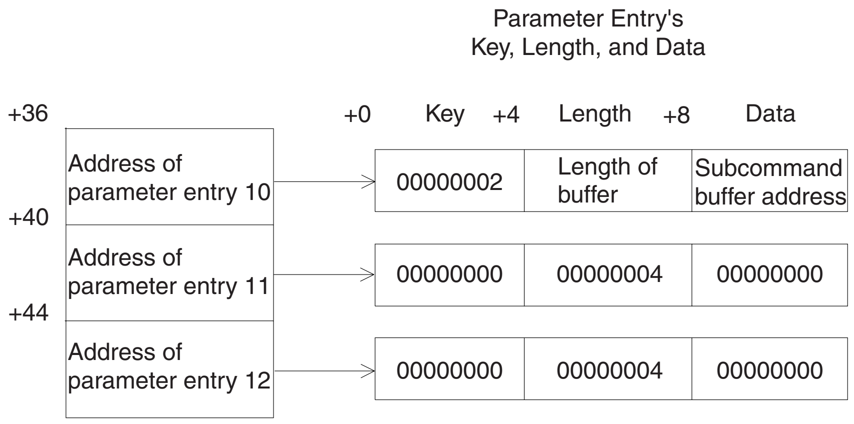


Figure 70. Exit-dependent data for the TEST subcommand initialization exit

Subcommand Buffer Address (Parameter Entry 10)
This parameter entry contains the address of the buffer containing the text of the entered subcommand and its keywords.

Alternate Buffer Address (Parameter Entry 11)
This parameter entry lets the exit return the address of an alternate buffer containing a substitute subcommand.

Subcommand Exit Word (Parameter Entry 12)
This parameter entry lets the subcommand initialization exit return a word of information (address or data) that is passed to the subcommand termination exit.

Parameter descriptions for the termination exit

The TEST subcommand termination exit receives the address of the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32.

Figure 71 on page 378 shows the exit-dependent data that the TEST subcommand termination exit receives beginning at offset +36 (decimal) in the parameter list.

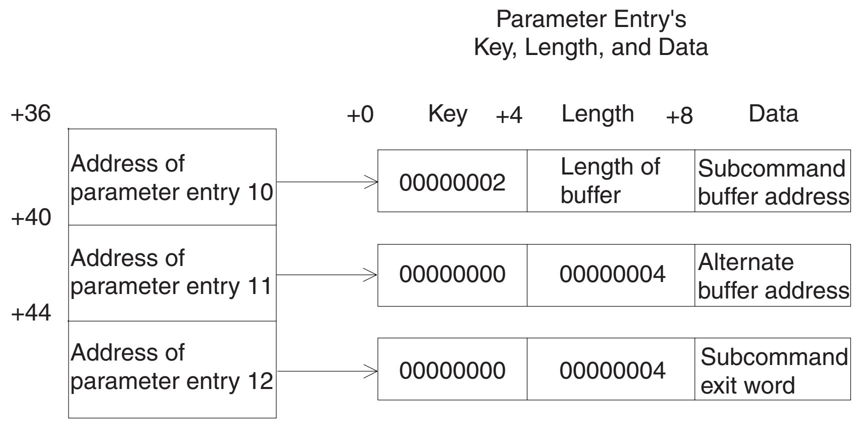


Figure 71. Exit-dependent data for the TEST subcommand termination exit

`Subcommand Buffer Address (Parameter Entry 10)`
This parameter entry contains the address of the buffer containing the text of the entered subcommand and its keywords.

`Alternate Buffer Address (Parameter Entry 11)`
This parameter entry contains the possible address of an alternate buffer set by the subcommand initialization exit.

`Subcommand Exit Word (Parameter Entry 12)`
This parameter entry contains a possible word of information (address or data) passed from the subcommand initialization exit.

Return specifications

The contents of the registers on return for both TEST subcommand exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the subcommand initialization exit

Table 64 shows the return codes that the TEST subcommand initialization exit supports.

Table 64. Return codes for the TEST subcommand initialization exit

Return code (decimal)	Description
0	Exit processing was successful. TEST processing continues.
4	Exit processing was unsuccessful. The subcommand terminates and a message with a reason code is issued.
8	Exit processing was unsuccessful. The subcommand terminates without a message.
12	Exit processing was unsuccessful. TEST issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the TEST command processor. For more information about the exit reason code, see “Exit reason code” on page 38.

Table 64. Return codes for the TEST subcommand initialization exit (continued)

Return code (decimal)	Description
16	Exit processing was unsuccessful. TEST terminates processing. The TEST command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the TEST command processor terminates without displaying a message to the user.

Return codes for the subcommand termination exit

Table 65 shows the return codes that the TEST subcommand termination exit supports.

Table 65. Return codes for the TEST subcommand termination exit

Return code (decimal)	Description
0	Exit processing was successful. TEST processing continues.
12	Exit processing was unsuccessful. TEST issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the TEST command processor. For more information about the exit reason code, see “Exit reason code” on page 38.
16	Exit processing was unsuccessful. TEST terminates processing. The TEST command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the TEST command processor terminates without displaying a message to the user.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exits must be reentrant, refreshable, and reusable. The exits are not APF-authorized.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

In some cases, you may not need to write a termination exit. This depends on:

- Whether the processing that the subcommand initialization exit performs requires a termination exit to perform clean-up activities
- How you use the exits to customize TEST subcommand processing

If the subcommand initialization exit obtains a system resource, you must write a subcommand termination exit to free the resource. For example, if the initialization exit obtains storage to return a new subcommand buffer to the TEST command processor, you must provide a subcommand termination exit to free the storage.

The initialization exit can change subcommand operands using the subcommand buffer. The exit checks the subcommand buffer it receives and determines whether to change any operands. To change the operands, the exit must:
- Obtain storage for a new subcommand buffer
- Build the new subcommand buffer
- Update the key, length, and data fields for the new subcommand buffer (parameter entry 11)
- Set return code 0 and return control to TEST

For more information about the parameter entries, see “TSO/E standard exit parameter list” on page 32. For more information about the format of the command buffer, see “Command buffer” on page 35.

Environment
- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY), ASCMODE(PRIMARY)
- Not APF-authorized

Installing the exits
You must name the TEST subcommand exits as follows:

Initialization
IKJEGCIE

Termination
IKJEGCTE

Link-edit each exit as a separate load module. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see “Installing the standard-format exits” on page 39.

Possible uses
Some possible uses of the TEST subcommand exits are described below:
- Change the operands that the user specifies on the subcommand

  You can use the initialization exit to change the operands that users specify on the TEST subcommand. The initialization exit receives the address of the subcommand buffer. It can change the operands the user specifies on the TEST subcommand by using a new subcommand buffer. For example, the initialization exit can scan the subcommand buffer and correct any errors on the subcommand.

  To check the subcommand buffer and change its contents, the initialization exit can:
  – Scan the subcommand buffer and decide, based on your own criteria, to change the command the user issued
  – Obtain storage for a new subcommand buffer
  – Build the new subcommand buffer
  – Update the key, length, and data fields for the new subcommand buffer as follows:

Key X'02'

Length
the length of the new subcommand buffer

Data the address of the new subcommand buffer

– Set a return code of 0 and return control to the TEST command processor.

The exit must not change the subcommand buffer it receives. It must create a new subcommand buffer and return the address of the new subcommand buffer to TEST.

For more information about the subcommand buffer and the new subcommand buffer, see “TSO/E standard exit parameter list” on page 32. For information about the format of the subcommand buffer, see “Command buffer” on page 35.

Chapter 38. Customizing the TESTAUTH command

Users can invoke the TESTAUTH command to test authorized assembler programs. This includes testing APPC/MVS transaction programs written in assembler language. This chapter describes how you can customize TESTAUTH command processing for your users. For information about how to customize the testing of *unauthorized* programs, see Chapter 37, "Customizing the TEST command," on page 369.

Before people at your installation can use the TESTAUTH command, you must add TESTAUTH to the table of authorized commands. Chapter 11, "Specifying authorized commands/programs, and commands not supported in the background," on page 151 describes how to maintain and update the table.

You should also limit individual users from using the TESTAUTH command. You can limit users in one of the following ways:
- Using TESTAUTH initialization exit routine IKJEGAUI. When a user issues TESTAUTH, IKJEGAUI can check the user-ID and issue a return code to let the user continue, to cancel the TESTAUTH command, or to invoke authority checking with RACF.
- Using RACF. You can use the RACF RDEFINE command to define TESTAUTH as a RACF resource belonging to the TSOAUTH RACF class. Then give selected users access to the TESTAUTH resource using the RACF PERMIT command. Note that users do NOT require a TSO segment in order to gain access to the TESTAUTH profile in the TESTAUTH class.

In addition to controlling access to the TESTAUTH command, you can customize the TESTAUTH command in several ways:
- You can write exits to tailor or monitor the processing of the TESTAUTH command or that of its subcommands.
- You can supply installation-written subcommands of the TESTAUTH command, and installation-written command processors to be invoked under TESTAUTH. You must define those subcommands and command processors to TESTAUTH, using member IKJTSOxx in SYS1.PARMLIB.

Writing Exits for the TESTAUTH Command

TSO/E provides the following exits for the TESTAUTH command and TESTAUTH subcommands:
- IKJEGAUI -- Initialization exit for the TESTAUTH command
- IKJEGAUT -- Termination exit for the TESTAUTH command
- IKJEGASI -- Initialization exit for TESTAUTH subcommands
- IKJEGAST -- Termination exit for TESTAUTH subcommands

Using the TESTAUTH exits, you can:
- Restrict certain users from using the TESTAUTH command or TESTAUTH subcommands.
- Change the operands a user specifies on the command or subcommands. For example, you can correct a user's errors or restrict users from using certain operands.

For more information about the TESTAUTH command exits, see "Writing exits for the TESTAUTH command" on page 384. For more information about the TESTAUTH subcommand exits, see “Writing exits for subcommands of the TESTAUTH command” on page 388.

Adding TESTAUTH subcommands and command processors

TSO/E supplies many subcommands for the TESTAUTH command, and many commands that can be invoked under TESTAUTH. If you need additional testing functions, you can write your own command processors or subcommand processors for use with TESTAUTH. You must first write the command or subcommand processor itself, and then define it to the TESTAUTH command. To do so, follow the same procedures described in “Adding TEST subcommands and command processors” on page 369. When you define a command or subcommand to TEST in SYS1.PARMLIB member IKJTSOxx or in CSECT IKJEGSCU, the subcommand or command is available to both TEST and TESTAUTH.

If you want to restrict a subcommand to TEST or TESTAUTH, you can use the subcommand's initialization exit to check its environment and cancel the subcommand if invoked under the wrong command.

Writing exits for the TESTAUTH command

Users issue the TESTAUTH command to test assembler programs that run in an authorized state. The TESTAUTH command lets the user trace a program's execution and diagnose possible errors. Users can execute programs under the TESTAUTH command to test the programs before putting them into production. For more information about using TESTAUTH, see *z/OS TSO/E Programming Guide*. For information about TESTAUTH and its operands, see *z/OS TSO/E System Programming Command Reference*.

TSO/E provides an initialization exit and a termination exit for the TESTAUTH command. You can use the exits to customize TESTAUTH processing for your users. The initialization exit receives control before the TESTAUTH command invokes the parse service routine to parse the command. The termination exit receives control just before the TESTAUTH command processor completes processing.

If the initialization exit returns successfully to the TESTAUTH command processor and TESTAUTH processing itself abends, the TESTAUTH command processor invokes the termination exit before it terminates.

You can use the initialization exit to change the operands that users specify on the command or correct user errors when they issue the command. You can use the termination exit to perform clean-up or special processing prior to TESTAUTH command completion. Depending on the processing your initialization exit performs, you may not need a corresponding termination exit.

The following highlights some ways you can use the TESTAUTH exits. For more information about how you can use the exits, see “Possible uses” on page 388.
- Restrict certain users from using the TESTAUTH command
- Correct a user's errors on the TESTAUTH command
- Change the operands a user specifies on the command
- Provide *pseudo-operands* that are equivalent to two or more TESTAUTH operands

TSO/E-supplied exits

TSO/E does not provide default exit routines for the TESTAUTH initialization and termination exits.

Entry specifications

For the TESTAUTH initialization and termination exits, the contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions for the initialization exit

The TESTAUTH initialization exit receives the address of the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32.

Figure 72 shows the exit-dependent data that the initialization exit receives beginning at offset +36 (decimal) in the parameter list.

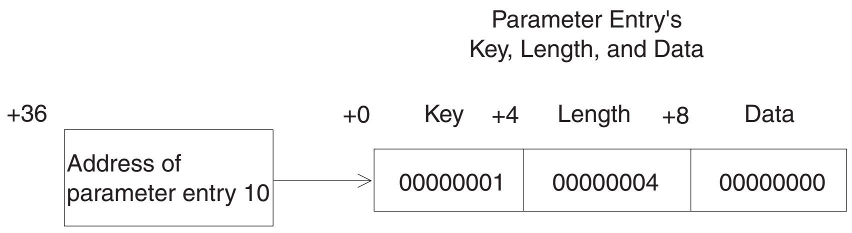


Figure 72. Exit-dependent data for the TESTAUTH initialization exit

`Authority Value (Parameter Entry 10)`
This parameter (X'00' on entry) lets the exit return a value to indicate the user's authority to use the TESTAUTH command. On return from the exit, the possible values are:

Value (hex)	Meaning
0	Use RACF to verify the user's authority
4	The user is authorized to use TESTAUTH
8	The user is not authorized to use TESTAUTH

Parameter descriptions for the termination exit

The TESTAUTH termination exit receives the address of the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. The termination exit does not receive any exit-dependent data.

If the initialization exit returns a new command buffer or an exit-to-exit communication word to the TESTAUTH command processor, TESTAUTH passes the values of these parameter entries to the termination exit. For more information about the parameter entries, see "TSO/E standard exit parameter list" on page 32.

Return specifications

The contents of the registers on return for both TESTAUTH exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the initialization and termination exits

Table 66 shows the return codes that the TESTAUTH initialization and termination exits support.

Table 66. Return codes for the TESTAUTH initialization and termination exits

Return code (decimal)	Description
0	Exit processing was successful. TESTAUTH processing continues.
12	Exit processing was unsuccessful. TESTAUTH issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the TESTAUTH command processor. For more information about the exit reason code, see "Exit reason code" on page 38.
16	Exit processing was unsuccessful. TESTAUTH terminates processing. The TESTAUTH command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the TESTAUTH command processor terminates without displaying a message to the user.

The termination exit receives control just before the completion of TESTAUTH processing. Therefore, the TESTAUTH command processor may have already successfully tested the authorized assembler program regardless of the return code the termination exit returns.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exit must be reentrant, refreshable, reusable, and reside in an APF-authorized library. It is inadvisable for the exit to have APF authorization unless it is designed also to be called as the entry point for a job step program.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

In some cases, you may not need to write a termination exit. This depends on:
- Whether the processing that the initialization exit performs requires a termination exit to perform clean-up activities
- How you use the exits to customize TESTAUTH processing

If the initialization exit obtains a system resource, you must write a termination exit to free the resource. For example, if the initialization exit obtains storage to return a new command buffer to the TESTAUTH command processor, you must provide a termination exit to free the storage.

The initialization exit can change TESTAUTH operands using the command buffer. The exit checks the command buffer it receives and determines whether to change any operands. To change the operands, the exit must:
- Obtain storage for a new command buffer
- Build the new command buffer
- Update the key, length, and data fields for the new command buffer (parameter entry 2)
- Set return code 0 and return control to TESTAUTH

For more information about the parameter entries, see “TSO/E standard exit parameter list” on page 32. For more information about the format of the command buffer, see “Command buffer” on page 35.

Environment
- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY), ASCMODE(PRIMARY)
- APF-authorized

Restrictions and limitations
The TESTAUTH exits must reside in an APF-authorized library.

Installing the exits
You must name the exits as follows:

Initialization
IKJEGAUI

Termination
IKJEGAUT

Link-edit each exit as a separate load module. The exits must reside in an APF-authorized library. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see “Installing the standard-format exits” on page 39.

Possible uses

Some possible uses of the TESTAUTH exits are described below.

- Restrict use of the TESTAUTH command to certain users

  The initialization exit can check the user ID and decide, based on your own criteria, to continue processing the TESTAUTH command or not. The exit can return a value in parameter entry 10 in the parameter list (see Figure 72 on page 385) to indicate that the user is authorized or not authorized to use the command, or that the user's RACF authority should be verified. If the user is not authorized, TESTAUTH issues a message; otherwise processing continues.
- Change the operands that the user specifies on the command

  You can use the initialization exit to change the operands that users specify on the TESTAUTH command. The initialization exit receives the address of the command buffer. It can change the operands the user specifies on the TESTAUTH command by using a new command buffer. For example, the initialization exit can scan the command buffer and correct any errors on the command.

  To check the command buffer and change its contents, the initialization exit can:
  - Scan the command buffer and decide, based on your own criteria, to change the command the user issued
  - Obtain storage for a new command buffer
  - Build the new command buffer
  - Update the key, length, and data fields for the new command buffer as follows:

    Key X'02'

    Length
    the length of the new command buffer

    Data the address of the new command buffer
  - Set a return code of 0 and return control to the TESTAUTH command processor

  The exit must not change the command buffer it receives. It must create a new command buffer and return the address of the new command buffer to TESTAUTH.

  For more information about the command buffer and the new command buffer, see “TSO/E standard exit parameter list” on page 32. For information about the format of the command buffer, see “Command buffer” on page 35.

Writing exits for subcommands of the TESTAUTH command

The TESTAUTH command includes a number of subcommands that users can issue while the TESTAUTH command is executing. For example, under the TESTAUTH command, users can issue the LIST subcommand to display the contents of a virtual storage area or registers, or the DELETE subcommand to delete a load module from virtual storage. For more information about using TESTAUTH with subcommands, see *z/OS TSO/E Programming Guide*. For information about TESTAUTH and its subcommands, see *z/OS TSO/E System Programming Command Reference*.

TSO/E provides an initialization exit and a termination exit for the subcommands of the TESTAUTH command. You can use the exits to customize TESTAUTH subcommand processing for your users. The initialization exit receives control before the TESTAUTH command invokes the subcommand. The termination exit receives control just after the TESTAUTH subcommand completes processing.

If the initialization exit returns successfully to the TESTAUTH subcommand and the TESTAUTH command processor itself abends, the TESTAUTH command processor invokes the subcommand termination exit before ending.

You can use the subcommand initialization exit to change the operands that users specify on the subcommand or correct user errors when they issue the subcommand. You can use the termination exit to perform clean-up or special processing prior to subcommand completion. Depending on the processing your subcommand initialization exit performs, you may not need a corresponding subcommand termination exit.

The following highlights some ways you can use the TESTAUTH subcommand exits. For more information about how you can use the exits, see “Possible uses” on page 393.
- Restrict certain users from using the TESTAUTH subcommand
- Correct a user's errors on the TESTAUTH subcommand
- Change the operands a user specifies on the subcommand
- Provide *pseudo-operands* that are equivalent to two or more operands of a TESTAUTH subcommand

TSO/E-supplied exits

TSO/E does not provide default exit routines for the TESTAUTH subcommand initialization and termination exits.

Entry specifications

For the TESTAUTH subcommand initialization and termination exits, the contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions for the initialization exit

The TESTAUTH subcommand initialization exit receives the address of the standard exit parameter list. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32.

Figure 73 on page 390 shows the exit-dependent data that the TESTAUTH subcommand initialization exit receives beginning at offset +36 (decimal) in the parameter list.

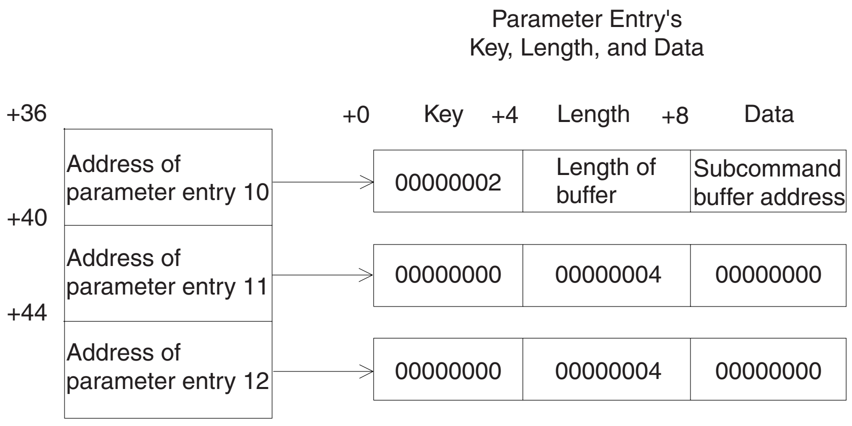


Figure 73. Exit-dependent data for the TESTAUTH subcommand initialization exit

Subcommand Buffer Address (Parameter Entry 10)
This parameter entry contains the address of the buffer containing the text of the entered subcommand and its keywords.

Alternate Buffer (Parameter Entry 11)
This parameter entry lets the exit return the address of an alternate buffer containing a substitute subcommand.

Subcommand Exit Word (Parameter Entry 12)
This parameter entry lets the subcommand initialization exit return a word of information (address or data) that is passed to the subcommand termination exit.

Parameter descriptions for the termination exit

The TESTAUTH subcommand termination exit receives the address of the standard exit parameter list. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32.

Figure 74 shows the exit-dependent data that the TESTAUTH subcommand termination exit receives beginning at offset +36 (decimal) in the parameter list.

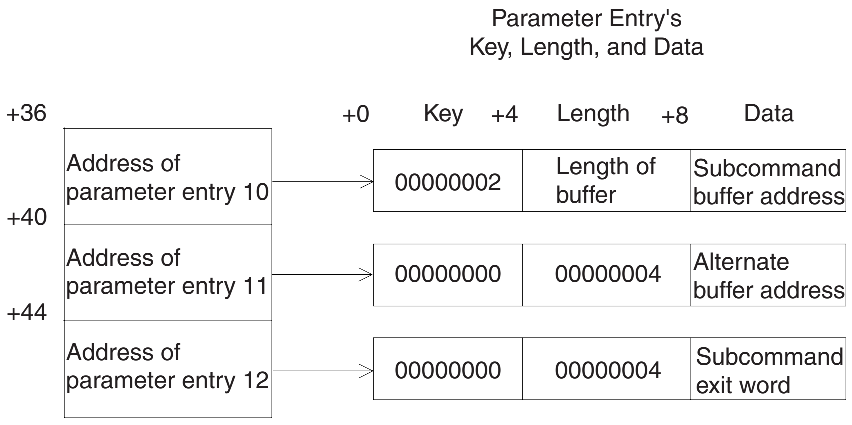


Figure 74. Exit-dependent data for the TESTAUTH subcommand termination exit

Subcommand Buffer Address (Parameter Entry 10)
This parameter entry contains the address of the buffer containing the text of the entered subcommand and its keywords.

Alternate Buffer (Parameter Entry 11)
This parameter entry contains the possible address of an alternate buffer set by the subcommand initialization exit.

Subcommand Exit Word (Parameter Entry 12)
This parameter entry contains a possible word of information (address or data) passed from the subcommand initialization exit.

Return specifications

The contents of the registers on return for both TESTAUTH subcommand exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the subcommand initialization exit

Table 67 shows the return codes that the TESTAUTH subcommand initialization exit supports.

Table 67. Return codes for the TESTAUTH subcommand initialization exit

Return code (decimal)	Description
0	Exit processing was successful. TESTAUTH processing continues.
4	Exit processing was unsuccessful. The subcommand terminates and a message with a reason code is issued.
8	Exit processing was unsuccessful. The subcommand terminates without a message.
12	Exit processing was unsuccessful. TESTAUTH issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the TESTAUTH command processor. For more information about the exit reason code, see “Exit reason code” on page 38.
16	Exit processing was unsuccessful. TESTAUTH terminates processing. The TESTAUTH command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the TESTAUTH command processor terminates without displaying a message to the user.

Return codes for the subcommand termination exit

Table 68 on page 392 shows the return codes that the TESTAUTH subcommand termination exit supports.

Table 68. Return codes for the TESTAUTH subcommand termination exit

Return code (decimal)	Description
0	Exit processing was successful. TESTAUTH processing continues.
12	Exit processing was unsuccessful. TESTAUTH issues an error message to the user and then terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the TESTAUTH command processor. For more information about the exit reason code, see "Exit reason code" on page 38.
16	Exit processing was unsuccessful. TESTAUTH terminates processing. The TESTAUTH command processor does not display a message to the user if the exit sets a return code of 16. Before the exit returns with return code 16, it can display a message to the user, for example, using PUTLINE.

If the exit returns an undefined return code, the TESTAUTH command processor terminates without displaying a message to the user.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exit must be reentrant, refreshable, reusable, and reside in an APF-authorized library. It is inadvisable for the exit to have APF authorization unless it is designed also to be called as the entry point for a job step program.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

In some cases, you may not need to write a termination exit. This depends on:
- Whether the processing that the subcommand initialization exit performs requires a termination exit to perform clean-up activities
- How you use the exits to customize TESTAUTH subcommand processing

If the subcommand initialization exit obtains a system resource, you must write a subcommand termination exit to free the resource. For example, if the initialization exit obtains storage to return a new subcommand buffer to the TESTAUTH command processor, you must provide a subcommand termination exit to free the storage.

The initialization exit can change subcommand operands using the subcommand buffer. The exit checks the subcommand buffer it receives and determines whether to change any operands. To change the operands, the exit must:
- Obtain storage for a new subcommand buffer
- Build the new subcommand buffer
- Update the key, length, and data fields for the address of the new subcommand buffer (parameter entry 11)
- Set return code 0 and return control to TESTAUTH

For more information about the parameter entries, see "TSO/E standard exit parameter list" on page 32. For more information about the format of the subcommand buffer, see "Command buffer" on page 35.

Environment

- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY), ASCMODE(PRIMARY)
- Reside in an APF-authorized library

Restrictions and limitations

The TESTAUTH subcommand exits must reside in an APF-authorized library.

Installing the exits

You must name the TESTAUTH subcommand exits as follows:

Initialization
IKJEGASI

Termination
IKJEGAST

Link-edit each exit as a separate load module. The exits must reside in an APF-authorized library. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in:

- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see “Installing the standard-format exits” on page 39.

Possible uses

Some possible uses of the TESTAUTH subcommand exits are described below:

- Change the operands that the user specifies on the subcommand

  You can use the initialization exit to change the operands that users specify on the TESTAUTH subcommand. The initialization exit receives the address of the subcommand buffer. It can change the operands the user specifies on the TESTAUTH subcommand by using a new command buffer. For example, the initialization exit can scan the subcommand buffer and correct any errors on the subcommand.

  To check the subcommand buffer and change its contents, the initialization exit can:

  – Scan the subcommand buffer and decide, based on your own criteria, to change the subcommand the user issued
  – Obtain storage for a new subcommand buffer
  – Build the new subcommand buffer
  – Update the key, length, and data fields for the new subcommand buffer as follows:

    Key X'02'

    Length
    the length of the new subcommand buffer

    Data the address of the new subcommand buffer

  – Set a return code of 0 and return control to the TESTAUTH command processor.

The exit must not change the subcommand buffer it receives. It must create a new subcommand buffer and return the address of the new subcommand buffer to TESTAUTH. For more information about the subcommand buffer and the new subcommand buffer, see “TSO/E standard exit parameter list” on page 32. For information about the format of the subcommand buffer, see “Command buffer” on page 35.

- Provide installation-defined pseudo-operands

  You can provide *pseudo-operands* for your installation's users that are equivalent to two or more TESTAUTH subcommand operands. Providing pseudo-operands makes it easier for users to issue the TESTAUTH subcommand. Users need not remember several TESTAUTH subcommand operands. They can specify the pseudo-operand. For example, you could associate a pseudo-operand named SUB1 with four TESTAUTH subcommand operands. The initialization exit can scan the subcommand buffer. If the exit finds the pseudo-operand SUB1, it can replace SUB1 with the actual TESTAUTH subcommand operands and return a new subcommand buffer.

Chapter 39. Customizing TRANSMIT and RECEIVE

Users use the TRANSMIT command to send information (a message) or a copy of information (a data set), or both, to another user. Users use the RECEIVE command to retrieve the data. For information about setting up the TRANSMIT and RECEIVE environment, see Chapter 13, "Setting up the TRANSMIT and RECEIVE environment," on page 161.

If RACF is installed, and if your installation has activated security label checking, data sets and messages have security labels (SECLABELs) associated with them. The security label of the transmitted information is checked with the security label of the receiving user's current logon session. The result of that check determines if the receiving user is authorized to receive the data set or message. If the check determines the receiving user is authorized for the proper security label, but is not currently logged on at the proper security label, the data set or message will not be deleted. If the check determines the receiving user is not authorized for the proper security label, then the data set or message may be deleted. However, your installation can set up a JES exit to hold or reroute data sets that would otherwise be deleted.

For more information about setting up security labels, see *z/OS Security Server RACF Security Administrator's Guide*.

Several TSO/E exits enable your installation to modify the way TRANSMIT and RECEIVE operations are performed, or to monitor or restrict transmission activity. The TSO/E exits are described in this chapter.

Writing exits for the TRANSMIT and RECEIVE commands

The following exits can be used to modify the way TRANSMIT and RECEIVE operations are performed:
- TRANSMIT/RECEIVE NAMES data set pre-allocation exit (INMCZ21R)
- TRANSMIT startup exit (INMXZ01 or INMXZ01R)
- TRANSMIT termination exit (INMXZ02 or INMXZ02R)
- TRANSMIT encryption exit (INMXZ03 or INMXZ03R)
- TRANSMIT log data set pre-allocation exit (INMXZ21R)
- RECEIVE initialization exit (INMRZ01 or INMRZ01R)
- RECEIVE termination exit (INMRZ02 or INMRZ02R)
- RECEIVE notification exit (INMRZ04 or INMRZ04R)
- RECEIVE acknowledgment notification exit (INMRZ05R)
- RECEIVE pre-acknowledgment notification exit (INMRZ06R)
- RECEIVE data set pre-processing exit (INMRZ11 or INMRZ11R)
- RECEIVE data set post-processing exit (INMRZ12 or INMRZ12R)
- RECEIVE data set decryption exit (INMRZ13 or INMRZ13R)
- RECEIVE post-prompt exit (INMRZ15R)
- RECEIVE log data set pre-allocation exit (INMRZ21R)
- JES2 TSO Interactive Data Transmission Facility screening and notification exit (exit 13 - MAILXIT)
- JES2 TSO RECEIVE authorization exit (exit 38)
- JES3 TSO Interactive Data Transmission Facility screening and notification exit (IATUX42)
- JES3 Data Set Headers exit (IATUX60)

Notice that most TRANSMIT and RECEIVE exits can have one of two program names: INMxxxx or INMxxxxR (for example, INMXZ01 or INMXZ01R). Both forms of the exits have access to the same information and can perform the same functions. Their differences are strictly programming-related, and are described in “Parameter descriptions” on page 399 and “Programming considerations” on page 402. If TSO/E finds a standard-format exit (INMxxxxR) it will use it and ignore the “non-standard” format exit, even if present. If it does not find a standard-format exit it will use the “non-standard” format exit.

This chapter gives a brief overview of default TRANSMIT and RECEIVE processing, and describes ways you can use exits to modify it. It also provides information required to write TRANSMIT and RECEIVE exits. For information on how to write the JES exits, see either *z/OS JES3 Customization* or *z/OS JES2 Installation Exits*.

Functional description

By default, after a user issues a TRANSMIT command:

- TRANSMIT reads the data set to be transmitted, converts the records to a format suitable for transmission, and adds header information. It then routes the data to a punch output class B file that is directed to the receiving node and user ID. There the data is queued on the JES spool.
- The user to whom the data was sent issues a RECEIVE command, and sees a message that identifies the data set to be received. RECEIVE then prompts the user for information to use in restoring the data set.
- RECEIVE restores the data to its original format, and writes it to the data set indicated by the user. If an acknowledgment is requested, RECEIVE sends an acknowledgment to the person who issued the TRANSMIT command. If the security checker is active and the security labels of the sender and receiver are different, the installation can specify whether the acknowledgment is returned at the sender's or receiver's security label through the RECEIVE pre-acknowledgment notification exit INMRZ06R. The default is to use the sender's security label.

TRANSMIT, RECEIVE, and JES exits are given control at various points in the processing. You can use them to:

- Control who can use the TRANSMIT and RECEIVE commands
- Modify acknowledgment processing
- Enable users to transmit and receive data types other than those that TRANSMIT and RECEIVE support
- Control encryption and decryption processing
- Allocate NAMES and log data sets
- Reply to RECEIVE prompts
- Modify users' responses to RECEIVE prompts
- Record network use

Controlling who uses TRANSMIT and RECEIVE

If you add TRANSMIT and RECEIVE to the table of authorized commands, by default, all TSO/E users can use them. You can use exits to:

- Limit the use of TRANSMIT and RECEIVE to specific users. Use the TRANSMIT startup exit or the RECEIVE initialization exit.

- Restrict the nodes to which a user can send transmitted data. Use the TRANSMIT startup exit, the RECEIVE data set pre-processing exit, or the RECEIVE post-prompt exit.
- Suppress receipt of a data set. Use the RECEIVE data set pre- processing exit or the RECEIVE post-prompt exit.
- Give users authority to receive data addressed to user IDs other than their own. Use the RECEIVE initialization exit.

The JES2 and JES3 exits (exit 13 and IATUX42, respectively) provide similar capability. Both can screen incoming files sent using the TRANSMIT command, and can delete them, reroute them, or allow them to be sent to the targeted user.

Modifying acknowledgment processing

By default, TSO/E users are not notified when they have a transmission to receive. You can use exits to provide that information. Specifically, you can:

- Notify the sender of the original transmission that an acknowledgment is available to be received. Use the RECEIVE acknowledgment notification exit, or the RECEIVE termination exit, exit 13 in JES2, or IATUX42 in JES3.
- Notify the receiver that a file has arrived. Use exit 13, IATUX42, or the TRANSMIT termination exit.
- Prevent or force acknowledgment, using the TRANSMIT startup exit.
- Associate an acknowledgment with a specific transmission. The TRANSMIT startup exit can specify a notification string that gets passed with the transmission. If an acknowledgment is sent, the notification string is included in the acknowledgment transmission. The RECEIVE notification exit, which gets control each time an acknowledgment is received, can use the notification string to tie the transmission to the acknowledgment. The string is also passed to the RECEIVE acknowledgment notification exit.

Transmitting unsupported data types

You can use TRANSMIT to send sequential or partitioned data sets with record formats of F, FS, FB, FBS, V, VB, VBS, and U. Data sets with machine and ASA print-control characters are also supported. Data sets with keys or labels, and ISAM and VSAM data sets are not. RECEIVE can process data from TRANSMIT and PROFS™.

To transmit data types that TRANSMIT and RECEIVE do not support, you can use three exits and two installation-selected utilities that convert the data to a form that TRANSMIT and RECEIVE recognize. The three exits are:

- TRANSMIT startup exit
- RECEIVE pre-processing exit or RECEIVE post-prompt exit
- RECEIVE post-processing exit.

You can use the TRANSMIT start-up exit to detect that a special data set type is being processed and to invoke your utility to copy the data into a temporary, pre-allocated, sequential data set. The exit passes the ddname of the temporary data set back to TRANSMIT and the data is transmitted. The exit can also pass up to ten local control records, which are transmitted with the data.

When the transmission reaches the RECEIVE data set pre-processing exit or the RECEIVE post-prompt exit, that exit is passed information about the transmission. The exit instructs RECEIVE to ignore the target dsname entered by the sender, and writes the received data into a pre-allocated temporary data set. That data set is passed to the RECEIVE data set post-processing exit, which invokes your utility to restore the data to its original format, and write it into the receiving user's target data set.

Controlling encryption and decryption processing

TRANSMIT and RECEIVE provide encryption and decryption options. By setting an installation default in the INMXPARM CSECT, you can require that all transmitted data be encrypted, give users the option of encrypting data, or prohibit encryption. Chapter 13, "Setting up the TRANSMIT and RECEIVE environment," on page 161 describes how to specify defaults in INMXPARM.

If you allow encryption, and have MVS Programmed Cryptographic Unit Support Program installed, TRANSMIT and RECEIVE use the Access Methods Services REPRO function to encipher and decipher the data. Before invoking the service, TRANSMIT or RECEIVE gives control to either the TRANSMIT encryption or the RECEIVE decryption exit. Those exits can:

- Modify the ENCIPHER or DECIPHER options that the user specified
- Invoke encryption or decryption processing themselves
- Prevent the data set from being encrypted or decrypted

Allocating NAMES and log data sets

TRANSMIT and RECEIVE use a NAMES data set to control TRANSMIT and RECEIVE processing and a log data set to keep records of TRANSMIT and RECEIVE activity. By default, TRANSMIT and RECEIVE obtain the NAMES and log data sets from operands specified on the TRANSMIT or RECEIVE command, or else use a NAMES or log data set defined by the user. You can also use exits to allocate the data sets or modify their names for the TRANSMIT and RECEIVE commands.

Specifically, the exits let you:

- Allocate the NAMES data set or modify its name on the TRANSMIT or RECEIVE command, using exit INMCZ21R.
- Allocate the log data set or modify its name on the TRANSMIT command, using exit INMXZ21R.
- Allocate the log data set or modify its name on the RECEIVE command, using exit INMRZ21R.

If RACF is installed and if your installation is using security labels, the installation-defined NAMES data sets must be allocated with a SECLABEL of SYSLOW. This allows TRANSMIT and RECEIVE to access the data set from any security label. For more information about setting up security labels, see *z/OS Security Server RACF Security Administrator's Guide*.

Replying to RECEIVE prompts

RECEIVE prompts users for information to control data restoration. You can use the RECEIVE data set pre-processing exit to specify the same parameters for which RECEIVE would otherwise prompt users, and bypass user prompting. The exit is particularly useful to installations that allow users to issue RECEIVE in the background.

Modifying users' responses to RECEIVE prompts

The users' responses to RECEIVE prompts can be modified using the RECEIVE post-prompt exit INMRZ15R. You can use this exit to:

- Override or add to the information provided by the user in response to the RECEIVE prompt

- Determine which users can use a particular network path
- Receive data set types not supported by the Interactive Data Transmission Facility, in conjunction with the RECEIVE post-processing exit INMRZI2R
- Suppress the reception of data sets

Record network use

Both the TRANSMIT termination and the RECEIVE data set post-processing exits are passed the information required to determine the volume and direction of network traffic. You can use those exits to collect and report statistics. The exits can use SMFWTM or SMFEWTM macros to write SMF records.

TSO/E-supplied exits

TSO/E supplies the following default exits. Each sets register 15 to zero and returns to the caller.

```
INMXZ01    INMRZ01    INMRZ11
INMXZ02    INMRZ02    INMRZ12
INMXZ03    INMRZ04    INMRZ13
```

TSO/E provides no default exits for the standard-format exits (INMxxxxR).

Entry specifications

The contents of the registers on entry to all TRANSMIT and RECEIVE exits are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions

The two forms of the TRANSMIT and RECEIVE exits (INMxxxxR and INMxxxx) receive essentially the same information, but in parameter lists that have different structures.

INMxxxxR parameter lists

At entry to the standard-format exits (those whose names end in R), register 1 points to the standard exit parameter list. For more information about the parameter list, see “TSO/E standard exit parameter list” on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4.

The parameter entries pointed to from offset +36 (decimal) to the end of the parameter list, contain exit-dependent data. The exit-dependent data is described separately for each exit, beginning with "Parameter descriptions for INMXZ01R" on page 407.

INMxxxx parameter lists

At entry to the exits whose names do not end in R, register 1 also points to a parameter list. Those parameters lists are also described separately for each exit, beginning with "Parameter descriptions for INMXZ01" on page 411.

Installation-defined parameters

In addition to pre-defined status and action flags for communicating with the command processors, TRANSMIT and RECEIVE exits can use several other parameters to communicate between themselves, the issuers of TRANSMIT and RECEIVE, and the command processors. Those parameters include the:

- **Exit-to-exit communication word.** The TRANSMIT and RECEIVE command processors pass to each exit an exit-to-exit communication word. The exits can use that word to communicate among themselves by passing information in the storage to which the word points. The startup or initialization exit needs to obtain the block of storage and store its address in the exit-to-exit communication word. TRANSMIT and RECEIVE initially set the word to zero, and thereafter do not modify it or the contents of the storage to which it points.
- **PARM keyword on the TRANSMIT and RECEIVE commands.** TRANSMIT and RECEIVE both have PARM keywords on which the issuer can specify a string of up to 247 bytes. All exits receive the string in their parameter lists, and can act differently for different PARM values. The installation needs to tell intended users what PARM values to specify.
- **PARM tag in NAMES.TEXT data set entries.** Some TRANSMIT and RECEIVE exits can also access a copy of the user string specified on the :PARM tag in the addressee's nickname entry in the sender's NAMES.TEXT data set. The string can be up to 30 bytes long. The PARM tag provides a way for users to request, and exits to provide, different processing for different users. The installation needs to tell intended users what PARM values to specify.
- **A notification string** that the TRANSMIT initialization exit passes to the RECEIVE notification and the RECEIVE acknowledgment notification exits. If the sender requested acknowledgment, the string is also returned to the sender after RECEIVE processing completes. It provides a means of tracking the receipt of specific transmissions, or triggering specific exit actions. The string can be up to 64 bytes long.
- **A user string** that the TRANSMIT initialization exit passes to the RECEIVE pre-processing, post-prompt, post-processing, and decryption exits. The string can be up to 247 bytes long.

Text units and text unit pointer lists

Many of the TRANSMIT and RECEIVE exits are passed addresses of text unit pointer lists. Each entry in the list points to a text unit that contains information used to control the transmission (for example, the data set organization). The exit can read, but not change the information in the text unit.

"Text units and text unit pointer lists" on page 474 shows the format of text units and the text unit pointer list, and describes the contents of each text unit.

Return specifications

On return from all TRANSMIT and RECEIVE exits, the contents of the registers must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for INMxxxxR exits

INMxxxxR exits are expected to pass back in register 15 one of the following return codes.

Table 69. Standard return codes that all INMxxxxR exits support

Return code (decimal)	Description
0	The exit successfully completed. TRANSMIT or RECEIVE processing continues.
12	The exit failed. TRANSMIT or RECEIVE processing ends. RECEIVE issues message IKJ79154I with the reason code in the parameter entry pointed to at offset +24 in the parameter list. If the exit sets a return code of 12, it also needs to provide the reason code, and might want to set an appropriate action flag in the parameter entry pointed to at offset +40.
16	The exit failed. TRANSMIT or RECEIVE processing ends, and either an exit-provided message or a standard error message is issued. To have an exit-provided message issued (either message INMX151I for TRANSMIT exits, or INMR151I for RECEIVE exits), the exit must provide the message text in the parameter entry pointed to at offset +44. It must also set the appropriate action flag in the parameter entry at offset +40.

Return codes for INMxxxx exits

The INMxxxx exits are expected to pass back in register 15 one of the following return codes.

Table 70. Return codes that all INMxxxx exits support

Return code (decimal)	Description
0	The exit successfully completed. TRANSMIT or RECEIVE processing continues.
4	The exit failed. TRANSMIT or RECEIVE processing ends, and either an exit-provided message or a standard error message is issued. To have an exit-provided message issued (either message INMX151I for TRANSMIT exits, or INMR151I for RECEIVE exits), the exit must provide the message text in the parameter entry at offset +16 in the parameter list. It must also set the appropriate action flag in the parameter at offset +8.

Programming considerations

All TRANSMIT and RECEIVE exits must follow standard linkage conventions. They must save registers on entry, and restore all registers, except register 15, on return. The exit must be reentrant, refreshable, reusable, and reside in an APF-authorized library. APF-authorization enables the exit to use restricted authorization checking functions (for example, RACHECK). It is inadvisable for the exit to have APF authorization unless it is designed also to be called as the entry point for a job step program.

Environment

TRANSMIT and RECEIVE exits must have the following attributes:
- State: Problem program
- Key: 8
- For INMxxxxR exits: AMODE(31), RMODE(ANY)
- For INMxxxx exits: AMODE(24), RMODE(24)
- Reside in an APF-authorized library

Restrictions and limitations

- Although INMxxxxR exits can execute in 31-bit addressing mode, the addresses the exits return to TRANSMIT or RECEIVE must point to virtual storage below 16 MB. TRANSMIT and RECEIVE both execute in 24-bit addressing mode.
- Because the TRANSMIT and RECEIVE exits are APF-authorized, use modesets carefully to ensure that integrity is maintained.

Installing the exits

Give the exits the following names, depending on whether they receive a standard exit parameter list.

Exit description	Uses standard exit list	Does not use standard exit list
TRANSMIT startup exit	INMXZ01R	INMXZ01
TRANSMIT termination exit	INMXZ02R	INMXZ02
TRANSMIT encryption exit	INMXZ03R	INMXZ03
RECEIVE initialization exit	INMRZ01R	INMRZ01
RECEIVE data set pre-processing exit	INMRZ11R	INMRZ11
RECEIVE post-prompt exit	INMRZ15R	
RECEIVE data set decryption exit	INMRZ13R	INMRZ13
RECEIVE notification exit	INMRZ04R	INMRZ04
RECEIVE acknowledgment notification exit	INMRZ05R	
TRANSMIT/RECEIVE NAMES data set pre-allocation exit	INMCZ21R	
TRANSMIT log data set pre-allocation exit	INMXZ21R	
RECEIVE log data set pre-allocation exit	INMRZ21R	
RECEIVE pre-acknowledgment notification exit	INMRZ06R	
RECEIVE data set post-processing exit	INMRZ12R	INMRZ12
RECEIVE termination exit	INMRZ02R	INMRZ02

Link-edit INMxxxxR exits as separate load modules, either in a separate load library that is exclusively for TSO/E exits, or in an existing library containing other routines. The exits can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB, provided it is authorized

For more information about choosing between those locations, see "Installing the standard-format exits" on page 39.

Link-edit INMxxxx exits with the TRANSMIT or RECEIVE command processor in SYS1.LINKLIB, and replace the TSO/E-supplied exits with the ones you write.

TRANSMIT and RECEIVE NAMES data set pre-allocation exit — INMCZ21R

Each time a NAMES data set is to be allocated, the TRANSMIT and RECEIVE pre-allocation exit, INMCZ21R, receives control. This exit may be invoked multiple times for each TRANSMIT and RECEIVE operation. This exit can allocate the NAMES data set, modify the NAMES data set name, or take no action. If the exit chooses to allocate the data set, the exit must pass the file name back to the TRANSMIT or RECEIVE command. If the exit modifies the data set name, TRANSMIT or RECEIVE command processing then allocates the data set for the exit if the exit has not already done so.

Under normal operations, the TRANSMIT and RECEIVE commands free the NAMES data sets automatically. Therefore, even if the exit allocated the NAMES data set, the data set is freed when the command processor terminates normally.

For TRANSMIT processing, INMCZ21R receives control just after the command buffer is parsed. Changes made to the command buffer do not affect TRANSMIT processing. During TRANSMIT processing, the allocation of the NAMES data set occurs before the TRANSMIT command initialization exit, INMXZ01(R) receives control. Therefore, exit INMCZ21R receives control before the initialization exit receives control. Exit INMCZ21R can use the exit-to-exit communication word. This communication word is passed to the TRANSMIT initialization exit, INMXZ01(R).

For RECEIVE processing, INMCZ21R receives control during the read operation of the incoming file.

Parameter descriptions for INMCZ21R

When INMCZ21R receives control, register 1 points to the standard exit parameter list. The parameter list is described in "TSO/E standard exit parameter list" on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMCZ21R receives beginning at offset +36 (decimal) in the parameter list.

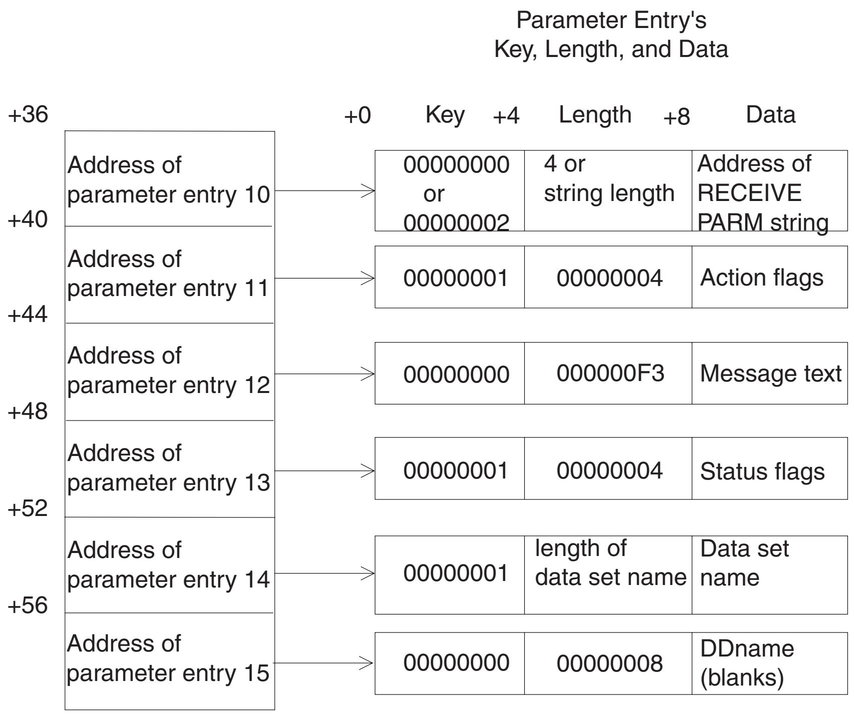


Figure 75. Exit-dependent data on entry to INMCZ21R

Following are descriptions of the information in the data fields of each parameter entry:

Address of the Command PARM String (Parameter Entry 10)

If the user entered a character string on the PARM keyword of the command, this data field contains the address of that string. If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

For more information about how you can use the command PARM keyword, see “Installation-defined parameters” on page 400.

Action Flags (Parameter Entry 11)

The data field contains a word of action flags, which the exit can set to control TRANSMIT and RECEIVE processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
`1... ....`	TRANSMIT or RECEIVE is to issue message INMC151I, using the text contained in the parameter entry pointed to at offset +44.
`.1.. ....`	If the return code from the exit is non-zero (end processing), the command is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that the

command issue message INMC151I, using the text contained in the parameter entry pointed to at offset +44.

`..11 ....`
Reserved

`.... 1...`
The exit has allocated the NAMES data set. The ddname representing the data set is contained in the parameter entry pointed to by offset +56.

`....  .1..`
The exit has modified the NAMES data set name that was passed in the parameter entry pointed to by offset +52.

`.... ..1.`
The exit tried to allocate the NAMES data set, but the allocation failed.

`.... ...1`
Reserved.

`Message Text (Parameter Entry 12)`
Exits can put in this data field the message text that the command is to issue with message ID INMC151I. The field initially contains blanks. If the exit inserts text in the data field, it must also set the key and length values to:

KEY: X'00000001'

LEN: Length of the message text. The maximum length is 243.

`Status Flags (Parameter Entry 13)`
The data field contains a word of status flags in which the command passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2 and 3 are reserved.

Bit **Meaning**

`1... ....`
TRANSMIT or RECEIVE is not interfacing with JES. A data set keyword (INDSNAME, INDATASET, OUTDSNAME, OUTDATASET) or a file keyword (INFILE, INDDNAME, OUTFILE, OUTDDNAME) was specified.

`.111 ....`
Reserved

`.... 1...`
Exit is operating under TRANSMIT

`.... 0...`
Exit is operating under RECEIVE

`.... .111`
Reserved

`Data Set Name (Parameter Entry 14)`
On entry the data field contains the name of the NAMES data set to be allocated by either the command invoking the exit or the exit itself.

If the exit modifies the NAMES data set name it must also update the length and key fields accordingly; and it must set the X'04' action flag to indicate that modifications were made to the NAMES data set name.

On return this information is used by the TRANSMIT or RECEIVE command to decide which NAMES data set to allocate and free (provided the allocate is not done by the exit itself, see parameter entry 15).

If the content of the length field is incorrect, the parameter is ignored and processing continues as it would have if the exit did not modify the NAMES data set name.

Note that if you want the exit to modify the NAMES data set name and to allocate the NAMES data set you must specify parameter entries 14 and 15.

DDNAME (Parameter Entry 15)

If the exit is to allocate the NAMES data set to a file, the file name should be placed in this parameter entry. Update the key and the length field accordingly. The length field must reflect the length of the file name. The X'08' action flag should be set to indicate that the data set has been allocated from within the exit.

Upon return from the exit the invoking TRANSMIT or RECEIVE command uses this information to free the allocated NAMES data set.

If the content of the length field is incorrect, the parameter is ignored and processing continues as it would have if the exit did not modify this parameter entry.

Note that if you want the exit to modify the NAMES data set name and to allocate the NAMES data set you must specify parameter entries 14 and 15.

TRANSMIT startup exit — INMXZ01R or INMXZ01

The TRANSMIT startup exit (INMXZ01R or INMXZ01) receives control after the TRANSMIT command processor has parsed the command and built the addressee list, but before it has done any transmission-related allocations. Its primary uses are to:

- **Control to which nodes and user IDs transmissions are sent.** The exit has access to and can change the information in the addressee list, which specifies the target node and user ID of each addressee. If the TRANSMIT command specified a nickname, the addressee list also includes the nickname and :NAME and :PARM tag information from the NAMES.TEXT data set. The exit can also identify the sender, and thus restrict network use differently for different users.
- **Prevent or force an acknowledgment to be sent to the issuer of TRANSMIT when the transmission is received.** The exit can prevent or force acknowledgment individually for each entry in the addressee list.
- **Perform initialization tasks** You can use the exit to obtain storage for the exit-to-exit communication word, allocate data sets, open files, or do other set-up processing for the other TRANSMIT exits.
- **Enable users to transmit data types other than those supported by TRANSMIT.** The exit can detect when an unsupported data type is being processed. It can then allocate a temporary data set, and invoke an installation-specified utility to convert the data to a format that TRANSMIT recognizes, and copy it into the data set. The exit passes the name of the data set to TRANSMIT for processing. The exit can also pass up to ten local control records, which are sent with the transmission. You must also write either the RECEIVE data set pre-processing exit (INMRZ11R or INMRZ11) or the RECEIVE post-prompt exit (INMRZ15R) and the RECEIVE data set post-processing exit (INMRZ12R or INMRZ12) to process the unsupported data. See “Functional description” on page 396 for more information about the data types TRANSMIT and RECEIVE do not support, and how to use exits to process that data.

Parameter descriptions for INMXZ01R

When INMXZ01R receives control, register 1 points to a standard exit parameter list. For more information about that parameter list, see “TSO/E standard exit parameter list” on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMXZ01R receives beginning at offset +36 (decimal) in the parameter list.

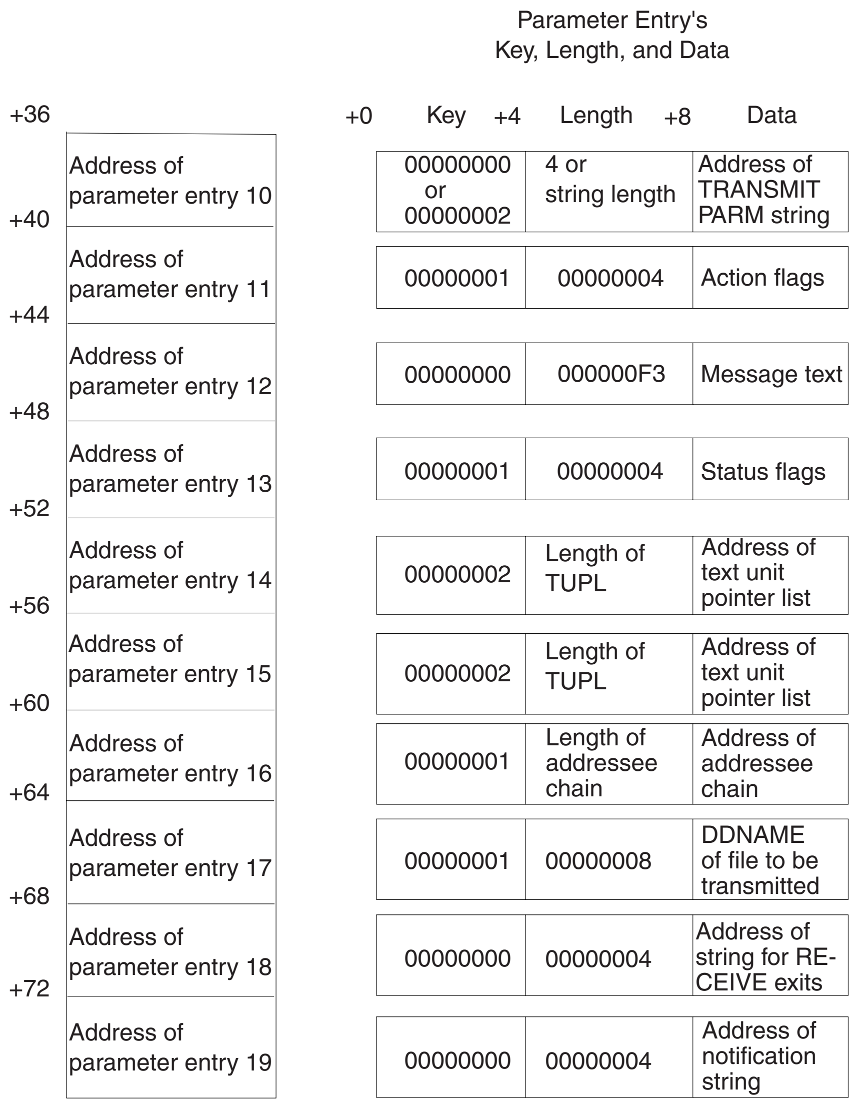

Parameter Entry's
Key, Length, and Data
+36
+0
Key
+4
Length
+8
Data
Address of parameter entry 10
00000000 or 00000002
4 or string length
Address of TRANSMIT PARM string
+40
Address of parameter entry 11
00000001
00000004
Action flags
+44
Address of parameter entry 12
00000000
000000F3
Message text
+48
Address of parameter entry 13
00000001
00000004
Status flags
+52
Address of parameter entry 14
00000002
Length of TUPL
Address of text unit pointer list
+56
Address of parameter entry 15
00000002
Length of TUPL
Address of text unit pointer list
+60
Address of parameter entry 16
00000001
Length of addressee chain
Address of addressee chain
+64
Address of parameter entry 17
00000001
00000008
DDNAME of file to be transmitted
+68
Address of parameter entry 18
00000000
00000004
Address of string for RE-CEIVE exits
+72
Address of parameter entry 19
00000000
00000004
Address of notification string

Figure 76. Exit-dependent data on entry to INMXZ01R

Following are descriptions of the information in the data fields of each parameter entry:

Address of the TRANSMIT PARM String (Parameter Entry 10)

If the user entered a character string on the PARM keyword of the TRANSMIT command, this data field contains the address of that string.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

For more information about how you can use the TRANSMIT PARM keyword, see "Installation-defined parameters" on page 400.

Action Flags (Parameter Entry 11)
The data field contains a word of action flags, which the exit can set to control TRANSMIT processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
`1... ....`	TRANSMIT is to issue message INMX151I, using the text contained in the parameter entry pointed to at offset +44.
`.1.. ....`	If the return code from the exit is non-zero (end processing), TRANSMIT is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that TRANSMIT issue message INMX151I, using the text contained in the parameter entry pointed to at offset +44.
`..11 ....`	Reserved
`.... 1...`	TRANSMIT is to send the data set that the exit specifies, instead of the data set indicated by the user. If the exit sets this bit to one, it also needs to return in the parameter entry at offset +64 the ddname of the data set to be sent.
`.... .111`	Reserved

Message Text (Parameter Entry 12)
Exits can put in this data field the message text that TRANSMIT is to issue with message ID INMX151I. The field initially contains blanks. If the exit inserts text in the data field, it must also set the key and length values to:

KEY: X'00000001'

LEN: Length of the message text. The maximum length is 243.

Status Flags (Parameter Entry 13)
The data field contains a word of status flags in which TRANSMIT passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2 and 3 are reserved.

Bit	Meaning
`1... ....`	The output target for TRANSMIT is not JES. The bit is on when the user specified either OUTDATASET, OUTDSNAME, OUTDDNAME, or OUTFILE on the TRANSMIT command.
`.111 1111`	Reserved

Address of a Text Unit Pointer List (Parameter Entry 14)

The data field contains the address of a text unit pointer list. The text units it points to identify the sender's node and user ID (INMFNODE and INMFUID). For more information about those text units or the text unit pointer list, see "Text units and text unit pointer lists" on page 474.

Note that the exit receives only *copies* of the actual text units; altering them has no effect on subsequent processing.

Address of a Text Unit Pointer List (Parameter Entry 15)

The data field contains the address of a text unit pointer list. The text unit it points to identifies the source of the data being transmitted -- either the data set name, the ddname, or an indication that the data was entered at the terminal (text units INMDSNAM, INDDNAM, or INMTERM, respectively). For more information about those text units or the text unit pointer list, see "Text units and text unit pointer lists" on page 474.

Note that the exit receives only *copies* of the actual text units; altering them has no effect on subsequent processing.

Address of the Addressee Chain (Parameter Entry 16)

The data field points to a chain of addresses to which the transmission is to be sent. By adding to, deleting, or changing information in the chain, the exit can control to whom the transmission is sent, and whether the sender receives an acknowledgment. Each entry in the chain has the following format:

Offset	Length	Value
+0	4	Pointer to the next entry in the chain, or zero to indicate the last entry.
+4	2	The length of the target node name or number specified either on the TRANSMIT command, or, if the issuer specified a nickname, in the associated entry in the NAMES.TEXT data set. Node numbers are valid for only JES2 nodes.
+6	8	Node name or number. The node name or number is left justified and padded with blanks. Node numbers are valid for only JES2 nodes.
+14	2	The length of the addressee's user ID.
+16	8	User ID. The user ID is left justified and padded with blanks.
+24	2	The length of the addressee's nickname. If no nickname is associated with the addressee, the value is zero.
+26	8	Nickname. The nickname is left justified and padded with blanks. If no nickname is associated with the addressee, the field is blank.
+34	2	The length of the addressee's name, taken from the :NAME tag in the NAMES.TEXT data set. If no name is found, or no nickname is associated with the addressee, the value is zero.
+36	30	Addressee's name. The name is left justified and padded with blanks. If no name is found, or no nickname is associated with the addressee, the field is blank.

Offset	Length	Value
+66	1	Flag byte: **1... ....** Request acknowledgment from this addressee **.1.. ....** Reserved **..1. ....** Transmission successfully completed **...1 1111** Reserved
+67	1	Reserved
+68	2	The length of the parameter string taken from the :PARM tag in the NAMES.TEXT data set. If no :PARM tag is found or no nickname is associated with the addressee, the value is zero.
+70	30	User parameter string on the :PARM tag in the NAMES.TEXT data set. The parameter string is left justified and padded with blanks. If no :PARM tag is found or no nickname is associated with the addressee, the field is blank.
+100	2	Halfword field that contains the length of the user ID specified at offset +102 in this entry. If the exit changes the length of that user ID, it needs to put the new length value in this length field.
+102	8	The sender's user ID. The field is left justified and padded with blanks. TRANSMIT places this user ID in the INMFUID text unit of the INMR01 header record of the transmission. Any changes the exit makes are reflected in INMR01. If the exit changes the length of the user ID, it also needs to put the new length value in the length field at offset +100.
+110	2	Halfword field that contains the length of the node name or number specified at the offset +112 in this entry. If the exit changes the length of that node name or number, it needs to put the new length value in this length field.
+112	8	The sender's node name or number. The field is left justified and padded with blanks. TRANSMIT places this node name or number in the INMFNODE text unit of the INMR01 header record of the transmission. Any changes the exit makes are reflected in INMR01. If the exit changes the length of the node name or number, it also needs to put the new length value in the length field at offset +110.

Ddname of the File to be Transmitted (Parameter Entry 17)

The exit can specify in this data field the ddname of a file to be transmitted in place of the data set, file, or terminal input specified by the user. If the exit puts a ddname in this field, it also needs to set action bit X'08' to one to request that TRANSMIT send the associated file.

Address of the User String for RECEIVE Exits (Parameter Entry 18)

The data field points to an area in which the exit can put a string to be passed to the RECEIVE data set exits (INMRZ11R, INMRZ12R, and INMRZ13R, or INMRZ11, INMRZ12, and INMRZ13) and the RECEIVE post-prompt exit (INMRZ15R).

If the exit specifies a string, it must also set the key and length fields to:

KEY:	X'00000002'
LEN:	Length of string. The maximum length is 247 bytes.

Address of a Notification String (Parameter Entry 19)

The data field points to an area in which the exit can put a notification string to be sent as part of the transmission. The notification string is passed to the RECEIVE notification exit (INMRZ04R or INMRZ04) and the RECEIVE acknowledgment notification exit (INMRZ05R). It is returned to the sender after RECEIVE processing completes. For more information about how you can use the notification string, see “Installation-defined parameters” on page 400.

If the exit specifies a string, it must also set the key and length fields to:

KEY:	X'00000002'
LEN:	Length of string. The maximum length is 64 bytes.

Parameter descriptions for INMXZ01

When INMXZ01 receives control, register 1 points to the following parameter list:

Offset	Length	Value
+0	4	Address of an exit-to-exit communication word. TRANSMIT exits can use this word to communicate among themselves by passing information in the storage to which it points. INMXZ01 can obtain the block of storage, and make it available to all exits by storing the address in the exit-to-exit communication word. TRANSMIT initially sets the value of the word to zero, and thereafter, does not modify it or the contents of the storage to which it points.
+4	4	Address of the character string that the user entered on the PARM keyword of the TRANSMIT command. For more information about how you can use the TRANSMIT PARM keyword, see “Installation-defined parameters” on page 400. The address points to an area consisting of a halfword length field followed by the character string. The length field specifies the length of the character string. If no string was specified, the value in the length field is zero.

Offset	Length	Value
+8	4	Address of a byte of action flags, which the exit can set to control TRANSMIT processing after it returns. **Bit** **Action** **`1... ....`** TRANSMIT is to issue message INMX151I, using the text contained in the parameter pointed to at offset +16. **`.1.. ....`** If the return code from the exit is non-zero (end processing), TRANSMIT is not to issue the normal error message. The exit either sent an appropriate message to the user, or requested that TRANSMIT issue message INMX151I, using the text contained in the parameter pointed to at offset +16. **`..11 ....`** Reserved **`.... 1...`** TRANSMIT is to send the data set that the exit specifies, instead of the data set indicated by the user. If the exit sets this bit to one, it also needs to return in the parameter at offset +36 the ddname of the data set to be sent. **`.... .111`** Reserved
+12	4	Address of the TSO/E command processor parameter list (the CPPL). The exit can use the parameter list to examine the initial TRANSMIT command, or to build an IOPL for invoking the TSO/E I/O service routines.
+16	4	Address of the message text that TRANSMIT is to issue with message ID INMX151I. When the exit is invoked, the address points to an area consisting of a halfword length field, followed by an area in which the exit can insert message text. The length value at entry is zero. Exits that pass back message text must put the length of text into the length field. (Do not include the halfword of the length field in the length specification.) The maximum length is 243 bytes.
+20	4	Address of a status flag byte. TRANSMIT uses this byte to pass indicators to the exit. **`1... ....`** The output target for TRANSMIT is not JES. The bit is on when the user specified either OUTDATASET, OUTDSNAME, OUTDDNAME, or OUTFILE on the TRANSMIT command. **`.111 1111`** Reserved

Offset	Length	Value
+24	4	Address of a text unit pointer list. The text units it points to identify the sender's node and user ID (INMFNODE and INMFUID). For more information about those text units or the text unit pointer list, see "Text units and text unit pointer lists" on page 474. Note that the exit receives only *copies* of the actual text units; altering them has no effect on subsequent processing.
+28	4	Address of a text unit pointer list. The text unit it points to identifies the source of the data being transmitted -- either the data set name, the ddname, or an indication that the data was entered at the terminal (text units INMDSNAM, INDDNAM, or INMTERM, respectively). For more information about those text units or the text unit pointer list, see "Text units and text unit pointer lists" on page 474. Note that the exit receives only *copies* of the actual text units; altering them has no effect on subsequent processing.
+32	4	Address of a chain of addresses to which the transmission is to be sent. By adding to, deleting, or changing the information specified in the chain, the exit can control to whom the transmission is sent, and whether the sender receives an acknowledgment. See 414 for the address chain format.
+36	4	Address of an 8-byte area in which the exit can return a ddname that represents an allocated data set. TRANSMIT is to send that data set instead of the data set, file, or terminal input specified by the user. If the exit puts a ddname in this field, it also needs to set action bit X'08' to one to request that TRANSMIT send the associated file.
+40	4	Address of an area in which the exit can put a string to be passed to RECEIVE data set exits (INMRZ11, INMRZ12, and INMRZ13, or INMRZ11R, INMRZ12R, or INMRZ13R) and the RECEIVE post-prompt exit (INMRZ15R). The area consists of a halfword length field, followed by a 247-byte data field. If the exit puts a string in the data field, it also needs to update the length field to reflect the actual length of the data inserted. The value of the length field is initially zero. If the length field is a negative value, RECEIVE ignores the parameter. If length of the data inserted is greater than 247 bytes, TRANSMIT uses only the first 247 bytes, and sets the length to 247.

Offset	Length	Value
+44	4	Address of an area in which the exit can put a notification string to be sent as part of the transmission. The notification string is passed to the RECEIVE notification exit (INMRZ04 or INMRZ04R) and the RECEIVE acknowledgment notification exit (INMRZ05R). It is returned to the sender after RECEIVE processing completes. For more information about how you can use the notification string, see “Installation-defined parameters” on page 400. The format of the area is a halfword length field, followed by a 64-byte data field. If the exit puts a string in the data field, it needs to update the length field to reflect the actual length of the data inserted. The value of the length field is initially zero. If the length field is a negative value, RECEIVE ignores the parameter. If length of the data inserted is greater than 64 bytes, TRANSMIT uses only the first 64 bytes, and sets the length to 64.

Each entry in the address chain has the following format:

Offset	Length	Value
+0	4	Pointer to the next entry in the chain, or zero to indicate the last entry.
+4	2	The length of the target node name or number specified either on the TRANSMIT command, or, if the issuer specified a nickname, in the associated entry in the NAMES.TEXT data set. Node numbers are valid for only JES2 nodes.
+6	8	Node name or number. The node name or number is left justified and padded with blanks. Node numbers are valid for only JES2 nodes.
+14	2	The length of the addressee's user ID.
+16	8	User ID. The user ID is left justified and padded with blanks.
+24	2	The length of the addressee's nickname. If no nickname is associated with the addressee, the value is zero.
+26	8	Nickname. The nickname is left justified and padded with blanks. If no nickname is associated with the addressee, the field is blank.
+34	2	The length of the addressee's name, taken from the :NAME tag in the NAMES.TEXT data set. If no name is found, or no nickname is associated with the addressee, the value is zero.
+36	30	Addressee's name. The name is left justified and padded with blanks. If no name is found, or no nickname is associated with the addressee, the field is blank.

Offset	Length	Value
+66	1	Flag byte: **1... ....** Request acknowledgment from this addressee **.1.. ....** Reserved **..1. ....** Transmission successfully completed **...1 1111** Reserved
+67	1	Reserved
+68	2	The length of the parameter string taken from the :PARM tag in the NAMES.TEXT data set. If no :PARM tag is found or no nickname is associated with the addressee, the value is zero.
+70	30	User parameter string on the :PARM tag in the NAMES.TEXT data set. The parameter string is left justified and padded with blanks. If no :PARM tag is found or no nickname is associated with the addressee, the field is blank.
+100	2	Halfword field that contains the length of the user ID specified at offset +102 in this entry. If the exit changes the length of that user ID, it needs to put the new length value in this length field.
+102	8	The sender's user ID. The field is left justified and padded with blanks. TRANSMIT places this user ID in the INMFUID text unit of the INMR01 header record of the transmission. Any changes the exit makes are reflected in INMR01. If the exit changes the length of the user ID, it also needs to put the new length value in the length field at offset +100.
+110	2	Halfword field that contains the length of the node name or number specified at the offset +112 in this entry. If the exit changes the length of that node name or number, it needs to put the new length value in this length field.
+112	8	The sender's node name or number. The field is left justified and padded with blanks. TRANSMIT places this node name or number in the INMFNODE text unit of the INMR01 header record of the transmission. Any changes the exit makes are reflected in INMR01. If the exit changes the length of the node name or number, it also needs to put the new length value in the length field at offset +110.

TRANSMIT termination exit — INMXZ02R or INMXZ02

The TRANSMIT termination exit (INMXZ02R or INMXZ02) receives control after TRANSMIT command processing is complete -- either the data has been transmitted or some error preventing transmission has occurred -- or when the TRANSMIT command ends abnormally or terminates because of an attention interrupt. The termination exit's primary uses are to:

- **Record network use.** The exit can access the information required to determine the volume and direction of network traffic. You can use SMFWTM or SMFEWTM macros to write SMF records from the exit.
- **Notify recipients on the same node that the transmission has been sent.** The exit can issue a SEND command to users on the same node to inform them they have data to receive. The exit can examine the TRANSMIT return code to see whether data was transmitted.
- **Clean up.** The exit needs to perform clean-up tasks, such as freeing storage obtained for the exit-to-exit communication word, closing any files that other TRANSMIT exits opened, and freeing any data sets that they allocated.

Parameter descriptions for INMXZ02R

When INMXZ02R receives control, register 1 points to the standard exit parameter list. For more information about that parameter list, see “TSO/E standard exit parameter list” on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMXZ02R receives beginning at offset +36 (decimal) in the parameter list.

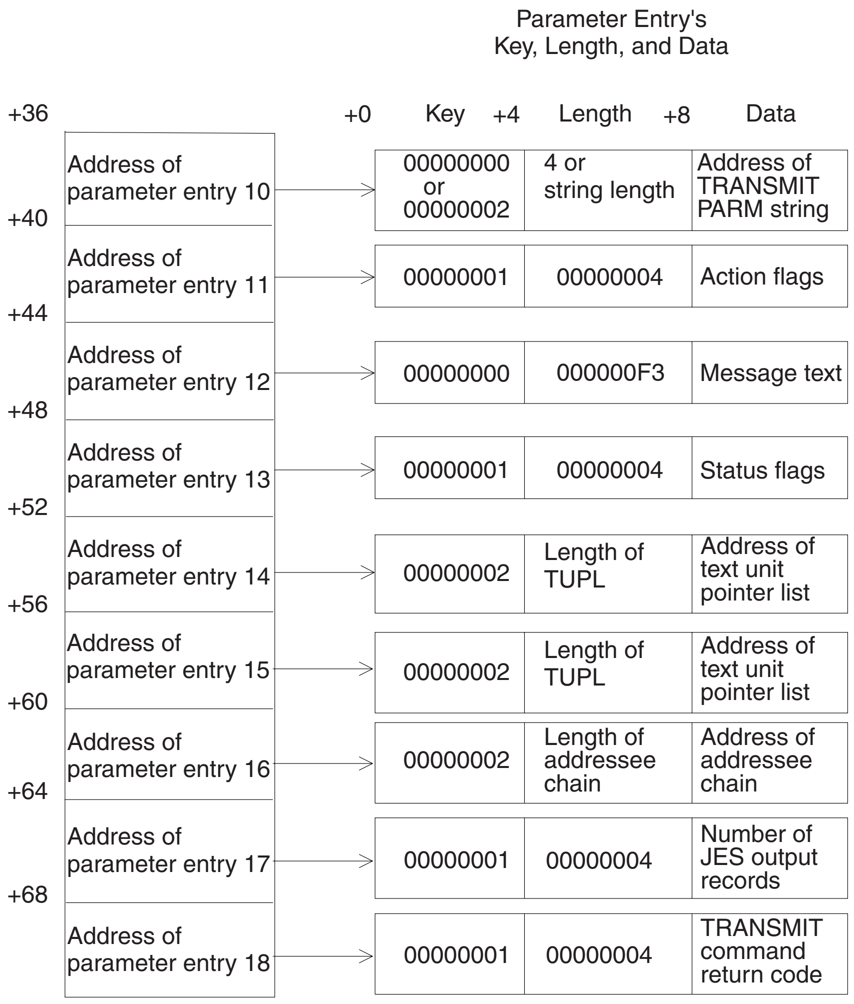

Parameter Entry's
Key, Length, and Data
+36
+40
+44
+48
+52
+56
+60
+64
+68
+0
Key
+4
Length
+8
Data
Address of parameter entry 10
00000000 or 00000002
4 or string length
Address of TRANSMIT PARM string
Address of parameter entry 11
00000001
00000004
Action flags
Address of parameter entry 12
00000000
000000F3
Message text
Address of parameter entry 13
00000001
00000004
Status flags
Address of parameter entry 14
00000002
Length of TUPL
Address of text unit pointer list
Address of parameter entry 15
00000002
Length of TUPL
Address of text unit pointer list
Address of parameter entry 16
00000002
Length of addressee chain
Address of addressee chain
Address of parameter entry 17
00000001
00000004
Number of JES output records
Address of parameter entry 18
00000001
00000004
TRANSMIT command return code

Figure 77. Exit-dependent data on entry to INMXZ02R

Note that many of the parameter entries are the same as those that INMXZ01R receives. Their descriptions are not repeated here. See “Parameter descriptions for

INMXZ01R" on page 407 for more information about them. Following are descriptions of only those parameter entries that are different, or that INMXZ02R must use differently:

`Action Flags (Parameter Entry 11)`
The data field contains a word of action flags, which the exit can set to control TRANSMIT processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
`1... ....`	TRANSMIT is to issue message INMX151I, using the text contained in the parameter entry pointed to at offset +44.
`.1.. ....`	If the return code from the exit is non-zero (end processing), TRANSMIT is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that TRANSMIT issue message INMX151I, using the text contained in the parameter entry pointed to at offset +44.
`..11 1111`	Reserved

`Address of Addressee Chain (Parameter Entry 16)`
This parameter entry is the same as parameter entry 16 that INMXZ01R receives. However, unlike INMXZ01R, this exit cannot affect subsequent processing by changing the information in the chain. It receives control after the transmission occurs.

`Number of JES Output Records (Parameter Entry 17)`
The data field contains the number of JES output records in the transmission sent to the last user in the addressee list. The number of records transmitted to others in the list might vary by one or two, because of variations in the length of the information identifying the addressee.

`TRANSMIT Return Code (Parameter Entry 18)`
The data field contains the return code that TRANSMIT sets.

Parameter descriptions for INMXZ02

When INMXZ02 receives control, register 1 points to the following parameter list. Note that many of the parameters are the same as those that INMXZ01 receives. Their descriptions are not repeated here. See "Parameter descriptions for INMXZ01" on page 411 for more information about them.

Offset	Length	Value
+0	4	Address of an exit-to-exit communication word.
+4	4	Address of the character string that the user entered on the PARM keyword of the TRANSMIT command.

Offset	Length	Value
+8	4	Address of a byte of action flags, which the exit can set to control TRANSMIT processing after it returns. **Bit** **Action** **1... ....** TRANSMIT is to issue message INMX151I, using the text contained in the parameter pointed to at offset +16. **.1.. ....** If the return code from the exit is non-zero (end processing), TRANSMIT is not to issue the normal error message. The exit either sent an appropriate message to the user, or requested that TRANSMIT issue message INMX151I, using the text contained in the parameter pointed to at offset +16. **..11 1111** Reserved
+12	4	Address of the TSO/E command processor parameter list (the CPPL).
+16	4	Address of the message text that TRANSMIT is to issue with message ID INMX151I.
+20	4	Address of a status flag byte. TRANSMIT uses this byte to pass indicators to the exit. **1... ....** The output target for TRANSMIT is not JES. The bit is on when the user specified either OUTDATASET, OUTDSNAME, OUTDDNAME, or OUTFILE on the TRANSMIT command. **.111 1111** Reserved.
+24	4	Address of a text unit pointer list. The text units it points to identify the sender's node and user ID (INMFNODE and INMFUID).
+28	4	Address of a text unit pointer list. The text unit it points to identifies the source of the data being transmitted -- either the data set name, the ddname, or an indication that the data was entered at the terminal (text units INMDSNAM, INDDNAM, or INMTERM, respectively).
+32	4	Address of a chain of addresses to which the transmission is to be sent. Unlike INMXZ01, this exit cannot affect subsequent processing by changing the information specified in the chain. The exit receives control after the transmission has occurred.
+36	4	Address of a fullword that contains the number of JES output records in the transmission sent to the last user in the addressee list. The number of records transmitted to others in the list might vary by one or two, because of variations in the length of the information identifying the addressee.
+40	4	Address of a fullword that contains the TRANSMIT command return code.

TRANSMIT encryption exit — INMXZ03R or INMXZ03

The TRANSMIT encryption exit (INMXZ03R or INMXZ03) receives control immediately after TRANSMIT prompts the user for REPRO command options for enciphering the data to be transmitted. (It is invoked only once, regardless of the number of people to whom the data is being sent.)

Note: If RACF is installed, and if your installation is using security labels, this exit receives control *after* INMXZ21R. The encryption exit can:

- Examine and, if appropriate, change the ENCIPHER options the user specifies. If TRANSMIT is to do the enciphering, it builds the REPRO command after the exit returns, using the changes the exit makes.
- Invoke encryption processing itself. The exit receives all the control and ddnames that TRANSMIT uses for encryption processing.
- Decide that encryption is not necessary, and prevent TRANSMIT from doing it.

Parameter descriptions for INMXZ03R

When INMXZ03R receives control, register 1 points to the standard exit parameter list. For more information about the parameter list, see “TSO/E standard exit parameter list” on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMXZ03R receives beginning at offset +36 (decimal).

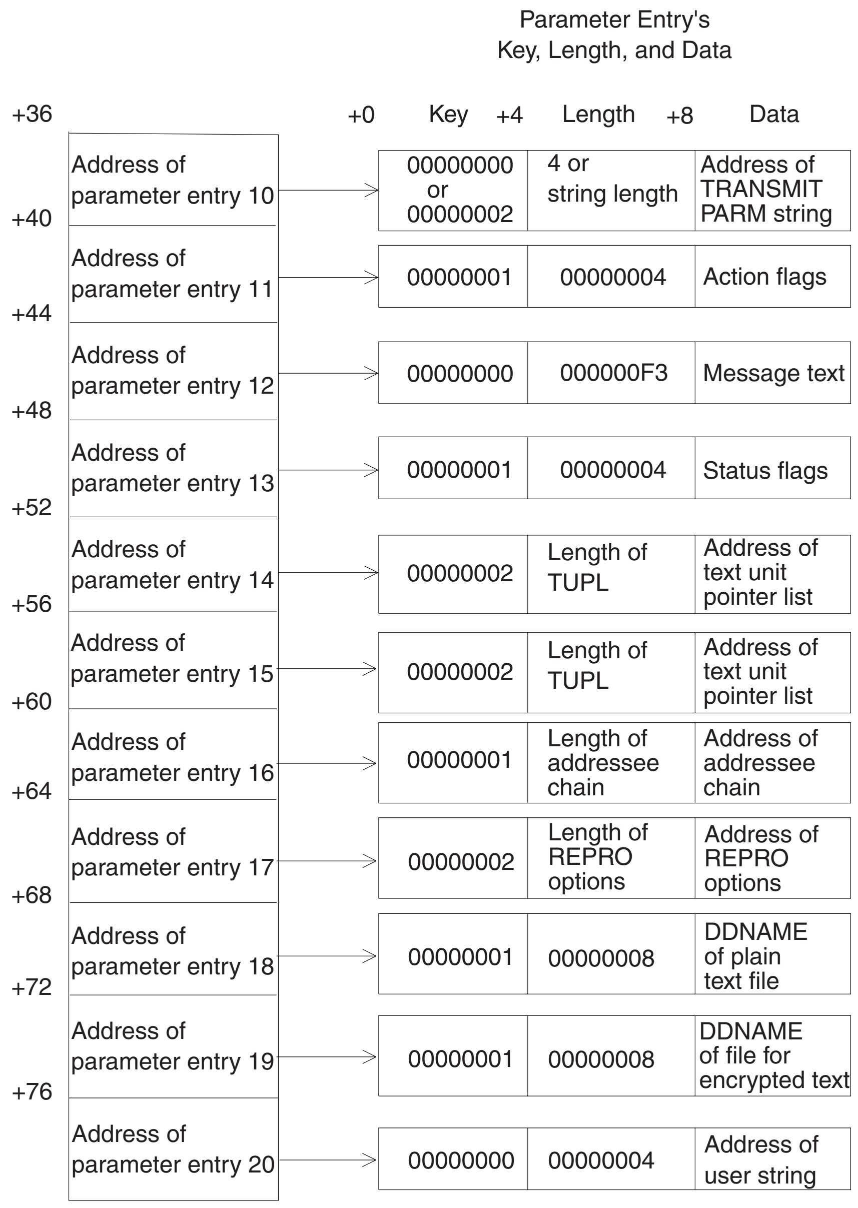

Parameter Entry's
Key, Length, and Data
+36
+0
Key
+4
Length
+8
Data
Address of parameter entry 10
00000000 or 00000002
4 or string length
Address of TRANSMIT PARM string
+40
Address of parameter entry 11
00000001
00000004
Action flags
+44
Address of parameter entry 12
00000000
000000F3
Message text
+48
Address of parameter entry 13
00000001
00000004
Status flags
+52
Address of parameter entry 14
00000002
Length of TUPL
Address of text unit pointer list
+56
Address of parameter entry 15
00000002
Length of TUPL
Address of text unit pointer list
+60
Address of parameter entry 16
00000001
Length of addressee chain
Address of addressee chain
+64
Address of parameter entry 17
00000002
Length of REPRO options
Address of REPRO options
+68
Address of parameter entry 18
00000001
00000008
DDNAME of plain text file
+72
Address of parameter entry 19
00000001
00000008
DDNAME of file for encrypted text
+76
Address of parameter entry 20
00000000
00000004
Address of user string

Figure 78. Exit-dependent data on entry to INMXZ03R

Note that many of the parameter entries are the same as those that INMXZ01R receives. Their descriptions are not repeated here. See “Parameter descriptions for INMXZ01R” on page 407 for more information about them. Following are descriptions of only those parameter entries that are different.

Action Flags (Parameter Entry 11)

The data field contains a word of action flags, which the exit can set to control TRANSMIT processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
`1... ....`	TRANSMIT is to issue message INMX151I, using the text contained in the parameter entry pointed to at offset +44.
`.1.. ....`	If the return code from the exit is non-zero (end processing),

TRANSMIT is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that TRANSMIT issue message INMX151I, using the text contained in the parameter entry pointed to at offset +44.

..11 ....
Reserved

.... 1...
TRANSMIT is not to issue the REPRO command. Either the exit did the encryption, or none is required.

If terminal input is to be encrypted, it is sent as a data set with the name 'prefix.DDNAME.TERMIN'. If the exit sets this bit to one and encrypts the input, the data set name remains 'prefix.DDNAME.TERMIN'.

Setting this bit to one does not prevent TRANSMIT from prompting the user for encryption options. TRANSMIT prompts for those options before invoking this exit.

.... .1..
The encrypted data is now in the sequential file identified by the ddname in the parameter entry pointed to at offset +72.

If the exit sets this bit to one, it also needs to set bit X'08' to one. If this bit is one and bit X'08' is zero, TRANSMIT invokes the REPRO services, and specifies the file pointed to at offset +68 as the input file and the file pointed to at offset +72 as the output file. The file pointed to at offset +72 is overwritten. If both bits are one, TRANSMIT builds an INMR02 control record for the encryption process which results in exit INMRZ13 or INMRZ13R gaining control during RECEIVE to decrypt the data set.

.... ..11
Reserved

Address of the Addressee Chain (Parameter Entry 16)
This parameter entry is the same as the parameter entry 16 that INMXZ01R receives. Like INMXZ01R, this exit can alter subsequent processing by changing the information in the chain.

Address of REPRO Options (Parameter Entry 17)
The data field points to an area that contains the options the user specified on the ENCIPHER keyword of the REPRO command. TRANSMIT uses that information to build a REPRO control statement that looks like the following.

```
REPRO INFILE(infile) OUTFILE(outfile) ENCIPHER(options
specified by user and INMRZ03R)
```

The exit can modify the ENCIPHER options. The modified information is used in encryption processing.

If the exit changes the length of the options string, it must also change the value in the length field accordingly. The maximum length allowed is 253 bytes. If length of the data inserted is greater than 253 bytes, TRANSMIT uses only the first 253 bytes, and sets the length to 253.

`Ddname of Plain Text File (Parameter Entry 18)`
The data field contains the ddname of the file that contains the data to be encrypted.

`Ddname of File for Encrypted Data (Parameter Entry 19)`
The data field contains the ddname of the file to which the encrypted data is to be written.

`Address of a String for INMRZ13R (Parameter Entry 20)`
This exit can pass a user string to the RECEIVE data set decryption exit (INMRZ13R or INMRZ13). If the exit passes a string, it must also put the following values in the key and length fields:

KEY: X'00000002'

LEN: Length of the string. The maximum length is 253 bytes.

Parameter descriptions for INMXZ03

When INMXZ03 receives control, register 1 points to the following parameter list. Note that many of the parameters are the same as those that INMXZ01 receives. Their descriptions are not repeated here. See “Parameter descriptions for INMXZ01” on page 411 for more information about them.

Offset	Length	Value
+0	4	Address of an exit-to-exit communication word.
+4	4	Address of the character string that the user entered on the PARM keyword of the TRANSMIT command.

Offset	Length	Value

+8	4	Address of a byte of action flags, which the exit can set to control TRANSMIT processing after it returns.
		
		Bit **Action**
		
		`1... ....`
		TRANSMIT is to issue message INMX151I, using the text contained in the parameter pointed to at offset +16.
		
		`.1.. ....`
		If the return code from the exit is non-zero (end processing), TRANSMIT is not to issue the normal error message. The exit either sent an appropriate message to the user, or requested that TRANSMIT issue message INMX151I, using the text contained in the parameter pointed to at offset +16.
		
		`..11 ....`
		Reserved
		
		`.... 1...`
		TRANSMIT is not to issue the REPRO command. Either the exit did the encryption, or none is required.
		
		If terminal input is to be encrypted, it is sent as a data set with the name ‘prefix.DDNAME.TERMIN’. If the exit sets this bit to one and encrypts the input, the data set name remains ‘prefix.DDNAME.TERMIN’.
		
		Setting this bit to one does not prevent TRANSMIT from prompting the user for encryption options. TRANSMIT prompts for those options before invoking this exit.
		
		`.... .1..`
		The encrypted data is now in the sequential file identified by the ddname in the parameter entry pointed to at offset +44.
		
		If the exit sets this bit to one, it also needs to set bit X'08' to one. If this bit is one and bit X'08' is zero, TRANSMIT invokes the REPRO services, and specifies the file pointed to at offset +40 as the input file and the file pointed to at offset +44 as the output file. The file pointed to at offset +44 is overwritten. If both bits are one, TRANSMIT builds an INMR02 control record for the encryption process which results in exit INMRZ13 gaining control during RECEIVE to decrypt the data set.
		
		`.... ..11`
		Reserved

+12	4	Address of the TSO/E command processor parameter list (the CPPL).
+16	4	Address of the message text that TRANSMIT is to issue with message ID INMX151I.

Offset	Length	Value
+20	4	Address of a status flag byte. TRANSMIT uses this byte to pass indicators to the exit. **1... ....** The output target for TRANSMIT is not JES. The bit is on when the user specified either OUTDATASET, OUTDSNAME, OUTDDNAME, or OUTFILE on the TRANSMIT command. **.111 1111** Reserved.
+24	4	Address of a text unit pointer list. The text units it points to identify the sender's node and user ID (INMFNODE and INMFUID).
+28	4	Address of a text unit pointer list. The text unit it points to identifies the source of the data being transmitted -- either the data set name, the ddname, or an indication that the data was entered at the terminal (text units INMDSNAM, INDDNAM, or INMTERM, respectively).
+32	4	Address of a chain of addresses to which the transmission is to be sent. Like INMXZ01, this exit can affect subsequent processing by changing the information specified in the chain.

+36	4	An area that contains the options the user specified on the ENCIPHER keyword of the REPRO command. TRANSMIT uses that information to build a REPRO control statement that looks like:
		`REPRO INFILE(infile) OUTFILE(outfile) ENCIPHER`
		`(options specified by user and INMRZ03R)`
		
		The exit can modify the ENCIPHER options. The modified information is used in encryption processing.
		
		The address points to an area that consists of a halfword length field, followed by a character string that contains the options. The length field specifies the length of the character string. If no string is specified, the value of the length field is zero.
		
		If the exit changes the length of the options string, it needs to put the new length value in the length field. The maximum length allowed is 253 bytes. If the length of the data inserted is greater than 253 bytes, TRANSMIT uses only the first 253 bytes, and sets the length to 253.

+40	4	Address of an eight-character field containing the ddname of the file that holds the data to be encrypted.
+44	4	Address of an eight-character field containing the ddname of the file to which the encrypted data is to be written.
+48	4	Address of an area in which the exit can put a string to be passed to the RECEIVE data set decryption exit (INMRZ13 or INMRZ13R). The format of the area is a halfword length field, followed by a 247-byte data field. If the exit puts a string in the data field, it also needs to update the length field to reflect the actual length of the data inserted. The value of the length field is initially zero. If the length field is a negative value, RECEIVE ignores the parameter. If length of the data inserted is greater than 247 bytes, TRANSMIT uses only the first 247 bytes, and sets the length to 247.

TRANSMIT log data set pre-allocation exit — INMXZ21R

Each time a log data set is to be allocated, the TRANSMIT pre-allocation exit, INMXZ21R, receives control. This exit may be invoked multiple times for each TRANSMIT operation. This exit can allocate the log data set, modify the data set name, or take no action. If the exit chooses to allocate the data set, the exit must pass the file name back to the TRANSMIT command. If the exit modifies the data set name, the TRANSMIT command will then allocate the data set for the exit if the exit has not already done so.

Under normal operations, the TRANSMIT command frees the log data sets automatically. Therefore, even if the exit allocated the log data set, the data set is freed when the command processor terminates normally.

During TRANSMIT processing, the allocation of the log data set occurs after the TRANSMIT command has freed the output file. This is after exit INMXZ03(R) would receive control.

If RACF is installed, and if you are using security labels, exit INMXZ03 receives control *after* INMXZ21R.

If the exit is to allocate the log data set, consider the following:

- TRANSMIT allocates the data set as NEW if it does not exist. This frees users from having to allocate the data set themselves.
- TRANSMIT allocates the data set with a disposition of NEW or MOD. The exit should also allocate the data set with a disposition of NEW or MOD.
- If the exit allocates the data set as NEW, the data set must have the characteristics of a log data set. The characteristics of a log data set are:
  - Record Format (RECFM) of VB
  - Logical Record Length (LRECL) of 255
  - Blocksize (BLKSIZE) of 3120

Errors during log processing do not normally terminate command processing. Instead, the command sets a return code of 4, indicating that failure occurred but the transmission was successful. If the exit gives a non-zero return code, TRANSMIT does not terminate processing, but sets a return code of 4, indicating that the logging function encountered an error.

Note: If RACF is installed, and if your installation is using security labels, the TRANSMIT command first opens all log data sets before transmitting the data. If any log data set cannot be opened, TRANSMIT processing will terminate.

Parameter descriptions for INMXZ21R

When INMXZ21R receives control, register 1 points to the standard exit parameter list. The parameter list is described in "TSO/E standard exit parameter list" on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMXZ21R receives beginning at offset +36 (decimal) in the parameter list.

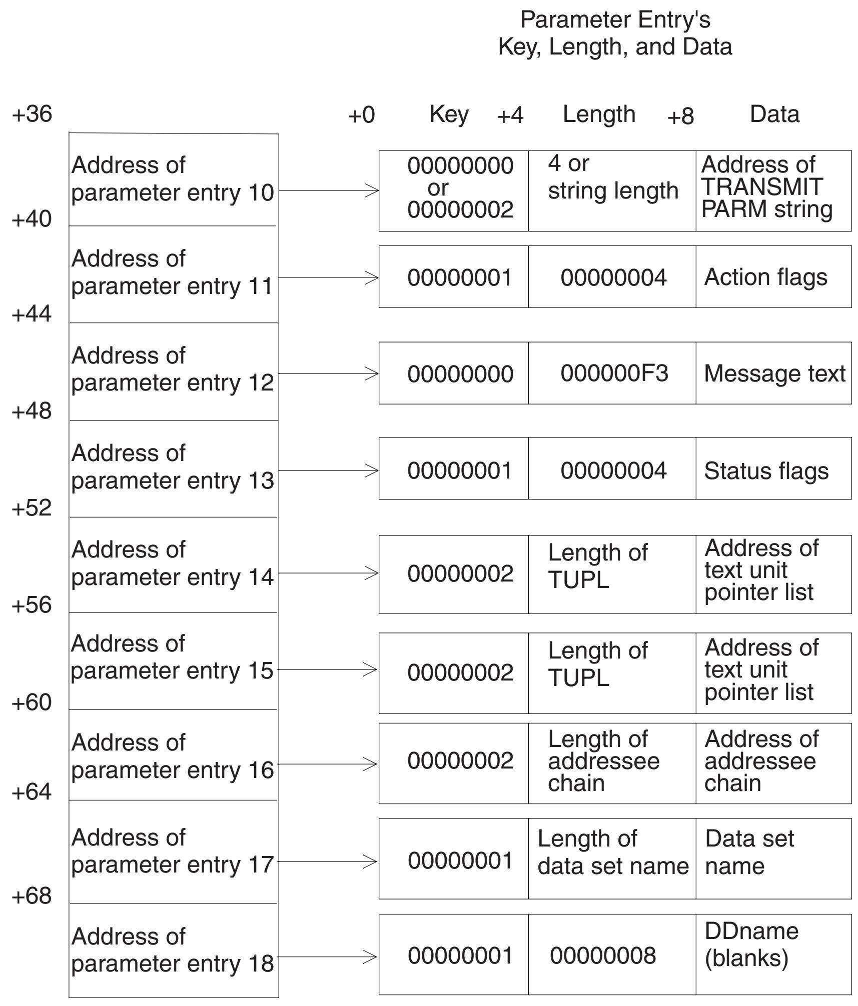

Parameter Entry's
Key, Length, and Data
+36
+40
+44
+48
+52
+56
+60
+64
+68
+0
Key
+4
Length
+8
Data
Address of parameter entry 10
00000000 or 00000002
4 or string length
Address of TRANSMIT PARM string
Address of parameter entry 11
00000001
00000004
Action flags
Address of parameter entry 12
00000000
000000F3
Message text
Address of parameter entry 13
00000001
00000004
Status flags
Address of parameter entry 14
00000002
Length of TUPL
Address of text unit pointer list
Address of parameter entry 15
00000002
Length of TUPL
Address of text unit pointer list
Address of parameter entry 16
00000002
Length of addressee chain
Address of addressee chain
Address of parameter entry 17
00000001
Length of data set name
Data set name
Address of parameter entry 18
00000001
00000008
DDname (blanks)

Figure 79. Exit-dependent data on entry to INMXZ21R

Following are descriptions of the information in the data fields of each parameter entry:

Address of the TRANSMIT PARM String (Parameter Entry 10)

If the user entered a character string on the PARM keyword of the TRANSMIT command, this data field contains the address of that string.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

For more information about how you can use the TRANSMIT PARM keyword, see "Installation-defined parameters" on page 400.

Action Flags (Parameter Entry 11)

The data field contains a word of action flags, which the exit can set to control TRANSMIT processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
`1... ....`	TRANSMIT is to issue message INMX151I, using the text contained in the parameter entry pointed to at offset +44.
`.1.. ....`	If the return code from the exit is non-zero (end processing), the command is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that the command issue message INMX151I, using the text contained in the parameter entry pointed to at offset +44.
`..11 ....`	Reserved
`.... 1...`	The exit has allocated the log data set. The ddname representing the data set is contained in the parameter entry pointed to by offset +64.
`.... .1..`	The exit has modified the log data set name that was passed in the parameter entry pointed to by offset +68.
`.... ..11`	Reserved

Message Text (Parameter Entry 12)

Exits can put in this data field the message text that the command is to issue with message ID INMX151I. The field initially contains blanks. If the exit inserts text in the data field, it must also set the key and length values to:

KEY:	X'00000001'
LEN:	Length of the message text. The maximum length is 243.

Status Flags (Parameter Entry 13)

The data field contains a byte of status flags in which the command passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2 and 3 are reserved.

Bit	Meaning
`1... ....`	TRANSMIT is not interfacing with JES. A data set keyword (OUTDSNAME, OUTDATASET) or a file keyword (OUTFILE, OUTDDNAME) was specified.
`.111 1111`	Reserved

Address of a Text Unit Pointer List (Parameter Entry 14)

The data field contains the address of a text unit pointer list. The text units it points to identify the sender's node and user ID (INMFNODE and INMFUID). For more information about those text units or the text unit pointer list, see “TSO/E standard exit parameter list” on page 32.

Note that the exit receives only copies of the actual text units; altering them has no effect on subsequent processing.

Address of a Text Unit Pointer List (Parameter Entry 15)

The data field contains the address of a text unit pointer list. The text unit it points to identifies the source of the data being transmitted -- either the data set name, the ddname, or an indication that the data was entered at the terminal (text units INMDSNAM, INMDDNAM or INMTERM). For more information about those text units or the text unit pointer list, see "Text units and text unit pointer lists" on page 474.

Note that the exit receives only copies of the actual text units; altering them has no effect on subsequent processing.

`Address of the Addressee Chain (Parameter Entry 16)`
This parameter entry is the same as the parameter entry 16 that INMXZ01R receives. Like INMXZ01R, this exit can alter subsequent processing by changing the information in the chain. See "Parameter descriptions for INMXZ01R" on page 407 for a description of this parameter entry.

`Data Set Name (Parameter Entry 17)`
The data field contains the name of the log data set that will be allocated by the command. If the exit chooses to modify the data set name, the length field should be updated to reflect the correct length of the data set name. In addition, if the exit modifies the data set name, the X'04' action flag should be set to indicate that modifications were made.

If the exit updates this parameter entry, update the length and key fields accordingly. If the length fields are incorrect, the parameter is ignored and processing continues as it would have if the exit did not modify this parameter.

`DDNAME (Parameter Entry 18)`
If the exit allocates the log data set to a file, the file name should be placed in this parameter entry. The length should be updated to reflect the length of the file name. If the exit allocates the data set, the X'08' action flag should be set to indicate that the data set has been allocated. If the exit allocates the data set, the command will free the data set through normal processing.

If the exit updates this parameter entry, update the length and key fields accordingly. If the length fields are incorrect, the parameter is ignored and processing continues as it would have if the exit did not modify this parameter.

RECEIVE initialization exit — INMRZ01 or INMRZ01R

The RECEIVE initialization exit (INMRZ01 or INMRZ01R) receives control after the RECEIVE command processor parses the command input line, but before it takes any other action. It receives control once per RECEIVE invocation. You can use the exit to:

- Perform initialization tasks, such as obtaining storage for the exit-to-exit communication word, and opening files.
- Determine if issuers of RECEIVE are allowed to use the command in any form.
- Determine if issuers of RECEIVE who do not have OPERATOR authority are allowed to read files not addressed to their user IDs.

Parameter descriptions for INMRZ01R

When INMRZ01R receives control, register 1 points to the standard exit parameter list. For more information about the parameter list, see "TSO/E standard exit parameter list" on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMRZ01R receives beginning at offset +36 (decimal).

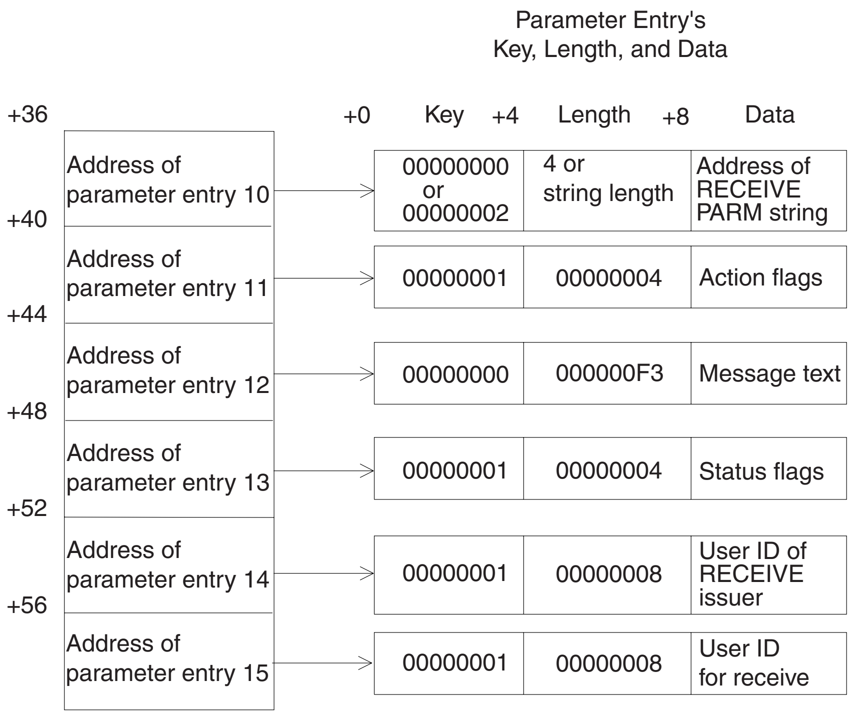


Figure 80. Exit-dependent data on entry to INMRZ01R

Following are descriptions of the information in the data fields of each parameter entry:

`Address of RECEIVE PARM String (Parameter Entry 10)`

If the user entered a character string on the PARM keyword of the RECEIVE command, this data field contains the address of that string.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

For more information about how you can use the RECEIVE PARM keyword, see "Installation-defined parameters" on page 400.

`Action Flags (Parameter Entry 11)`

The data field contains a word of action flags, which the exit can set to control RECEIVE processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit **Action**

`1... ....`
RECEIVE is to issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.

`.1.. ....`
If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes

message IKJ79154I to be issued; or requested that RECEIVE issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.

..11 ....
Reserved

.... 1...
RECEIVE is to bypass normal authorization checking and continue processing. The exit has validated the user ID in the parameter entry pointed to at offset +56.

.... .1..
The exit has determined that the user is not authorized to receive files for the user ID requested. RECEIVE processing ends.

.... ..11
Reserved

Message Text (Parameter Entry 12)
Exits can put in this data field the message text that RECEIVE is to issue with message ID INMR151I. The field initially contains blanks. If the exit inserts text in the data field, it must also set the key and length values to:

KEY:	X'00000001'
LEN:	Length of the message text. The maximum length is 243.

Status Flags (Parameter Entry 13)
The data field contains a word of status flags in which RECEIVE passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2 and 3 are reserved.

Bit **Meaning**

1... ....
The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command.

.111 ....
Reserved

.... 1...
The user ID in the parameter entry pointed to by offset +56 is the user ID specified on the USERID parameter of the RECEIVE command.

.... .111
Reserved

User ID of RECEIVE Issuer (Parameter Entry 14)
The data field contains the user ID of the person who issued RECEIVE. The user ID is taken from the PSCBUSER field of the protected step control block (PSCB).

User ID Receiving For (Parameter Entry 15)
If the person who issued RECEIVE specified the USERID parameter, this data field contains the user ID specified. If USERID was not specified, this data field contains the user ID of the person who issued RECEIVE (the same user ID as in the parameter entry pointed to at offset +52).

Parameter descriptions for INMRZ01

When INMRZ01 receives control, register 1 points to the following parameter list.

Offset	Length	Value
+0	4	Address of an exit-to-exit communication word. RECEIVE exits can use this word to communicate among themselves by passing information in the storage to which the it points. INMRZ01 can obtain the block of storage, and make it available to all exits by storing the address in the exit-to-exit communication word. RECEIVE initially sets the value of the word to zero, and thereafter, does not modify it or the contents of the storage to which it points.
+4	4	Address of the character string that the user entered on the PARM keyword of the RECEIVE command. The address points to an area consisting of a halfword length field followed by the character string. The length field specifies the length of the character string. If no string was specified, the value in the length field is zero. For more information about how you can use the RECEIVE PARM keyword, see “Installation-defined parameters” on page 400.
+8	4	Address of a byte of action flags, which the exit can set to control RECEIVE processing after it returns. **Bit** **Action** **1... ....** RECEIVE is to issue message INMR151I, using the text contained in the parameter pointed to at offset +16. **.1.. ....** If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either sends an appropriate message to the user, or requests that RECEIVE issue message INMR151I, using the text contained in the parameter pointed to at offset +16. **..11 ....** Reserved **.... 1...** RECEIVE is to bypass normal authorization checking and continue processing. The exit has validated the user ID at offset +28. **.... .1..** The exit has determined that the user is not authorized to receive files for the user ID requested. RECEIVE processing terminates. **.... ..11** Reserved
+12	4	Address of the TSO/E command processor parameter list (the CPPL). The exit can use the parameter list to examine the initial RECEIVE command, or to build an IOPL for invoking the TSO/E I/O service routines.

Offset	Length	Value
+16	4	Address of the message text that RECEIVE is to issue with message ID INMR151I. When the exit is invoked, the address points to an area consisting of a halfword length field, followed by an area in which the exit can insert message text. The length value at entry is zero. Exits that pass back message text must put the length of text into the length field. (Do not include the halfword of the length field in the length specification.) The maximum length is 243 bytes.
+20	4	Address of a status flag byte. RECEIVE uses this byte to pass indicators to the exit. **1... ....** The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command. **.111 ....** Reserved **.... 1...** The user ID pointed to at offset +28 is the user ID specified on the USERID parameter of the RECEIVE command. **.... .111** Reserved
+24	4	Address of an 8-character user ID field that contains the user ID of the person who issued RECEIVE. The user ID is taken from the PSCBUSER field of the protected step control block (PSCB).
+28	4	Address of an 8-character user ID field. The field contains either the user ID specified on the USERID parameter of RECEIVE, or, if the USERID parameter was not specified, the user ID of the person who issued RECEIVE (the same user ID pointed to at offset +24).

RECEIVE termination exit — INMRZ02 or INMRZ02R

The RECEIVE termination exit (INMRZ02 or INMRZ02R) receives control after all RECEIVE command processing is complete, or when the RECEIVE command ends abnormally or terminates because of an attention interrupt. The termination exit is invoked once per RECEIVE invocation. You can use the exit to perform clean-up processing, such as freeing storage obtained for the exit-to-exit communication word, closing any files that other RECEIVE exits opened, and freeing any data sets that they allocated.

Parameter descriptions for INMRZ02R

When INMRZ02R receives control, register 1 points to the standard exit parameter list. The parameter list is described in "TSO/E standard exit parameter list" on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMRZ02R receives beginning at offset +36 (decimal).

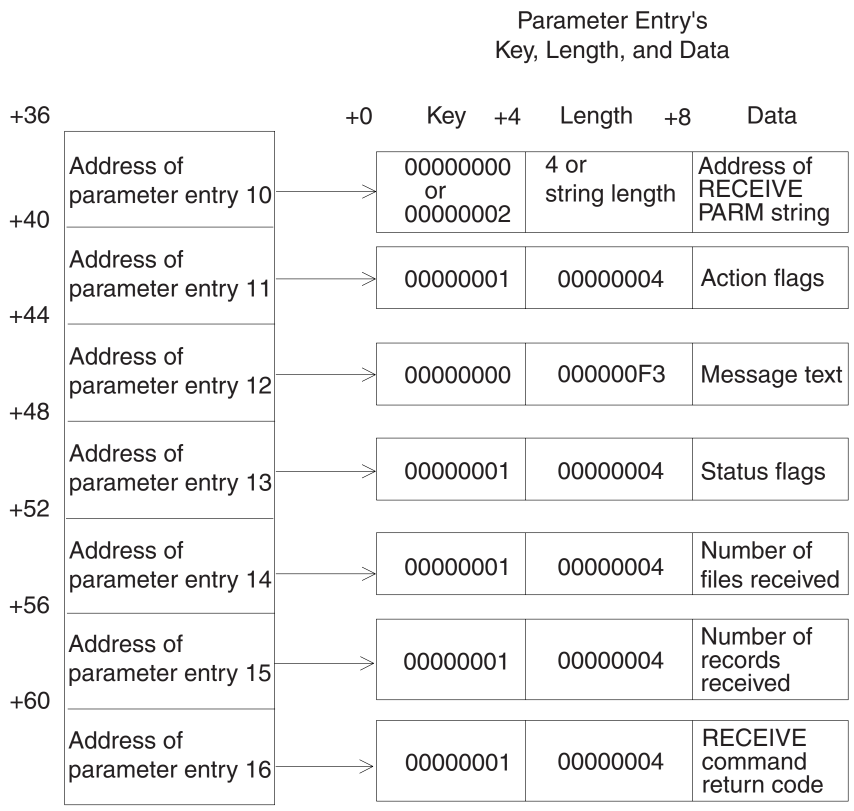

Parameter Entry's
Key, Length, and Data
+36
+40
+44
+48
+52
+56
+60
+0
Key
+4
Length
+8
Data
Address of parameter entry 10
Address of parameter entry 11
Address of parameter entry 12
Address of parameter entry 13
Address of parameter entry 14
Address of parameter entry 15
Address of parameter entry 16
00000000 or 00000002
4 or string length
Address of RECEIVE PARM string
00000001
00000004
Action flags
00000000
000000F3
Message text
00000001
00000004
Status flags
00000001
00000004
Number of files received
00000001
00000004
Number of records received
00000001
00000004
RECEIVE command return code

Figure 81. Exit-dependent data on entry to INMRZ02R

Parameter entries 10 and 12 (RECEIVE PARM string and message text for message ID INMR151I, respectively) are the same for all RECEIVE exits. For a description of those parameter entries, see "Parameter descriptions for INMRZ01R" on page 428. Following are descriptions of the other parameter entries:

Action Flags (Parameter Entry 11)

The data field contains a word of action flags, which the exit can set to control RECEIVE processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
1...	RECEIVE is to issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
.1..	If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that RECEIVE issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
..11 1111	Reserved

Status Flags (Parameter Entry 13)

The data field contains a word of status flags in which RECEIVE passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2 and 3 are reserved.

Bit **Meaning**

1... ....
The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command.

.111 1111
Reserved

Number of Files Received (Parameter Entry 14)
The data field contains the number of files received during this invocation of RECEIVE.

Number of Records Received (Parameter Entry 15)
The data field contains the number of records received during this invocation of RECEIVE.

RECEIVE Command Return Code (Parameter Entry 16)
The data field contains the RECEIVE command return code.

Parameter descriptions for INMRZ02

When INMRZ02 receives control, register 1 points to the following parameter list. Note that many of the parameters are the same as those that INMRZ01 receives. Explanations of those parameters are not repeated here. For more information about them, see "Parameter descriptions for INMRZ01" on page 431.

Offset	Length	Value
+0	4	Address of an exit-to-exit communication word.
+4	4	Address of the character string that the user entered on the PARM keyword of the RECEIVE command.
+8	4	Address of a byte of action flags, which the exit can set to control TRANSMIT processing after it returns. **Bit** **Action** **1... ....** RECEIVE is to issue message INMR151I, using the text contained in the parameter pointed to at offset +16. **.1.. ....** If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either sends an appropriate message to the user, or requests that RECEIVE issue message INMR151I, using the text contained in the parameter pointed to at offset +16. **..11 1111** Reserved
+12	4	Address of the TSO/E command processor parameter list (the CPPL).
+16	4	Address of the message text that RECEIVE is to issue with message ID INMR151I.

Offset	Length	Value
+20	4	Address of a status flag byte. RECEIVE uses this byte to pass indicators to the exit. **1... ....** The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command. **.111 1111** Reserved
+24	4	Address of a fullword that contains the number of files received during this invocation of the RECEIVE command.
+28	4	Address of a fullword that contains the number of records received during this invocation of RECEIVE.
+32	4	Address of a fullword that contains the RECEIVE command return code.

RECEIVE notification exit — INMRZ04 or INMRZ04R

The RECEIVE notification exit (INMRZ04 or INMRZ04R) receives control each time an acknowledgment is received. If the person who issued RECEIVE has several acknowledgments to receive, the exit is invoked multiple times during RECEIVE processing -- once per acknowledgment.

Installations can use this exit to process acknowledgment information. The TRANSMIT startup exit (INMXZ01 or INMXZ01R) can pass a notification string in the transmission. This exit and the RECEIVE acknowledgment notification exit receive the string. When RECEIVE transmits an acknowledgment to the originator of the transmission, the notification string is also included. This exit can use that notification string to tie transmissions to acknowledgments.

Parameter descriptions for INMRZ04R

When INMRZ04R receives control, register 1 points to the standard exit parameter list. The parameter list is described in “TSO/E standard exit parameter list” on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMRZ04R receives beginning at offset +36 (decimal).

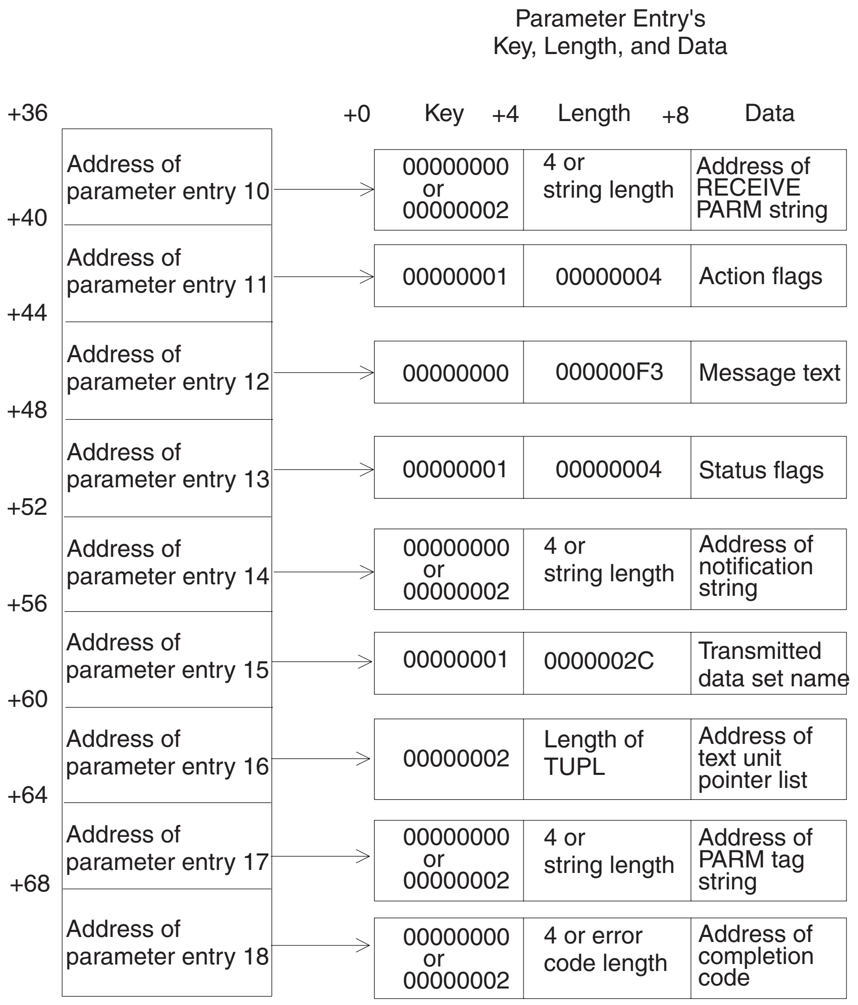

Parameter Entry's
Key, Length, and Data
+36
+40
+44
+48
+52
+56
+60
+64
+68
+0
Key
+4
Length
+8
Data
Address of parameter entry 10
Address of parameter entry 11
Address of parameter entry 12
Address of parameter entry 13
Address of parameter entry 14
Address of parameter entry 15
Address of parameter entry 16
Address of parameter entry 17
Address of parameter entry 18
00000000 or 00000002 | 4 or string length | Address of RECEIVE PARM string
00000001 | 00000004 | Action flags
00000000 | 000000F3 | Message text
00000001 | 00000004 | Status flags
00000000 or 00000002 | 4 or string length | Address of notification string
00000001 | 0000002C | Transmitted data set name
00000002 | Length of TUPL | Address of text unit pointer list
00000000 or 00000002 | 4 or string length | Address of PARM tag string
00000000 or 00000002 | 4 or error code length | Address of completion code

Figure 82. Exit-dependent data on entry to INMRZ04R

Parameter entries 10 and 12 (RECEIVE PARM string and message text for message ID INMR151I, respectively) are same for all RECEIVE exits. For a description of those parameter entries, see "Parameter descriptions for INMRZ01R" on page 428. Following are descriptions of the other parameter entries INMRZ04R receives:

Action Flags (Parameter Entry 11)

The data field contains a word of action flags, which the exit can set to control RECEIVE processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
`1... ....`	RECEIVE is to issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
`.1.. ....`	If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that RECEIVE issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.

..11 1111
: Reserved

Status Flags (Parameter Entry 13)
: The data field contains a word of status flags in which RECEIVE passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2 and 3 are reserved.

Bit	Meaning
1...	The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command.
.111 1111	Reserved

Address of User Notification String (Parameter Entry 14)
: If the TRANSMIT initialization exit (INMRZ01R or INMRZ01) passes a notification string, this data field contains the address of that string.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

For more information about how you can use the notification string, see "Installation-defined parameters" on page 400.

Transmitted Data Set Name (Parameter Entry 15)
: The name of the data set that was transmitted. If the data was transmitted as a message, the field contains the string, '**MESSAGE**'.

Address of Text Unit Pointer List (Parameter Entry 16)
: The data field contains the address of a text unit pointer list. The list points to *copies* of text units that are sent with the transmission. They identify the originator of the transmission, and can include:

```
INMFACK   INMFNODE  INMFTIME  INMFUID  INMFVERS  INMLRECL
INMNUMF   INMUSERP
```

For more information about those text units or text unit pointer lists, see "Text units and text unit pointer lists" on page 474. Note that altering the text units has no effect on subsequent processing. The exit receives copies of the actual text units for information only.

Address of PARM Tag String (Parameter Entry 17)
: The data field might contain the address of a string taken from the :PARM tag in the addressee's nickname entry in the sender's NAMES.TEXT data set. The string is present if the TRANSMIT command specified a nickname directly or via a distribution list, and the entry for that nickname in the sender's NAMES.TEXT data set contains the :PARM tag.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

Address of RECEIVE Completion Code (Parameter Entry 18)
The data field points to an area that can contain a RECEIVE completion code -- either an error code, or a string ('DELETED') indicating the receiver deleted the file.

If a completion code is set:

KEY: X'00000002'

LEN: Length of completion code

If no completion code is set:

KEY: X'00000000'

LEN: X'00000004'

Parameter descriptions for INMRZ04

When INMRZ04 receives control, register 1 points to the following parameter list. Note that many of the parameters are the same as those that INMRZ01 receives. Explanations of those parameters are not repeated here. For more information about them, see "Parameter descriptions for INMRZ01" on page 431.

Offset	Length	Value
+0	4	Address of an exit-to-exit communication word.
+4	4	Address of the character string that the user entered on the PARM keyword of the RECEIVE command.
+8	4	Address of a byte of action flags, which the exit can set to control RECEIVE processing after the exit returns. **Bit** **Action** **1... ....** RECEIVE is to issue message INMR151I, using the text contained in the parameter pointed to at offset +16. **.1.. ....** If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either sends an appropriate message to the user, or requests that RECEIVE issue message INMR151I, using the text contained in the parameter pointed to at offset +16. **..11 1111** Reserved
+12	4	Address of the TSO/E command processor parameter list (the CPPL).
+16	4	Address of the message text that RECEIVE is to issue with message ID INMR151I.

Offset	Length	Value
+20	4	Address of a status flag byte. RECEIVE uses this byte to pass indicators to the exit. **1... ....** The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command. **.111 1111** Reserved
+24	4	If the TRANSMIT initialization exit (INMRZ01R or INMRZ01) passes a notification string, this field contains the address of that string. The address points to an area that consists of a halfword length field followed by the character string. The length value is the length of the character string only. If no string is specified, the value in the length field is zero.
+28	4	Address of a 44-character field that contains the name of the data set that was transmitted. If the data was transmitted as a message, the field contains the string, '***MESSAGE***'.

+32	4	Address of a text unit pointer list. The list points to *copies* of text units that are sent with the transmission. They identify the originator of the transmission, and can include:
		
		`INMFACK   INMFNODE  INMFTIME  INMFUID`
		`INMFVERS  INMLRECL  INMNUMF   INMUSERP`
		
		For more information about those text units or text unit pointer lists, see "Text units and text unit pointer lists" on page 474. Note that altering the text units has no effect on subsequent processing. The exit receives copies of the actual text units for information only.

+36	4	Address of the string taken from the :PARM tag in the addressee's nickname entry in the sender's NAMES.TEXT data set. The string is present if the TRANSMIT command specified a nickname directly or via a distribution list, and the entry for that nickname in the sender's NAMES.TEXT data set contains the :PARM tag. The address points to an area that consists of a halfword length field followed by the character string. The length value is the length of the character string only. If no string is specified, the value in the length field is zero.
+40	4	Address of an area that contains the RECEIVE completion code -- either an error code, or a string ('DELETED') the receiver deleted the file. The area consists of a halfword length field followed by the completion code. The length value is the length of the code only, and does not include the halfword length field. If the RECEIVE command was successful, the value in the length field is zero.

RECEIVE acknowledgment notification exit — INMRZ05R

If the sender of data requested acknowledgment, the RECEIVE acknowledgment notification exit (INMRZ05R) receives control after the acknowledgment has been sent. However, this exit does not receive control if exit INMRZ06R has suppressed the acknowledgment. If RECEIVE is processing several files, each with a request for acknowledgment, this exit is invoked multiple times during RECEIVE processing. The exit enables installations to notify the sender that an acknowledgment is waiting to be received by the sender. The sender can then receive the acknowledgment and free spool space.

Note: If RACF is installed, your installation can activate security label checking. If your installation has activated security label checking, and the result of that check indicates the receiver is at a different security label than the sender is logged on with, exit INMRZ05R is invoked. However, the sender may not get the acknowledgment.

Parameter descriptions for INMRZ05R

When INMRZ05R receives control, register 1 points to the standard exit parameter list. The parameter list is described in “TSO/E standard exit parameter list” on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMRZ05R receives beginning at offset +36 (decimal).

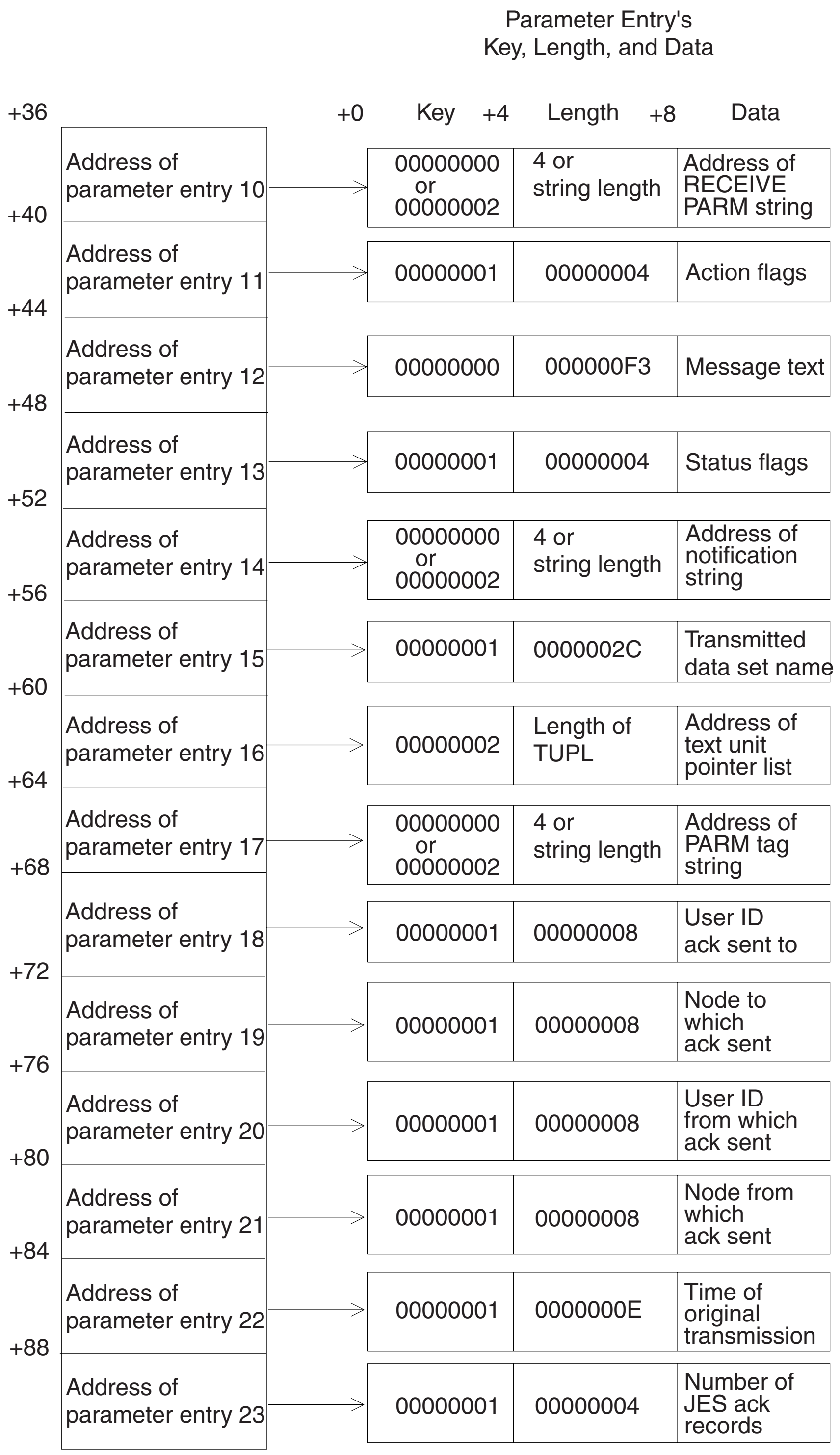

Parameter Entry's
Key, Length, and Data
+36
+0
Key
+4
Length
+8
Data
Address of parameter entry 10
00000000 or 00000002
4 or string length
Address of RECEIVE PARM string
+40
Address of parameter entry 11
00000001
00000004
Action flags
+44
Address of parameter entry 12
00000000
000000F3
Message text
+48
Address of parameter entry 13
00000001
00000004
Status flags
+52
Address of parameter entry 14
00000000 or 00000002
4 or string length
Address of notification string
+56
Address of parameter entry 15
00000001
0000002C
Transmitted data set name
+60
Address of parameter entry 16
00000002
Length of TUPL
Address of text unit pointer list
+64
Address of parameter entry 17
00000000 or 00000002
4 or string length
Address of PARM tag string
+68
Address of parameter entry 18
00000001
00000008
User ID ack sent to
+72
Address of parameter entry 19
00000001
00000008
Node to which ack sent
+76
Address of parameter entry 20
00000001
00000008
User ID from which ack sent
+80
Address of parameter entry 21
00000001
00000008
Node from which ack sent
+84
Address of parameter entry 22
00000001
0000000E
Time of original transmission
+88
Address of parameter entry 23
00000001
00000004
Number of JES ack records

Figure 83. Exit-dependent data on entry to INMRZ05R

Parameter entries 10 and 12 (RECEIVE PARM string and message text for message ID INMR151I, respectively) are the same for all RECEIVE exits. For a description of those parameter entries, see “Parameter descriptions for INMRZ01R” on page 428. Following are descriptions of the other parameter entries:

Action Flags (Parameter Entry 11)
The data field contains a word of action flags, which the exit can set to control RECEIVE processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
`1... ....`	RECEIVE is to issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
`.1.. ....`	If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that RECEIVE issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
`..11 1111`	Reserved

Status Flags (Parameter Entry 13)
The data field contains a word of status flags in which RECEIVE passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2 and 3 are reserved.

Bit	Meaning
`1... ....`	The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command.
`.111 1111`	Reserved

Address of User Notification String (Parameter Entry 14)
If the TRANSMIT initialization exit (INMRZ01R or INMRZ01) passes a notification string in the transmission, this data field contains the address of that string.

If a string was specified:

KEY:	X'00000002'
LEN:	Length of the string

If no string was specified:

KEY:	X'00000000'
LEN:	X'00000004'

For more information about how you can use the notification string, see “Installation-defined parameters” on page 400.

Transmitted Data Set Name (Parameter Entry 15)
The name of the data set that was transmitted. If the data was transmitted as a message, the field contains the string, ‘***MESSAGE*** ’.

Address of Text Unit Pointer List (Parameter Entry 16)
The data field contains the address of a text unit pointer list. The list points to *copies* of text units that are sent with the transmission. They identify the originator of the transmission, and can include:

```
INMFACK   INMFNODE  INMFTIME  INMFUID  INMFVERS  INMLRECL
INMNUMF   INMUSERP
```

For more information about those text units or text unit pointer lists, see "Text units and text unit pointer lists" on page 474. Note that altering the text units has no effect on subsequent processing. The exit receives copies of the actual text units for information only.

Address of PARM Tag String (Parameter Entry 17)
The data field might contain the address of a string taken from the :PARM tag in the addressee's nickname entry in the sender's NAMES.TEXT data set. The string is present if the TRANSMIT command specified a nickname directly or via a distribution list, and the entry for that nickname in the sender's NAMES.TEXT data set contains the :PARM tag.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

User ID to Which Acknowledgment is Sent (Parameter Entry 18)
The data field contains the user ID to which the acknowledgment is to be sent. Because the acknowledgment is sent before this exit gets control, RECEIVE ignores any changes the exit makes to this data field.

Node to Which Acknowledgment is Sent (Parameter Entry 19)
The data field contains the node to which the acknowledgment is to be sent. Because the acknowledgment is sent before this exit gets control, RECEIVE ignores any changes the exit makes to this data field.

User ID from Which Acknowledgment is Sent (Parameter Entry 20)
The data field contains the user ID from which the acknowledgment is to be sent. RECEIVE ignores any changes the exit makes to this field.

Node from Which Acknowledgment is Sent (Parameter Entry 21)
The data field contains the node from which the acknowledgment is to be sent. RECEIVE ignores any changes the exit makes to this field.

Time of Original Transmission (Parameter Entry 22)
The data field contains the timestamp of when the original transmission occurred. The time is in standard GMT format. RECEIVE ignores any changes the exit makes to this field.

Number of JES OUTPUT Records (Parameter Entry 23)
The data field contains the number of JES output records transmitted in the acknowledgment. RECEIVE ignores any changes the exit makes to this field.

RECEIVE pre-acknowledgment notification exit — INMRZ06R

If the sender of data requested acknowledgment, the RECEIVE pre-acknowledgment notification exit (INMRZ06R) receives control before the acknowledgment is sent. The exit enables installations to suppress acknowledgments and alter sending and receiving userid/node pairs. If the SECLABELs of the sender and receiver are different, the installation can specify which SECLABEL to use to send the acknowledgment.

If RECEIVE is processing several files, each with an acknowledgment, this exit is invoked multiple times during RECEIVE processing.

Parameter descriptions for INMRZ06R

When INMRZ06R receives control, register 1 points to the standard exit parameter list. The parameter list is described in "TSO/E standard exit parameter list" on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMRZ06R receives beginning at offset +36 (decimal).

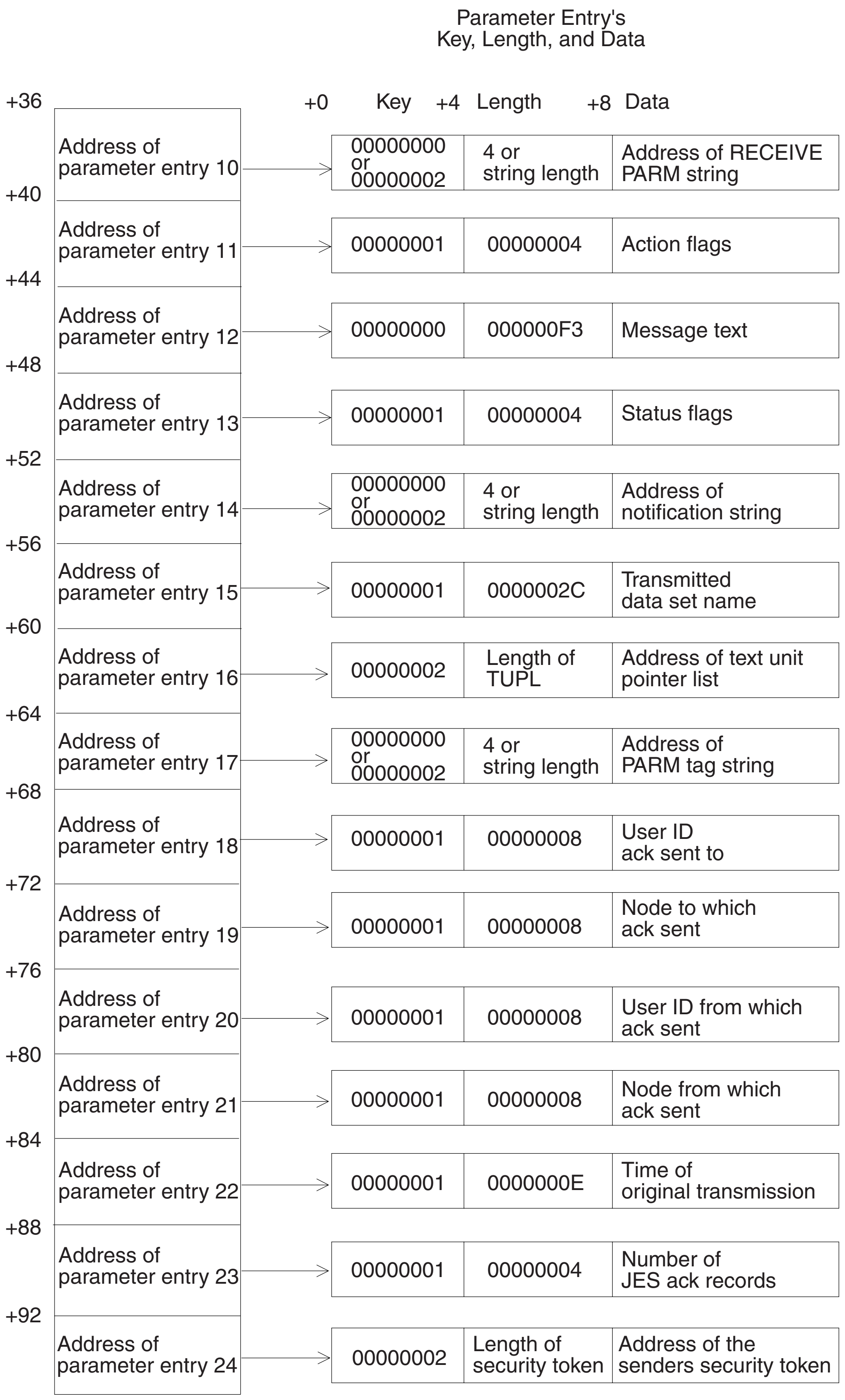

Parameter Entry's
Key, Length, and Data
+36
+40
+44
+48
+52
+56
+60
+64
+68
+72
+76
+80
+84
+88
+92
+0 Key +4 Length +8 Data
Address of parameter entry 10 → 00000000 or 00000002 | 4 or string length | Address of RECEIVE PARM string
Address of parameter entry 11 → 00000001 | 00000004 | Action flags
Address of parameter entry 12 → 00000000 | 000000F3 | Message text
Address of parameter entry 13 → 00000001 | 00000004 | Status flags
Address of parameter entry 14 → 00000000 or 00000002 | 4 or string length | Address of notification string
Address of parameter entry 15 → 00000001 | 0000002C | Transmitted data set name
Address of parameter entry 16 → 00000002 | Length of TUPL | Address of text unit pointer list
Address of parameter entry 17 → 00000000 or 00000002 | 4 or string length | Address of PARM tag string
Address of parameter entry 18 → 00000001 | 00000008 | User ID ack sent to
Address of parameter entry 19 → 00000001 | 00000008 | Node to which ack sent
Address of parameter entry 20 → 00000001 | 00000008 | User ID from which ack sent
Address of parameter entry 21 → 00000001 | 00000008 | Node from which ack sent
Address of parameter entry 22 → 00000001 | 0000000E | Time of original transmission
Address of parameter entry 23 → 00000001 | 00000004 | Number of JES ack records
Address of parameter entry 24 → 00000002 | Length of security token | Address of the senders security token

Figure 84. Exit-dependent data on entry to INMRZ06R

Parameter entries 10 and 12 (RECEIVE PARM string and message text for message ID INMR151I, respectively) are the same for all RECEIVE exits. For a description of those parameter entries, see “Parameter descriptions for INMRZ01R” on page 428. Following are descriptions of the other parameter entries:

Action Flags (Parameter Entry 11)

The data field contains a word of action flags, which the exit can set to control RECEIVE processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
`1... ....`	RECEIVE is to issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
`.1.. ....`	If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that RECEIVE issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
`..11 ....`	Reserved
`.... 1...`	RECEIVE is to suppress acknowledgment processing. No acknowledgment is to be sent to the requesting user, and exit INMRZ05R should not be invoked. The exit has performed acknowledgment processing itself or determined that an acknowledgment was not necessary.
`.... .1..`	RECEIVE is to use the receiver's SECLABEL for the acknowledgment message when this bit is on; otherwise, the sender's SECLABEL will be used for the acknowledgment message.
`.... ..11`	Reserved

Status Flags (Parameter Entry 13)

The data field contains a word of status flags in which RECEIVE passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2 and 3 are reserved.

Bit	Meaning
`1... ....`	The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command.
`.111 1111`	Reserved

Address of User Notification String (Parameter Entry 14)

If the TRANSMIT initialization exit (INMRZ01R or INMRZ01) passes a notification string in the transmission, this data field contains the address of that string.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

For more information about how you can use the notification string, see "Installation-defined parameters" on page 400.

Transmitted Data Set Name (Parameter Entry 15)
The name of the data set that was transmitted. If the data was transmitted as a message, the field contains the string, '***MESSAGE*** '.

Address of Text Unit Pointer List (Parameter Entry 16)
The data field contains the address of a text unit pointer list. The list points to *copies* of text units that are sent with the transmission. They identify the originator of the transmission, and can include:

```
INMFACK   INMFNODE  INMFTIME  INMFUID  INMFVERS  INMLRECL
INMNUMF   INMUSERP
```

For more information about those text units or text unit pointer lists, see "Text units and text unit pointer lists" on page 474. Note that altering the text units has no effect on subsequent processing. The exit receives copies of the actual text units for information only.

Address of PARM Tag String (Parameter Entry 17)
The data field might contain the address of a string taken from the :PARM tag in the addressee's nickname entry in the sender's NAMES.TEXT data set. The string is present if the TRANSMIT command specified a nickname directly or via a distribution list, and the entry for that nickname in the sender's NAMES.TEXT data set contains the :PARM tag.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

User ID to Which Acknowledgment is Sent (Parameter Entry 18)
The data field contains the user ID to which the acknowledgment is to be sent. This parameter may be modified by the exit. Any modification made to this parameter affects the current acknowledgment only. Changes are passed along in the acknowledgment transmission, and are also passed to exit INMRZ05R.

Node to Which Acknowledgment is Sent (Parameter Entry 19)
The data field contains the node to which the acknowledgment is to be sent. This parameter may be modified by the exit. Any modification made to this parameter affects the current acknowledgment only. Changes are passed along in the acknowledgment transmission, and are also passed to exit INMRZ05R.

User ID from Which Acknowledgment is Sent (Parameter Entry 20)
The data field contains the user ID from which the acknowledgment is to be sent. This parameter may be modified by the exit. Any modification made to this parameter affects the current acknowledgment only. Changes are passed along in the acknowledgment transmission, and are also passed to exit INMRZ05R.

Node from Which Acknowledgment is Sent (Parameter Entry 21)
The data field contains the node from which the acknowledgment is to be sent.

This parameter may be modified by the exit. Any modification made to this parameter affects the current acknowledgment only. Changes are passed along in the acknowledgment transmission, and are also passed to exit INMRZ05R.

Time of Original Transmission (Parameter Entry 22)
The data field contains the timestamp of when the original transmission occurred. The time is in standard GMT format. RECEIVE ignores any changes the exit makes to this field.

Number of JES OUTPUT Records (Parameter Entry 23)
The data field contains the number of JES output records transmitted in the acknowledgment. The exit can update the data field to reflect the number of records transmitted if the exit performed acknowledgment processing.

Address of the Sender's Security Token (Parameter Entry 24)
The data field contains the address of the sender's encrypted security token. The exit can decrypt the security token using the RACF TOKENMAP macro. For complete information about using the TOKENMAP macro, see *z/OS Security Server RACF Macros and Interfaces*.

RECEIVE data set pre-processing exit — INMRZ11 or INMRZ11R

The RECEIVE data set pre-processing exit (INMRZ11 or INMRZ11R) receives control just before the RECEIVE command processor prompts the user for actions to be taken with the transmitted data set. You can use the exit to:

- Replace user interaction required in determining what to do with the arriving transmission. The exit receives information about the incoming data set and can specify the same parameters for which RECEIVE would otherwise prompt the user.
- Determine which users can use a particular network path. The exit can check the source of a message before delivering it to the user.
- In conjunction with the RECEIVE data set post-processing exit (INMRZ12R or INMRZ12), receive data set types not supported by TRANSMIT and RECEIVE. This exit instructs RECEIVE to ignore the target dsname that the sender specified, and to write the received data into a temporary data set that INMXZ01R or another exit allocated.
- Suppress reception of data sets by setting the action flag that tells RECEIVE all processing on the current file is complete.

Parameter descriptions for INMRZ11R

When INMRZ11R receives control, register 1 points to the standard exit parameter list. The parameter list is described in "TSO/E standard exit parameter list" on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMRZ11R receives beginning at offset +36 (decimal).

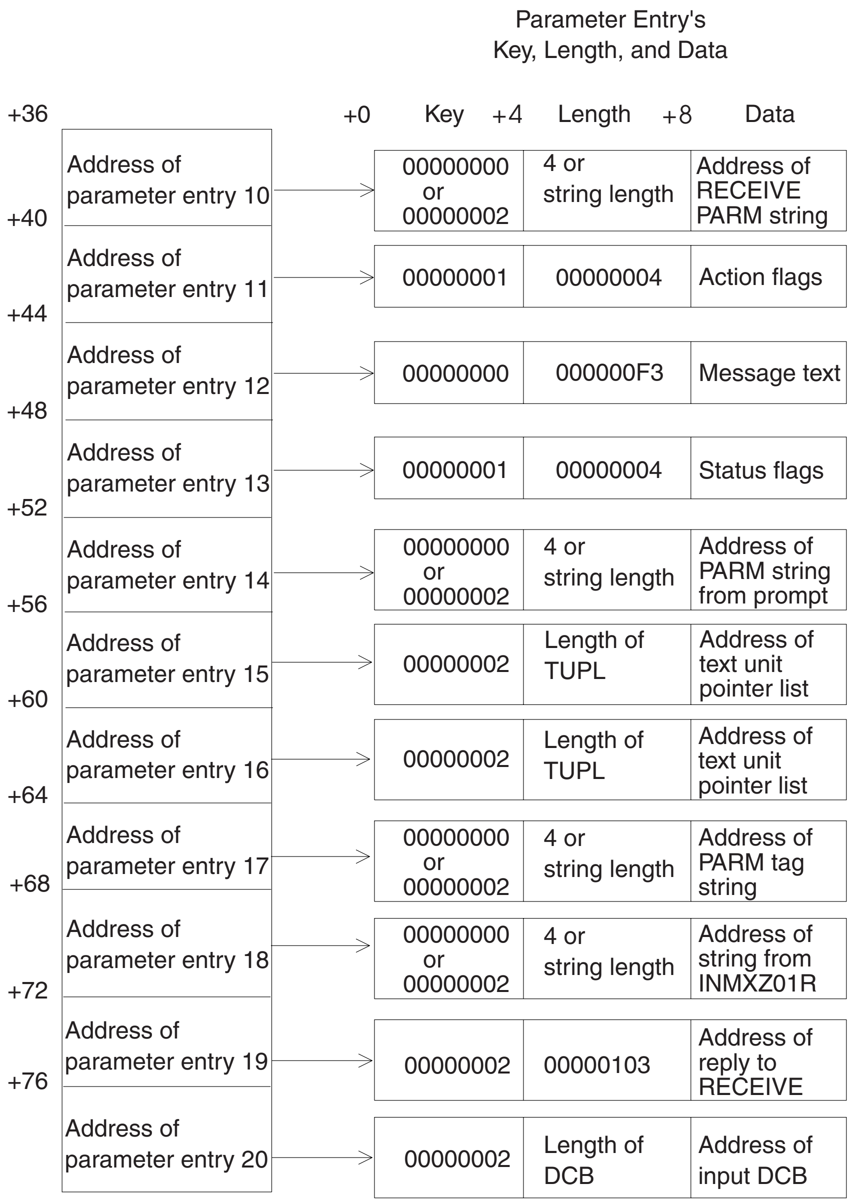

Parameter Entry's
Key, Length, and Data
+36
+40
+44
+48
+52
+56
+60
+64
+68
+72
+76
+0
Key
+4
Length
+8
Data
Address of parameter entry 10
Address of parameter entry 11
Address of parameter entry 12
Address of parameter entry 13
Address of parameter entry 14
Address of parameter entry 15
Address of parameter entry 16
Address of parameter entry 17
Address of parameter entry 18
Address of parameter entry 19
Address of parameter entry 20
00000000 or 00000002 | 4 or string length | Address of RECEIVE PARM string
00000001 | 00000004 | Action flags
00000000 | 000000F3 | Message text
00000001 | 00000004 | Status flags
00000000 or 00000002 | 4 or string length | Address of PARM string from prompt
00000002 | Length of TUPL | Address of text unit pointer list
00000002 | Length of TUPL | Address of text unit pointer list
00000000 or 00000002 | 4 or string length | Address of PARM tag string
00000000 or 00000002 | 4 or string length | Address of string from INMXZ01R
00000002 | 00000103 | Address of reply to RECEIVE
00000002 | Length of DCB | Address of input DCB

Figure 85. Exit-dependent data on entry to INMRZ11R

Parameter entries 10 and 12 (RECEIVE PARM string and message text for message ID INMR151I, respectively) are the same for all RECEIVE exits. For a description of those parameter entries, see “Parameter descriptions for INMRZ01R” on page 428.

Following are descriptions of the other parameter entries:

Action Flags (Parameter Entry 11)

The data field contains a word of action flags, which the exit can set to control RECEIVE processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
`1... ....`	RECEIVE is to issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.

.1.. ....
If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that RECEIVE issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.

..11 ....
Reserved

.... 1...
RECEIVE is not to prompt the user. Instead, it is to use the reply the exit specifies in the parameter entry pointed to at offset +72. If bit X'01' is specified, RECEIVE ignores it.

.... .1..
RECEIVE is to restore the incoming data to a temporary data set. The RECEIVE data set post-processing exit (INMRZ12R or INMRZ12) should complete processing the data.

Setting this bit to one does not prevent RECEIVE from prompting the user for allocation parameters to use in restoring the data; it only causes RECEIVE to ignore the user's reply. Thus, if you set this bit to one, you should also set bit X'08' to one, to prevent RECEIVE from prompting the user.

.... ..1.
All processing on the current file is complete. If the return code from the exit is zero, no more processing is done on the file. RECEIVE can delete the file from the JES spool and process the next file, if one exists.

If the return code is non-zero, RECEIVE ignores this bit.

....1
RECEIVE is to prompt the user for allocation parameters, and not take the action that RESTORE requests unless the user requests it. (RESTORE is the usual default action when processing data sets.) If action bit X'08' is one, RECEIVE ignores this bit.

Status Flags (Parameter Entry 13)
The data field contains a word of status flags in which RECEIVE passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2 and 3 are reserved.

Bit **Meaning**

1... ....
The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command.

.1.. ....
RECEIVE does not recognize the input data. Either an error occurred in the transmission, or the data was sent by a program other than TRANSMIT or PROFS.

If the exit is to process the data, it can get the address of the input DCB from the parameter entry pointed to at offset +76. The exit can also get the ddname of the file via the text unit pointer list in the parameter entry pointed to at offset +56. Before processing the data, the exit must close then reopen the data set to ensure that it starts reading at the beginning. (When the exit receives control, RECEIVE has already read past any control records, and has read at least the first record of the data set.)

..11 ....
Reserved

.... 1...
RECEIVE is to restore the data to its original format and write it to the appropriate log, which is the action that RESTORE(LOG) requests, and the default when receiving messages.

.... .111
Reserved

Address of PARM String from user–entered PARM keyword (Parameter Entry 14)
If the user entered a character string on the PARM keyword on the RECEIVE command, this data field contains the address of that string.

If a string was specified:

KEY:	X'00000002'
LEN:	Length of the string

If no string was specified:

KEY:	X'00000000'
LEN:	X'00000004'

Address of Text Unit Pointer List (Parameter Entry 15)
The data field contains the address of a text unit pointer list. The list points to *copies* of text units that are sent with the transmission. They describe the incoming data, and can include:

```
INMBLKSZ  INMCREAT  INMDDNAM  INMDIR   INMDSNAM  INMDSORG
INMEXPDT  INMFM     INMLCHG   INMLRECL INMLREF   INMMEMBR
INMRECFM  INMSERP   INMSIZE   INMTERM  INMUTILN
```

For more information about those text units or text unit pointer lists, see "Text units and text unit pointer lists" on page 474. Note that altering the text units has no effect on subsequent processing. The exit receives copies of the actual text units for information only.

Address of Text Unit Pointer List (Parameter Entry 16)
The data field contains the address of another text unit pointer list. The list points to *copies* of text units that are sent with the transmission. They identify the originator of the transmission, and can include:

```
INMFACK   INMFNODE  INMFTIME  INMFUID  INMFVERS  INMLRECL
INMNUMF   INMUSERP
```

For more information about those text units or text unit pointer lists, see "Text units and text unit pointer lists" on page 474. Note that altering the text units has no effect on subsequent processing. The exit receives copies of the actual text units for information only.

Address of PARM Tag String (Parameter Entry 17)
The data field might contain the address of a string taken from the :PARM tag in the addressee's nickname entry in the sender's NAMES.TEXT data set. The string is present if the TRANSMIT command specified a nickname directly or via a distribution list, and the entry for that nickname in the sender's NAMES.TEXT data set contains the :PARM tag.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

Address of a String from INMXZ01R (Parameter Entry 18)

The TRANSMIT initialization exit (INMXZ01R or INMXZ01) can pass a user string to the RECEIVE data set exits, including INMRZ11R. If it does, this data field contains the address of that string.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

Address of Reply to RECEIVE (Parameter Entry 19)

The data field contains the address of an area in which the exit can build a reply to the RECEIVE command prompt. The address points to an area that contains: a halfword length, followed by a halfword of zeros, followed by 255 bytes (X'FF') where the exit can supply the prompt data. At entry, the length field contains 259 bytes (X'0103'). This value denotes the maximum length of the buffer. To supply a receive prompt, the exit must do the following:

1. Set bit X'08' in the action flags (parameter entry 11) on to indicate that RECEIVE is not to prompt the user.
2. Supply the data for the RECEIVE reply in the 255-byte (X'FF') area.
3. Set the halfword length field in the header to the length of the reply that was supplied, plus a 4-byte header.

If the exit builds a reply, it also needs to put the following values in the key and length fields:

KEY: X'00000002'

LEN: Length of the replay plus a 4-byte header

Address of the Input DCB (Parameter Entry 20)

The data field contains the address of the input DCB that identifies the data set being received. Before reading the data, the exit must close then reopen the data set to ensure that it starts reading at the beginning. (When the exit receives control, RECEIVE has already read past any control records, and has read at least the first record of the data set.)

Parameter descriptions for INMRZ11

When INMRZ11 receives control, register 1 points to the following parameter list. Note that many of the parameters are the same as those that INMRZ01 receives. Explanations of those parameters are not repeated here. For more information about them, see "Parameter descriptions for INMRZ01" on page 431.

Offset	Length	Value
+0	4	Address of an exit-to-exit communication word.
+4	4	Address of the character string that the user entered on the PARM keyword of the RECEIVE command.
+8	4	Address of a byte of action flags, which the exit can set to control RECEIVE processing after it returns. **Bit** **Action** **`1... ....`** RECEIVE is to issue message INMR151I, using the text contained in the parameter pointed to at offset +16. **`.1.. ....`** If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either sends an appropriate message to the user, or requests that RECEIVE issue message INMR151I, using the text contained in the parameter pointed to at offset +16. **`..11 ....`** Reserved **`.... 1...`** RECEIVE is not to prompt the user. Instead, it is to use the reply the exit specifies in the area pointed to at offset +44. If bit X'01' is set to one, RECEIVE ignores it. **`.... .1..`** RECEIVE is to restore the incoming data to a temporary data set. The RECEIVE data set post-processing exit (INMRZ12R or INMRZ12) should complete processing the data. Setting this bit to one does not prevent RECEIVE from prompting the user for allocation parameters to use in restoring the data; it only causes RECEIVE to ignore the user's reply. Thus, if you set this bit to one, you should also set bit X'08' to one, to prevent RECEIVE from prompting the user. **`.... ..1.`** All processing on the current file is complete. If the return code from the exit is zero, no more processing is done on the file. RECEIVE can delete the file from the JES spool and process the next file, if one exists. If the return code is non-zero, RECEIVE ignores this bit. **`.... ...1`** RECEIVE is to prompt the user for allocation parameters, and not take the action that RESTORE requests unless the user requests it. (RESTORE is the usual default action when processing data sets.) If action bit X'08' is one, RECEIVE ignores this bit.
+12	4	Address of the TSO/E command processor parameter list (the CPPL).
+16	4	Address of the message text that RECEIVE is to issue with message ID INMR151I.

Offset	Length	Value
+20	4	Address of a status flag byte. RECEIVE uses this byte to pass indicators to the exit. **1... ....** The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command. **.1.. ....** RECEIVE does not recognize the input data. Either an error occurred in the transmission, or the data was sent by a program other than TRANSMIT or PROFS. If the exit is to process the data, it can get the address of the input DCB from offset +48. The exit can also get the ddname of the file via the text unit pointer list at offset +28. Before processing the data, the exit must close then reopen the data set to ensure that it starts reading at the beginning. (When the exit receives control, RECEIVE has already read past any control records, and has read at least the first record of the data set.) **..11 ....** Reserved **.... 1...** RECEIVE is to restore the data to its original format and write it to the appropriate log, which is the action that RESTORE(LOG) requests, and the default when receiving messages. **.... .111** Reserved
+24	4	If the user entered a character string on the PARM keyword in response to the RECEIVE prompt, this parameter contains the address of that string. The address points to an area that consists of a halfword length field followed by the character string. The length value is the length of the character string only. If no string is specified, the length value is zero.

+28	4	Address of a text unit pointer list. The list points to *copies* of text units that are sent with the transmission. They describe the incoming data, and can include:
		
		`INMBLKSZ  INMCREAT  INMDDNAM  INMDIR   INMDSNAM`
		`INMDSORG  INMFM     INMEXPDT  INMLCHG  INMLRECL`
		`INMLREF   INMMEMBR  INMRECFM  INMSIZE  INMTERM`
		`INMSERP   INMUTILN`
		
		For more information about those text units or text unit pointer lists, see “Text units and text unit pointer lists” on page 474. Note that altering the text units has no effect on subsequent processing. The exit receives copies of the actual text units for information only.

Offset	Length	Value
+32	4	Address of another text unit pointer list. The list points to *copies* of text units that are sent with the transmission. They identify the originator of the transmission, and can include: `INMFACK INMFNODE INMFTIME INMFUID INMFVERS INMLRECL INMNUMF INMUSERP` For more information about those text units or text unit pointer lists, see “Text units and text unit pointer lists” on page 474. Note that altering the text units has no effect on subsequent processing. The exit receives copies of the actual text units for information only.
+36	4	Address of a string taken from the :PARM tag in the addressee's nickname entry in the sender's NAMES.TEXT data set. The string is present if the TRANSMIT command specified a nickname directly or via a distribution list, and the entry for that nickname in the sender's NAMES.TEXT data set contains the :PARM tag. The address points to an area that consists of a halfword length field followed by the character string. The length value is the length of the character string only. If no string is specified, the value in the length field is zero.
+40	4	The TRANSMIT initialization exit (INMXZ01R or INMXZ01) can pass a user string to the RECEIVE data set exits, including INMRZ11. If it does, this data field contains the address of that string. The address points to an area that consists of a halfword length field followed by the character string. The length value is the length of the character string only. If no string is specified, the value in the length field is zero.
+44	4	The address of an area in which the exit can build a reply to the RECEIVE command prompt. The address points to an area that consists of a halfword length field followed by a halfword of zeros, followed by 255-bytes (X'FF') where the exit can supply the prompt data. At entry the length field contains 259 bytes (X'0103'). This value denotes the maximum length of the buffer. To supply a receive prompt, the exit must do the following: 1. Set bit X'08' in the action flags (parameter entry 11) on to indicate that RECEIVE is not to prompt the user. 2. Supply the data for the RECEIVE reply in the 255-byte (X'FF') area. 3. Set the halfword length field in the header to the length of the reply that was supplied, plus a 4-byte header.
+48	4	Address of the input DCB that identifies the data set being received. Before reading the data, the exit must close then reopen the data set to ensure that it starts reading at the beginning. (When the exit receives control, RECEIVE has already read past any control records, and has read at least the first record of the data set.)

RECEIVE data set post-processing exit — INMRZ12 or INMRZ12R

The RECEIVE data set post-processing exit (INMRZ12 or INMRZ12R) receives control after all RECEIVE processing for a data set is completed except for deleting the input file. You can use the exit:

- In conjunction with either the RECEIVE data set pre-processing exit (INMRZ11R or INMRZ11) or the RECEIVE post-prompt exit (INMRZ15R) to receive data set types not supported by TRANSMIT and RECEIVE. The exit invokes the utility you specify to restore the data to its original format and write it into the receiving user's target data set.
- Record network use. The exit, as well as exits INMXZ02R and INMXZ02, has access to the information required to determine the volume and direction of network traffic. You can use SMFWTM or SMFEWTM macros to write SMF records from the exit.

Parameter descriptions for INMRZ12R

When INMRZ12R receives control, register 1 points to the standard exit parameter list. The parameter list is described in "TSO/E standard exit parameter list" on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMRZ11R receives beginning at offset +36 (decimal).

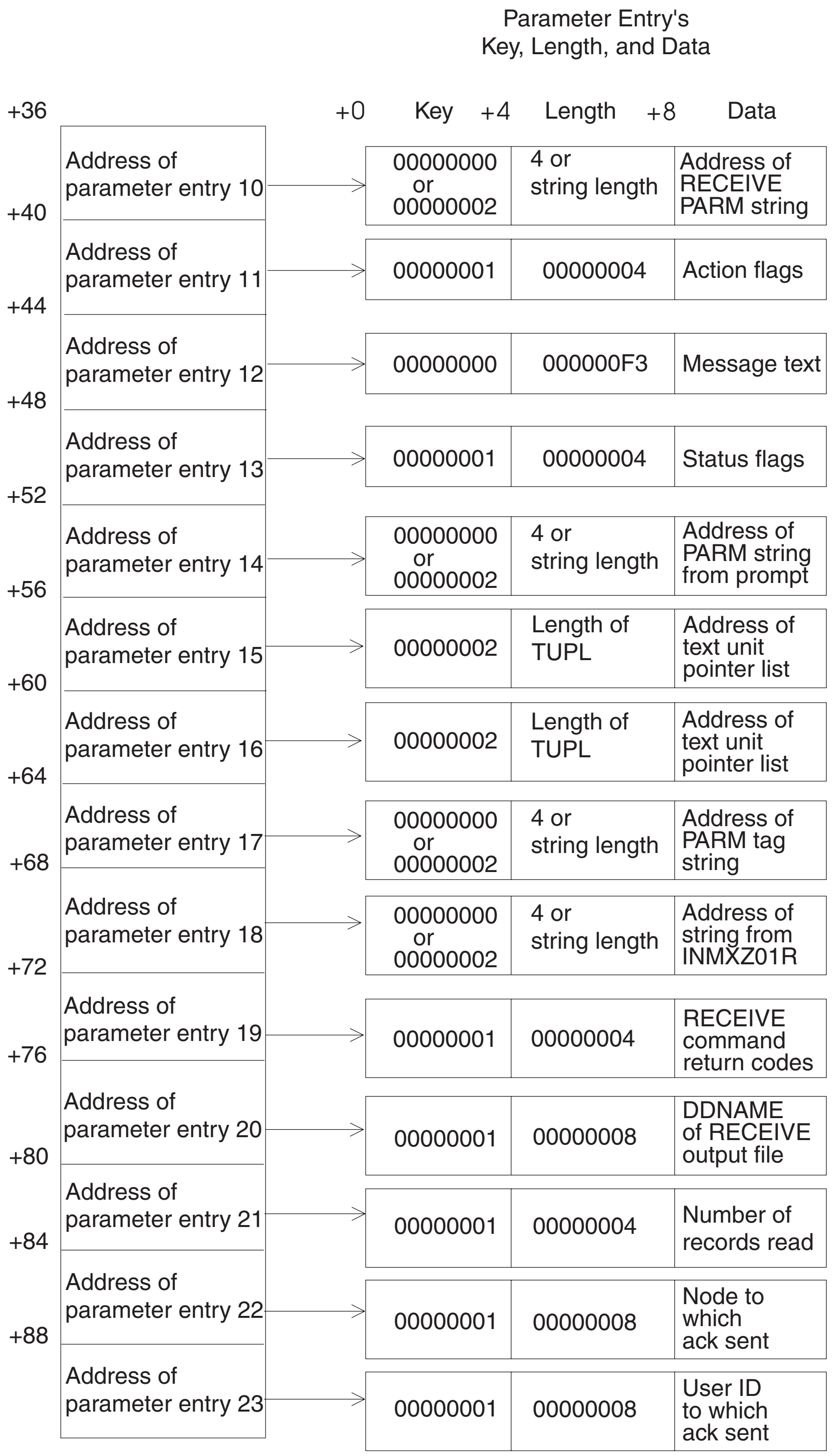

Parameter Entry's
Key, Length, and Data
+36
+40
+44
+48
+52
+56
+60
+64
+68
+72
+76
+80
+84
+88
+0
Key
+4
Length
+8
Data
Address of parameter entry 10
00000000 or 00000002
4 or string length
Address of RECEIVE PARM string
Address of parameter entry 11
00000001
00000004
Action flags
Address of parameter entry 12
00000000
000000F3
Message text
Address of parameter entry 13
00000001
00000004
Status flags
Address of parameter entry 14
00000000 or 00000002
4 or string length
Address of PARM string from prompt
Address of parameter entry 15
00000002
Length of TUPL
Address of text unit pointer list
Address of parameter entry 16
00000002
Length of TUPL
Address of text unit pointer list
Address of parameter entry 17
00000000 or 00000002
4 or string length
Address of PARM tag string
Address of parameter entry 18
00000000 or 00000002
4 or string length
Address of string from INMXZ01R
Address of parameter entry 19
00000001
00000004
RECEIVE command return codes
Address of parameter entry 20
00000001
00000008
DDNAME of RECEIVE output file
Address of parameter entry 21
00000001
00000004
Number of records read
Address of parameter entry 22
00000001
00000008
Node to which ack sent
Address of parameter entry 23
00000001
00000008
User ID to which ack sent

Figure 86. Exit-dependent data on entry to INMRZ12R

Parameter entries 10 and 12 (RECEIVE PARM string and message text for message ID INMR151I, respectively) are the same for all RECEIVE exits. For a description of those parameter entries, see "Parameter descriptions for INMRZ01R" on page 428.

Following are descriptions of the other parameter entries:

Action Flags (Parameter Entry 11)

The data field contains a word of action flags, which the exit can set to control RECEIVE processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
`1... ....`	RECEIVE is to issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
`.1.. ....`	If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that RECEIVE issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
`..11 ....`	Reserved
`.... 1...`	RECEIVE is to keep (not delete) the input file being received, even though no error has occurred.
`.... .1..`	RECEIVE is to delete the input file being received, even though errors are indicated.
`.... ..11`	Reserved

Status Flags (Parameter Entry 13)

The data field contains a word of status flags in which RECEIVE passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2 and 3 are reserved.

Bit	Meaning
`1... ....`	The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command.
`.1.. ....`	RECEIVE does not recognize the input data. Either an error occurred in the transmission, or the data was sent by a program other than TRANSMIT or PROFS.
`..11 ....`	Reserved
`.... 1...`	RECEIVE is to end processing. Either the user replied ‘END’ to a prompt, or an error occurred while prompting.
`.... .1..`	The user entered ‘DELETE’ in response to a prompt.
`.... ..11`	Reserved

RECEIVE Return Code (Parameter Entry 19)
The data field contains the RECEIVE return code.

Ddname of RECEIVE Output File (Parameter Entry 20)
If the RECEIVE pre-processing exit (INMRZ11R or INMRZ11) or the RECEIVE post-prompt exit (INMRZ15R) requested that the output be written to a temporary data set, this data field contains the ddname of that output file. If that request was not made, the field is blank.

Number of Records Read from the Input File (Parameter Entry 21)
The data field contains the number of records read from the input file during this receive operation.

Node to Which an Acknowledgment is to be Sent (Parameter Entry 22)
The data field contains either the node name or the node number to which the acknowledgment is sent. The exit can change that information.

User ID to Which an Acknowledgment is to be Sent (Parameter Entry 23)
The data field contains the user ID to which the acknowledgment is to be sent. The exit can change that information.

Parameter descriptions for INMRZ12

When INMRZ12 receives control, register 1 points to the following parameter list. Note that many of the parameters are the same as those that INMRZ01 and INMRZ11 receive. Explanations of those parameters are not repeated here. For more information about them, see “Parameter descriptions for INMRZ01” on page 431 or “Parameter descriptions for INMRZ11” on page 452.

Offset	Length	Value
+0	4	Address of an exit-to-exit communication word.
+4	4	Address of the character string that the user entered on the PARM keyword of the RECEIVE command.
+8	4	Address of a byte of action flags, which the exit can set to control RECEIVE processing after it returns. **Bit** **Action** `1... ....` RECEIVE is to issue message INMR151I, using the text contained in the parameter pointed to at offset +16. `.1.. ....` If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either sends an appropriate message to the user, or requests that RECEIVE issue message INMR151I, using the text contained in the parameter pointed to at offset +16. `..11 ....` Reserved `.... 1...` RECEIVE is to keep (not delete) the input file being received, even though no error has occurred. `.... .1..` RECEIVE is to delete the input file being received, even though errors are indicated. `.... ..11` Reserved

Offset	Length	Value
+12	4	Address of the TSO/E command processor parameter list (the CPPL).
+16	4	Address of the message text that RECEIVE is to issue with message ID INMR151I.
+20	4	Address of a status flag byte. RECEIVE uses this byte to pass indicators to the exit. **1... ....** The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command. **.1.. ....** RECEIVE does not recognize the input data. Either an error occurred in the transmission, or the data was sent by a program other than TRANSMIT or PROFS. **..11 ....** Reserved **.... 1...** RECEIVE is to end processing. Either the user replied 'END' to a prompt, or an error occurred while prompting. **.... .1..** The user entered 'DELETE' in response to a prompt. **.... ..11** Reserved
+24	4	If the user entered a character string on the PARM keyword in response to the RECEIVE prompt, this parameter contains the address of that string. The address points to an area that consists of a halfword length field followed by the character string. The length value is the length of the character string only. If no string is specified, the length value is zero.
+28	4	Address of a text unit pointer list. The list points to *copies* of text units that describe the incoming data.
+32	4	Address of another text unit pointer list. The list points to *copies* of text units that identify the originator of the transmission.
+36	4	Address of a string taken from the :PARM tag in the addressee's nickname entry in the sender's NAMES.TEXT data set.
+40	4	The TRANSMIT initialization exit (INMXZ01R or INMXZ01) can pass a user string to the RECEIVE data set exits, including this exit. If it does, this data field contains the address of that string.
+44	4	Address of a fullword that contains the RECEIVE command return code.
+48	4	If the RECEIVE pre-processing exit (INMRZ11R or INMRZ11) or the RECEIVE post-prompt exit (INMRZ15R) requested that the output be written to a temporary data set, this field contains the ddname of that output file. If that request was not made, the field is blank.
+52	4	Address of a fullword that contains the number of records read from the input file during this invocation of RECEIVE.
+56	4	Address of an 8-byte field that contains the node name or node number to which an acknowledgment is to be sent. The exit can modify this field.

Offset	Length	Value
+60	4	Address of an 8-byte field that contains the user ID to which an acknowledgment is to be sent. The exit can modify this field.

RECEIVE data set decryption exit — INMRZ13 or INMRZ13R

The RECEIVE data set decryption exit (INMRZ13 or INMRZ13R) receives control immediately after RECEIVE prompts the user for REPRO command options for deciphering an incoming file, and immediately before RECEIVE calls the Access Method Services REPRO function to decipher the file. The exit can:

- Examine and, if appropriate, change the options the user specifies. If RECEIVE is to do the deciphering, it builds the REPRO command after the exit returns, using any changes the exit makes.
- Invoke decryption processing itself. The exit is passed all the controls and ddnames that RECEIVE uses for decryption processing.
- Decide that decryption is not necessary, and prevent RECEIVE from doing it.

Parameter descriptions for INMRZ13R

When INMRZ13R receives control, register 1 points to the standard exit parameter list. The parameter list is described in "TSO/E standard exit parameter list" on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMRZ13R receives beginning at offset +36 (decimal).

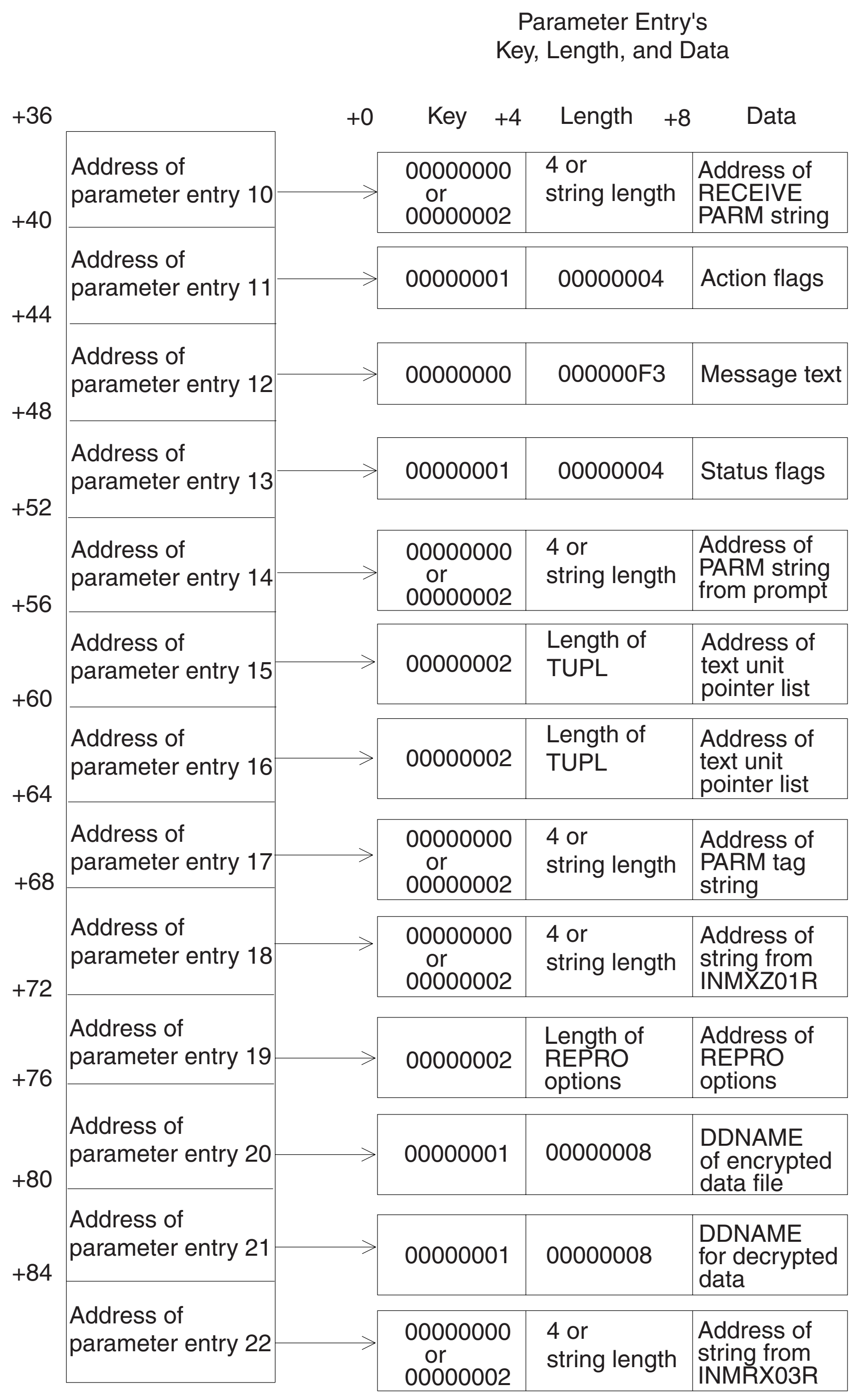

Parameter Entry's
Key, Length, and Data
+36
+40
+44
+48
+52
+56
+60
+64
+68
+72
+76
+80
+84
+0
Key
+4
Length
+8
Data
Address of parameter entry 10
00000000 or 00000002
4 or string length
Address of RECEIVE PARM string
Address of parameter entry 11
00000001
00000004
Action flags
Address of parameter entry 12
00000000
000000F3
Message text
Address of parameter entry 13
00000001
00000004
Status flags
Address of parameter entry 14
00000000 or 00000002
4 or string length
Address of PARM string from prompt
Address of parameter entry 15
00000002
Length of TUPL
Address of text unit pointer list
Address of parameter entry 16
00000002
Length of TUPL
Address of text unit pointer list
Address of parameter entry 17
00000000 or 00000002
4 or string length
Address of PARM tag string
Address of parameter entry 18
00000000 or 00000002
4 or string length
Address of string from INMXZ01R
Address of parameter entry 19
00000002
Length of REPRO options
Address of REPRO options
Address of parameter entry 20
00000001
00000008
DDNAME of encrypted data file
Address of parameter entry 21
00000001
00000008
DDNAME for decrypted data
Address of parameter entry 22
00000000 or 00000002
4 or string length
Address of string from INMRX03R

Figure 87. Exit-dependent data on entry to INMRZ13R

Many of the parameters are the same as those that INMRZ01R and INMRZ11R receive. Explanations of those parameters are not repeated here. For more information about them, see “Parameter descriptions for INMRZ01R” on page 428 or “Parameter descriptions for INMRZ11R” on page 448.

Following are descriptions of only those parameters that are different.

`Action Flags (Parameter Entry 11)`

The data field contains a word of action flags, which the exit can set to control RECEIVE processing after the exit returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
`1... ....`	RECEIVE is to issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
`.1.. ....`	If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that RECEIVE issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
`..11 ....`	Reserved
`.... 1...`	RECEIVE is not to issue the REPRO command. Either the exit deciphered the data, or no deciphering is required. If the exit deciphered the data, RECEIVE assumes that the exit copied the data from the file identified by the enciphered data ddname to the file identified by the plain text data ddname. Those ddnames are contained in the parameter entries pointed to at offsets +76 and +80, respectively.
`.... .1..`	Reserved
`.... ..1.`	All processing on the current file is complete. If the return code from the exit is zero, no more processing is done on the file. RECEIVE can delete the file from the JES spool and process the next file, if one exists. If the return code is non-zero, RECEIVE ignores this bit.
`.... ...1`	Reserved

`Status Flags (Parameter Entry 13)`

The data field contains a word of status flags in which RECEIVE passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2 and 3 are reserved.

Bit	Meaning
`1... ....`	The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command.
`.111 1111`	Reserved

`Address of REPRO Options (Parameter Entry 19)`

The data field points to an area that contains the options the user specified on the DECIPHER keyword of the REPRO command. RECEIVE uses that information to build a REPRO control statement that looks like:

```
REPRO INFILE(infile) OUTFILE(outfile) DECIPHER(options specified by
user and INMRZ13R)
```

The exit can modify the DECIPHER options. The modified information is used in decryption processing.

If the exit changes the length of the options string, it must also change the value in the length field accordingly. The maximum length allowed is 253 bytes. If the length of the data inserted is greater than 253 bytes, RECEIVE uses only the first 253 bytes, and sets the length to 253.

Ddname of Encrypted Data File (Parameter Entry 20)
The data field contains the ddname of the file that contains the encrypted data. The data set that corresponds to the ddname is a temporary data set that RECEIVE allocated. Do not modify this field.

Ddname of File for Decrypted Data (Parameter Entry 21)
The data field contains the ddname of the file to which the deciphered data is to be written. The file might or might not be the final output of RECEIVE processing. (For example, if partitioned data sets are being transmitted, the file contains them in their unloaded form.) Do not modify this field.

Address of a String from INMRX03R (Parameter Entry 22)
The TRANSMIT data set encryption exit (INMXZ03R or INMXZ03) can pass a user string to this exit. If it is does, this data field contains the address of that string.

If a string was specified:

KEY:	X'00000002'
LEN:	Length of the string

If no string was specified:

KEY:	X'00000000'
LEN:	X'00000004'

Parameter descriptions for INMRZ13

When INMRZ13 receives control, register 1 points to the following parameter list. Note that many of the parameters are the same as those that INMRZ01 and INMRZ11 receive. Explanations of those parameters are not repeated here. For more information about them, see “Parameter descriptions for INMRZ01” on page 431 or “Parameter descriptions for INMRZ11” on page 452.

Offset	Length	Value
+0	4	Address of an exit-to-exit communication word.
+4	4	Address of the character string that the user entered on the PARM keyword of the RECEIVE command.

Offset	Length	Value
+8	4	Address of a byte of action flags, which the exit can set to control RECEIVE processing after the exit returns. **Bit** **Action** **1... ....** RECEIVE is to issue message INMR151I, using the text contained in the parameter pointed to at offset +16. **.1.. ....** If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either sends an appropriate message to the user, or requests that RECEIVE issue message INMR151I, using the text contained in the parameter pointed to at offset +16. **..11 ....** Reserved **.... 1...** RECEIVE is not to issue the REPRO command. Either the exit deciphered the data, or no deciphering is required. If the exit deciphered the data, RECEIVE assumes that the exit copied the data from the file identified by the enciphered data ddname to the file identified by the plain text data ddname. Those ddnames are contained in the parameter entries pointed to at offsets +48 and +52, respectively. **.... .1..** Reserved **.... ..1.** All processing on the current file is complete. If the return code from the exit is zero, no more processing is done on the file. RECEIVE can delete the file from the JES spool and process the next file, if one exists. If the return code is non-zero, RECEIVE ignores this bit. **.... ...1** Reserved
+12	4	Address of the TSO/E command processor parameter list (the CPPL).
+16	4	Address of the message text that RECEIVE is to issue with message ID INMR151I.
+20	4	Address of a status flag byte. RECEIVE uses this byte to pass indicators to the exit. **1... ....** The input source for RECEIVE is not JES. The bit is on when the user specified either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command. **.111 1111** Reserved

Offset	Length	Value
+24	4	If the user entered a character string on the PARM keyword in response to the RECEIVE prompt, this parameter contains the address of that string. The address points to an area that consists of a halfword length field followed by the character string. The length value is the length of the character string only. If no string is specified, the length value is zero.
+28	4	Address of a text unit pointer list. The list points to *copies* of text units that describe the incoming data.
+32	4	Address of another text unit pointer list. The list points to *copies* of text units that identify the originator of the transmission.
+36	4	Address of a string taken from the :PARM tag in the addressee's nickname entry in the sender's NAMES.TEXT data set.
+40	4	The TRANSMIT initialization exit (INMXZ01R or INMXZ01) can pass a user string to the RECEIVE data set exits, including INMRZ12. If it does, this data field contains the address of that string.

+44	4	Address of an area that contains the options the user specified on the DECIPHER keyword of the REPRO command. RECEIVE uses that information to build a REPRO control statement that looks like:
		`REPRO INFILE(infile) OUTFILE(outfile) DECIPHER`
		`(options specified by user and INMRZ13R)`
		
		The exit can modify the DECIPHER options. The modified information is used in decryption processing.
		
		If the exit changes the length of the string, it must put the new length value in the length field. The maximum length allowed is 253 bytes. If the length of the data inserted is greater than 253 bytes, RECEIVE uses only the first 253 bytes, and sets the length to 253.

+48	4	Address of an 8-byte field that contains the ddname of the file that contains the encrypted data. The data set that corresponds to the ddname is a temporary data set that RECEIVE allocated. Do not modify this field.
+52	4	Address of an 8-byte field that contains the ddname of the file to which the deciphered data is to be written. The file might or might not be the final output of RECEIVE processing. (For example, if partitioned data sets are being transmitted, the file contains them in their unloaded form.) Do not modify this field.
+56	4	The TRANSMIT data set encryption exit (INMXZ03R or INMXZ03) can pass a user string to this exit. If it is does, this data field contains the address of that string. The address points to an area that consists of a halfword length field followed by the character string. The length value is the length of the character string only. If no string is specified, the length value is zero.

RECEIVE post-prompt exit — INMRZ15R

The RECEIVE post-prompt exit INMRZ15R receives control after the RECEIVE command processor prompts the user for actions to be taken with the transmitted data set, and just before the output data set is to be allocated. You can use the exit to:

- Determine what action to take with the arriving transmission after the user has responded to it. Exit INMRZ15R can be used to override and/or complement the user interaction required with the transmitted data set. The exit receives information about the incoming data set and the user's response to the RECEIVE prompt. It can specify the same parameters, or include additional parameters, for which RECEIVE already prompted the user.
- Determine which users can use a particular network path. The exit can check the source of a message before delivering it to the user.
- Receive data set types not supported by TRANSMIT and RECEIVE in conjunction with the RECEIVE data set post-processing exit (INMRZ12R or INMRZ12). This exit instructs RECEIVE to ignore the target dsname that the sender specified, and to write the received data into a temporary data set that INMXZ01R or another exit allocated.
- Suppress reception of data sets by setting the action flag that tells RECEIVE all processing on the current file is complete.

Note: INMRZ15R does not receive control if exit INMRZ11R requests that the user is not to be prompted for restore parameters.

Parameter descriptions for INMRZ15R

When INMRZ15R receives control, register 1 points to the standard exit parameter list. The parameter list is described in “TSO/E standard exit parameter list” on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMRZ15R receives beginning at offset +36 (decimal) in the parameter list.

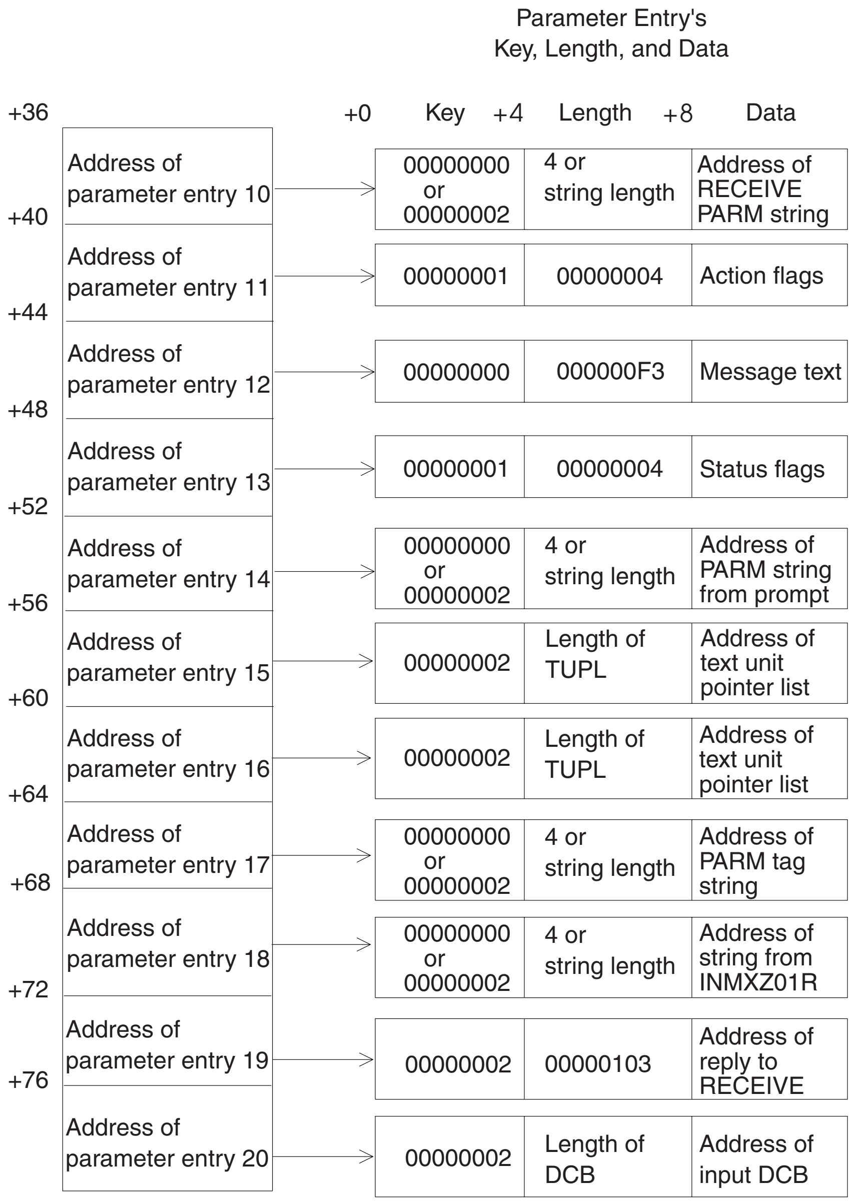

Parameter Entry's
Key, Length, and Data
+36
+0
Key
+4
Length
+8
Data
Address of parameter entry 10
00000000 or 00000002
4 or string length
Address of RECEIVE PARM string
+40
Address of parameter entry 11
00000001
00000004
Action flags
+44
Address of parameter entry 12
00000000
000000F3
Message text
+48
Address of parameter entry 13
00000001
00000004
Status flags
+52
Address of parameter entry 14
00000000 or 00000002
4 or string length
Address of PARM string from prompt
+56
Address of parameter entry 15
00000002
Length of TUPL
Address of text unit pointer list
+60
Address of parameter entry 16
00000002
Length of TUPL
Address of text unit pointer list
+64
Address of parameter entry 17
00000000 or 00000002
4 or string length
Address of PARM tag string
+68
Address of parameter entry 18
00000000 or 00000002
4 or string length
Address of string from INMXZ01R
+72
Address of parameter entry 19
00000002
00000103
Address of reply to RECEIVE
+76
Address of parameter entry 20
00000002
Length of DCB
Address of input DCB

Figure 88. Exit-dependent data on entry to INMRZ15R

Parameter entries 10 and 12 (RECEIVE PARM string and message text for message ID INMR151I, respectively) are the same for all RECEIVE exits. For a description of those parameter entries, see “Parameter descriptions for INMRZ01R” on page 428.

Following are descriptions of the other parameter entries:

Action Flags (Parameter Entry 11)

The data field contains a word of action flags, which the exit can set to control RECEIVE processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Action
`1... ....`	RECEIVE is to issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.

.1.. ....
: If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that RECEIVE issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.

..11 ....
: Reserved

.... 1...
: RECEIVE is to use the reply the exit specifies in the parameter entry pointed to at offset +72.

.... .1..
: RECEIVE is to restore the incoming data to a temporary data set. The RECEIVE data set post-processing exit (INMRZ12R or INMRZ12) should complete processing the data.

.... ..1.
: All processing on the current file is complete. If the return code from the exit is zero, no more processing is done on the file. RECEIVE can delete the file from the JES spool and process the next file, if one exists.

  If the return code is non-zero, RECEIVE ignores this bit.

....1
: RECEIVE is to issue message INMR045I when the output data set to be used for restoring the arriving transmission already exists.

Status Flags (Parameter Entry 13)

The data field contains a word of status flags in which RECEIVE passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Meaning
1...	The input source for RECEIVE is not JES. The bit is on when the user specifies either INDATASET, INDSNAME, INDDNAME, or INFILE on the RECEIVE command.
.1..	RECEIVE does not recognize the input data. Either an error occurred in the transmission or the data was sent by a program other than TRANSMIT or PROFS. If the exit is to process the data, it can get the address of the input DCB from the parameter entry pointed to at offset +76. The exit can also get the ddname of the file through the text unit pointer list in the parameter entry pointed to at offset +56. Before processing the data, the exit must close then reopen the data set to ensure that it starts reading at the beginning. (When the exit receives control, RECEIVE has already read past any control records, and has read at least the first record of the data set.)
..11	Reserved
.... 1...	RECEIVE is to restore the data to its original format and write it to the

appropriate log, which is the action that RESTORE(LOG) requests as well as the default when receiving messages.

.... .1..
The user has supplied a response to the RECEIVE prompt. The user's response is found in the parameter entry pointed to at offset +72.

.... ..11
Reserved

Address of PARM String from User Prompt (Parameter Entry 14)
If the user entered a character string on the PARM keyword in response to the RECEIVE prompt, this data field contains the address of that string.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

Address of Text Unit Pointer List (Parameter Entry 15)
The data field contains the address of a text unit pointer list. The list points to *copies* of text units that are sent with the transmission. They describe the incoming data, and can include:

```
INMBLKSZ  INMCREAT  INMDDNAM  INMDIR   INMDSNAM  INMDSORG
INMEXPDT  INMFM     INMLCHG   INMLRECL INMLREF   INMMEMBR
INMRECFM  INMSERP   INMSIZE   INMTERM  INMUTILN
```

For more information about these text units or text unit pointer lists, see “Text units and text unit pointer lists” on page 474. Note that altering the text units has no effect on subsequent processing. The exit receives copies of the actual text units for information only.

Address of Text Unit Pointer List (Parameter Entry 16)
The data field contains the address of another text unit pointer list. The list points to *copies* of text units that are sent with the transmission. They identify the originator of the transmission, and can include:

```
INMFACK   INMFNODE  INMFTIME  INMFUID
INMFVERS  INMLRECL  INMNUMF   INMUSERP
```

For more information about these text units or text unit pointer lists, see “Text units and text unit pointer lists” on page 474. Note that altering the text units has no effect on subsequent processing. The exit receives copies of the actual text units for information only.

Address of PARM Tag String (Parameter Entry 17)
The data field can contain the address of a string taken from the :PARM tag in the addressee's nickname entry in the sender's NAMES.TEXT data set. The string is present if the TRANSMIT command specified a nickname directly or through a distribution list, and the entry for that nickname in the sender's NAMES.TEXT data set contains the :PARM tag.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

Address of a String from INMXZ01R (Parameter Entry 18)

The TRANSMIT initialization exit (INMXZ01R or INMXZ01) can pass a user string to the RECEIVE data set exits, including INMRZ15R. If it does, this data field contains the address of that string.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

Address of Reply to RECEIVE (Parameter Entry 19)

The data field contains the address of the area in which the exit can find the user's response to the RECEIVE prompt. The exit can also use this area to build an overriding or complementary reply to the RECEIVE prompt. If the exit builds a reply, it must also set the key and length values to:

KEY: X'00000001'

LEN: Length of the reply plus a 4-byte header

The first two bytes (bytes 0 and 1) in the 4-byte header contain the length of the reply buffer. The second two bytes (bytes 2 and 3) contain the length of the user's reply to the RECEIVE prompt.

Note: Bytes 2 and 3 are 0 if the user does not supply a response.

Changing the value of bytes 2 and 3 in the header has no effect on processing. The exit can modify bytes 0 and 1, in effect changing the length of the reply buffer.

Address of the Input DCB (Parameter Entry 20)

The data field contains the address of the input DCB that identifies the data set being received. Before reading the data, the exit must close then reopen the data set to ensure that it starts reading at the beginning. (When the exit receives control, RECEIVE has already read past any control records, and has read at least the first record of the data set.)

RECEIVE log data set pre-allocation exit — INMRZ21R

Each time a log data set is to be allocated, the RECEIVE pre-allocation exit, INMRZ21R, receives control. This exit may be invoked multiple times for each RECEIVE operation. This exit can allocate the log data set, modify the data set name, or take no action. If the exit chooses to allocate the data set, the exit must pass the file name back to the RECEIVE command. If the exit modifies the data set name, the RECEIVE command will then allocate the data set for the exit if the exit has not already done so.

Under normal operations, the RECEIVE command frees the log data sets automatically. Therefore, even if the exit allocated the log data set, the data set is freed when the command processor terminates normally.

During RECEIVE processing, the allocation of the log data set occurs after the RECEIVE command has begun reading the input file.

If the exit is to allocate the log data set, consider the following:

- RECEIVE allocates the data set as NEW if it does not exist. This frees users from having to allocate the data set themselves.
- RECEIVE allocates the data set with a disposition of NEW or MOD. The exit should also allocate the data set with a disposition of NEW or MOD.
- If the exit allocates the data set as NEW, the data set must have the characteristics of a log data set. The characteristics of a log data set are:
  - Record Format (RECFM) of VB
  - Logical Record Length (LRECL) of 255
  - Blocksize (BLKSIZE) of 3120

Errors during log processing do not normally terminate command processing. Instead, the command sets a return code of 4, indicating that an error occurred but the transmission was successful. If the exit gives a non-zero return code, RECEIVE does not terminate processing, but sets a return code of 4, indicating that the logging function encountered an error.

Parameter descriptions for INMRZ21R

When INMRZ21R receives control, register 1 points to the standard exit parameter list. The parameter list is described in "TSO/E standard exit parameter list" on page 32. Note that the TRANSMIT and RECEIVE exits do not use the new command buffer field in the parameter entry pointed to at offset +4. Following is a description of the *exit-dependent* data that INMRZ21R receives beginning at offset +36 (decimal) in the parameter list.

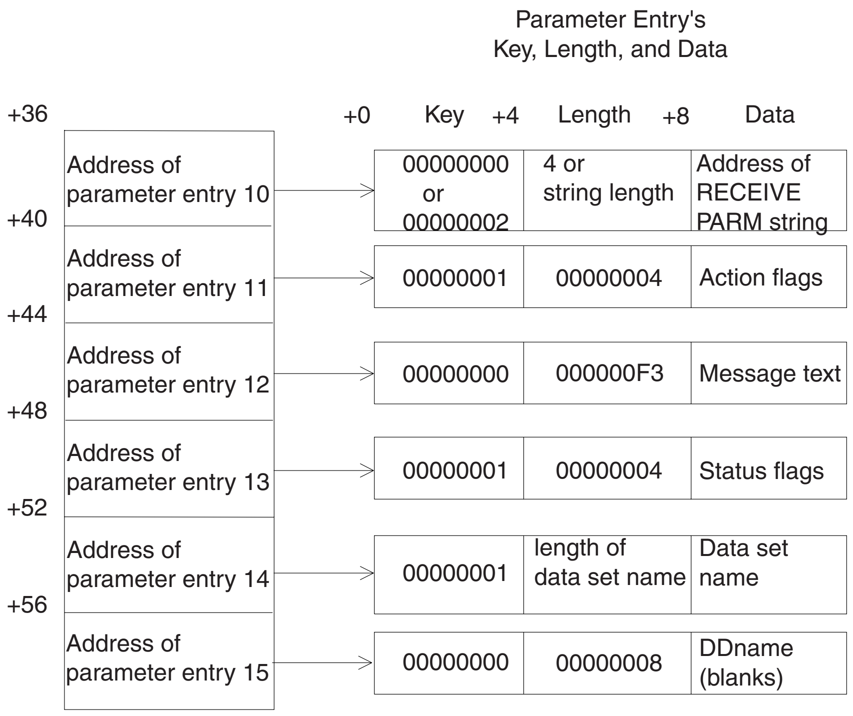


Figure 89. Exit-dependent data on entry to INMRZ21R

Following are descriptions of the information in the data fields of each parameter entry:

`Address of the RECEIVE PARM String (Parameter Entry 10)`

If the user entered a character string on the PARM keyword of the RECEIVE command, this data field contains the address of that string.

If a string was specified:

KEY: X'00000002'

LEN: Length of the string

If no string was specified:

KEY: X'00000000'

LEN: X'00000004'

For more information about how you can use the RECEIVE PARM keyword, see “Installation-defined parameters” on page 400.

`Action Flags (Parameter Entry 11)`

The data field contains a word of action flags, which the exit can set to control RECEIVE processing after it returns. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	**Action**
`1... ....`	RECEIVE is to issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
`.1.. ....`	If the return code from the exit is non-zero (end processing), RECEIVE is not to issue the normal error message. The exit either: already sent an appropriate message to the user; set return code 12, which causes message IKJ79154I to be issued; or requested that RECEIVE issue message INMR151I, using the text contained in the parameter entry pointed to at offset +44.
`..11 ....`	Reserved
`.... 1...`	The exit has allocated the log data set. The ddname representing the data set is contained in the parameter entry pointed to by offset +56.
`.... .1..`	The exit has modified the log data set name that was passed in the parameter entry pointed to by offset +52.
`.... ..11`	Reserved

`Message Text (Parameter Entry 12)`

Exits can put in this data field the message text that the command is to issue with message ID INMR151I. The field initially contains blanks. If the exit inserts text in the data field, it must also set the key and length values to:

KEY: X'00000001'

LEN: Length of the message text. The maximum length is 243.

Status Flags (Parameter Entry 13)
The data field contains a byte of status flags in which RECEIVE passes indicators to the exit. The flags in byte 0 are defined as follows. Bytes 1, 2, and 3 are reserved.

Bit	Meaning
`1... ....`	RECEIVE is not interfacing with JES. A file keyword (INFILE, INDDNAME) was specified.
`.111 1111`	Reserved

Data Set Name (Parameter Entry 14)
The data field contains the name of the log data set that will be allocated by the command. If the exit chooses to modify the data set name, the length field should be updated to reflect the correct length of the data set name. In addition, if the exit modifies the data set name, the X'04' action flag should be set to indicate that modifications were made. If the exit updates this parameter entry, update the length and key fields accordingly. If the length fields are incorrect, the parameter is ignored and processing continues as it would have if the exit did not modify this parameter.

DDNAME (Parameter Entry 15)
If the exit allocates the log data set to a file, the file name should be placed in this parameter entry. The length should be updated to reflect the length of the file name. If the exit allocates the data set, the X'08' action flag should be set to indicate that the data set has been allocated. If the exit allocates the data set, the command will free the data set through normal processing.

If the exit updates this parameter entry, update the length and key fields accordingly. If the length fields are incorrect, the parameter is ignored and processing continues as it would have if the exit did not modify this parameter.

Text units and text unit pointer lists

Many of the TRANSMIT and RECEIVE exits are passed the addresses of text unit pointer lists. Each entry in the text unit pointer list points to a copy of a text unit that contains information used to control the transmission -- information such as the record format, or the origin node name or user ID. Because the exits receive read-only copies of the text units, they cannot affect subsequent processing by changing the information in them.

Text units are sent in control records as part of the transmission. It is possible that a text unit could be split among two or more control records. For more information about the format of control records, see “Format of transmitted data” on page 482.

The following figure shows the format of text unit pointer lists and text units. Note that the last entry in the text unit pointer list has the high-order bit is set to one.

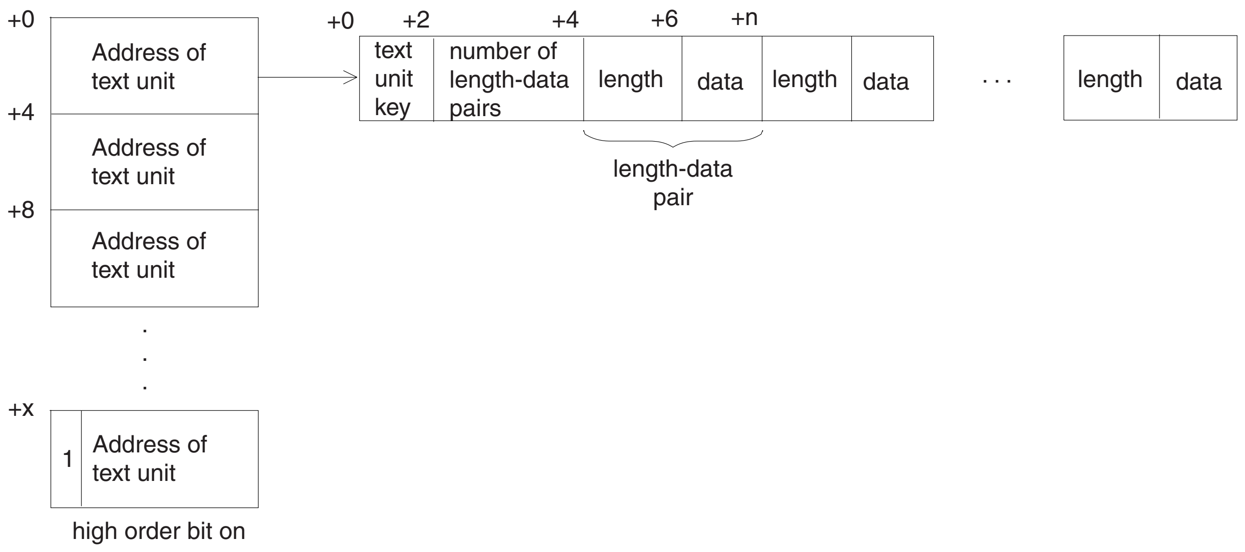


Figure 90. Format of text unit pointer lists and text units

Offset	Length	Description
+0	2	**Text unit key**. The key identifies the type of information contained in the text unit. Possible key values are given in "Types of text units."
+2	2	**Number**. The number field contains the number of length-data pairs that follow. Most of the text units have only one length and one data field.
+4	2	**Length**. The first of perhaps many length fields. The length value includes only the length of the data field immediately following it, and not its own two-byte length.
+6	n	**Data**. The first of perhaps many data fields. The data field contains the parameter information being passed, for example, the target node name. The descriptions of the individual text units, which follow, describe the content of each.
+(8+n)		Second length-data entry if the number field indicates more than one entry is present.

Types of text units

Following is a list of the text units that are currently defined. (Text unit key names are defined by macro INMTEXTU). Table 71 on page 476 describes each text unit in more detail and gives an example of each.

Text unit key (hex)	Mnemonic	Data
0030	INMBLKSZ	Block size
1022	INMCREAT	Creation date
0001	INMDDNAM	DDNAME for the file
000C	INMDIR	Number of directory blocks
0002	INMDSNAM	Name of the file
003C	INMDSORG	File organization
8028	INMEATTR	Extended attribute status
1027	INMERRCD	RECEIVE command error code
0022	INMEXPDT	Expiration date

Text unit key (hex)	Mnemonic	Data
1026	INMFACK	Originator requested notification
102D	INMFFM	Filemode number
1011	INMFNODE	Origin node name or node number
1024	INMFTIME	Origin timestamp
1012	INMFUID	Origin user ID
1023	INMFVERS	Origin version number of the data format
1021	INMLCHG	Date last changed
0042	INMLRECL	Logical record length
1020	INMLREF	Date last referenced
8018	INMLSIZE	Data set size in megabytes.
0003	INMMEMBR	Member name list
102F	INMNUMF	Number of files transmitted
102A	INMRECCT	Transmitted record count
0049	INMRECFM	Record format
000B	INMSECND	Secondary space quantity
102C	INMSIZE	File size in bytes
0028	INMTERM	Data transmitted as a message
1001	INMTNODE	Target node name or node number
1025	INMTTIME	Destination timestamp
1002	INMTUID	Target user ID
8012	INMTYPE	Data set type
1029	INMUSERP	User parameter string
1028	INMUTILN	Name of utility program

Dates and times

All dates and times specified in text units are expressed in GMT using the standard format: EBCDIC characters for the year(4), month(2), day(2), hour(2), minute(2), second(2), and fraction of seconds(n). Only the information that is known is specified. For example, if only the year is known, the value is `yyyy`. If microseconds are known, the value is `yyyymmddhhmmssffffff`.

Numeric values

All numeric values in text units are specified with a length of 1 to 8 bytes. If the field is longer than 8 bytes, only the low-order 4 bytes are used.

Table 71. Text unit descriptions and format

Mnemonic	Key (hex)	Number	Length	Data

INMBLKSZ	0030	1	1 to 8 bytes	**Blocksize of the file**
				
				Example: Specification of blocksize 32768.
				`KEY    NUM    LEN    DATA`
				`0030   0001   0004   00008000`

INMCREAT	1022	1	4 or more bytes	**Creation date of the file, in standard format**.
				
				Example: Specification of September 25, 1987.
				`KEY    NUM    LEN    DATA`
				`1022   0001   0008   F1F9F8F7F0F9F2F5`

INMDDNAM	0001	1	Length of name	**DDNAME associated with the file**.
				
				Example: Specification of DDNAME DD1.
				`KEY    NUM    LEN    DATA`
				`0001   0001   0003   C4C4F1`

Table 71. Text unit descriptions and format (continued)

Mnemonic	Key (hex)	Number	Length	Data

INMDIR	000C	1	1 to 8 bytes	**Number of directory blocks in the file**.
				
				Example: Specification of 15 directory blocks.
				`KEY  NUM  LEN  DATA`
				`000C 0001 0003 00000F`

INMDSNAM	0002	Number of fields in file name	Length of file name field	**Name of the file**.
				
				File names are divided into fields. In MVS, the name contains a maximum of 22 fields, each having at most 8 characters. The fields are separated by periods: AA.BB.CC.DD is an example. The total length, including the periods, must not exceed 44 characters.
				
				In CMS, file names have three fields: filename, filetype, and filemode, with a maximum length of 8, 8, and 2 characters, respectively. The fields are separated by blanks: REPORT SCRIPT A1 is an example.
				
				When transmitting a CMS file, the filemode character (but not the number) is specified in this text unit. The filemode number is specified in the INMFFM text unit. The filemode character is the first field of the file name.
				
				Note that TSO/E data set names are transmitted differently in control records than in the text units available to installation exits through text unit pointer lists (as shown in the examples that follow). In the text units pointed to from text unit pointer lists, the name is sent as a single length-data pair. In control records, each part of the name and its length is a separate length-data pair.
				
				Example: Specification of TSO/E data set A.B in text units available through pointer lists.
				`KEY  NUM  LEN  DATA`
				`0002 0001 0003 C14BC2`
				
				Example: Specification of TSO/E data set A.B in INMR02 control records.
				`KEY  NUM  LEN1  DATA1  LEN2  DATA2`
				`0002 0002 0001  C1     0001  C2`

INMDSORG	003C	1	2 bytes	**The file organization**, either:
				X'0008' for VSAM
				X'0200' for partitioned organization
				X'4000' for physical sequential
				
				Example: Specification of a physical sequential file.
				`KEY  NUM  LEN  DATA`
				`003C 0001 0002 4000`

Table 71. Text unit descriptions and format (continued)

Mnemonic	Key (hex)	Number	Length	Data

INMEATTR	8028	1	1 byte	**Extended attribute status**. It can be in any of the following formats:
				X'00' Unspecified
				X'01' NO
				X'02' OPT
				
				Example: Specification of EATTR as OPT.
				`KEY  NUM  LEN  DATA`
				`8028 0001 0001 02`

INMERRCD	1027	1	1 or more bytes	**A string indicating the result of the RECEIVE operation**.
				
				Example: Specification that the transmitted file was "RECEIVED".
				`KEY  NUM  LEN  DATA`
				`1027 0001 0008 D9C5C3C5C9E5C5C4`

INMEXPDT	0022	1	4 or more bytes	**Expiration date of the file, in standard format**.
				
				Example: Specification of January 1, 1988.
				`KEY  NUM  LEN  DATA`
				`0022 0001 0008 F1F9F8F8F0F1F0F1`

INMFACK	1026	1	1 to 64 bytes	**Blanks or a notification string that the issuer of TRANSMIT specified**.
				
				This text unit exists only if the sender (or installation exit) requested acknowledgment that the transmitted file was received. If the sender also specified a notification string to identify the transmission, that string is in the text unit data field. If no string was specified, the length and data fields are blank.
				
				Example: Specification of notification with string FRED.
				`KEY  NUM  LEN  DATA`
				`1026 0001 0004 C6D9C5C4`
				
				Example: Specification of notification without a string.
				`KEY  NUM  LEN  DATA`
				`1026 0001`

INMFFM	102D	1	1 byte	**Filemode number of a CMS file**.
				
				Example: Specification of filemode number 0.
				`KEY  NUM  LEN  DATA`
				`102D 0001 0001 F0`

INMFNODE	1011	1	Length of node name or number	**Name or number of the origin node**.
				
				Example: Specification of node VENICE.
				`KEY  NUM  LEN  DATA`
				`1011 0001 0006 E5C5D5C9C3C5`

INMFTIME	1024	1	4 or more bytes	**The time the transmission was created, in standard format.**
				
				Example: Specification of July 19, 1987 at 3:20 PM.
				`KEY  NUM  LEN  DATA`
				`1024 0001 000C F1F9F8F7F0F7F1F9F5F2F0`

Table 71. Text unit descriptions and format (continued)

Mnemonic	Key (hex)	Number	Length	Data

INMFUID	1012	1	Length of user ID	**User ID of the originator of the transmission**.
				
				If the transmission is a file, the originator is the issuer of TRANSMIT. If the transmission is an acknowledgment, the originator is the issuer of the acknowledgment.
				
				Example: Specification of user ID IMBUSER.
				`KEY    NUM    LEN    DATA`
				`1012   0001   0007   C9C2D4E4E2C5D9`

INMFVERS	1023	1	1 to 8 bytes	**The version number of the data format used for the transmission**.
				
				Example: Specification of version 1.
				`KEY    NUM    LEN    DATA`
				`1023   0001   0004   00000001`

INMLCHG	1021	1	4 or more bytes	**Date the file was last changed, in standard format**.
				
				Example: Specification of April 1, 1987 at 8:12 PM.
				`KEY    NUM    LEN   DATA`
				`1021   0001   000E  F1F9F8F7F0F4F0F1F2F0F1F2`

INMLRECL	0042	1	1 to 8 bytes	**The actual or maximum number of bytes in the logical record in the file.**
				
				Example: Specification of 80-byte records.
				`KEY    NUM    LEN    DATA`
				`0042   0001   0001   50`

INMLREF	1020	1	4 or more bytes	**Date the file was last referenced, in standard format**.
				
				Example: Specification of February 14, 1988.
				`KEY    NUM    LEN    DATA`
				`1020   0001   0008   F1F9F8F7F0F2F1F4`

INMLSIZE	8018	1	4 bytes	**Size of the file in megabytes (MB)**.
				Note: The value in this text unit for a partitioned data set specifies the size of the PDS, not the size of a member.
				
				Example: Specification of a two gigabyte file.
				`KEY     NUM   LEN    DATA`
				`8018    0001  0004   00000800`

INMMEMBR	0003	1 or more	Length of member name	**List of member names**.
				
				If more than one member is being transmitted, there are multiple length-data field pairs, one for each member.
				
				Example: Specification of members IEASYS00 and A.
				`KEY  NUM  FLGS  LEN1  DATA1`
				`0003 0002 xxxx  0008  C9C5C1E8E2F0F0`
				`LEN2  DATA2`
				`0001  C1`

Table 71. Text unit descriptions and format (continued)

Mnemonic	Key (hex)	Number	Length	Data

INMNUMF	102F	1	1 to 8 bytes	**Number of files that make up the transmission**.
				
				If any files are being transmitted, this text unit must be in the INMR01 control record. If the text unit is missing, the number of files is assumed to be zero, which is true only when the transmission is an acknowledgment.
				
				Example: Specification of two files in the transmission.
				`KEY    NUM    LEN    DATA`
				`102F   0001   0004   00000002`

INMRECCT	102A	1	1 to 8 bytes	**Number of records transmitted**.
				
				Example: Specification of 129 records.
				`KEY    NUM    LEN    DATA`
				`102A   0001   0001   81`

INMRECFM	0049	1	2 bytes	**The record format of the file**.
				
				The value is the result of "logically ORing" one or more of the following values together:
				X'0001' Shortened VBS format used for transmission records
				X'xx02' Varying length records without the 4-byte header
				X'0200' Data includes machine code printer control characters
				X'0400' Data contains ASA printer control characters
				X'0800' Standard fixed records or spanned variable records
				X'1000' Blocked records
				X'2000' Track overflow or variable ASCII records
				X'4000' Variable-length records
				X'8000' Fixed-length records
				X'C000' Undefined records
				
				Example: Specification of fixed block records.
				`KEY    NUM    LEN    DATA`
				`0049   0001   0002   9000`

INMSECND	000B	1	3 bytes	**Secondary space quantity**.
				
				Example: Specification of 10 blocks.
				`KEY    NUM    LEN    DATA`
				`000B   0001   0003   00000A`

INMSIZE	102C	1	4 bytes	**Size of the file in bytes**.
				
				Note that the value in text units for partitioned data sets specifies the size of the PDS, not the size of a member.
				
				Example: Specification of a 1,000,000 bytes file.
				`KEY    NUM    LEN    DATA`
				`102C   0001   0004   000F4240`

Table 71. Text unit descriptions and format (continued)

Mnemonic	Key (hex)	Number	Length	Data

INMTERM	0028	0	Omitted	**Omitted**. This text unit indicates that the data was transmitted as a message.
				
				Example: Specification of transmitted data.
				`KEY    NUM    LEN    DATA`
				`0028   0000`

INMTNODE	1001	1	Length of node name	**Node name to which the transmission is being sent**
				
				Example: Specification of node ROME.
				`KEY    NUM    LEN    DATA`
				`1001   0001   0004   D9D6D4C5`

INMTTIME	1025	1	4 or more bytes	**The time the transmission was received, in standard format**.
				
				Example: Specification of March 14, 1987 at 8:30 AM.
				`KEY    NUM   LEN   DATA`
				`1025   0001  000E  F1F9F8F7F0F3F1F4F0F8F3F0`

INMTUID	1002	1	Length of user ID	**User ID to which the transmission is being sent**.
				
				Example: Specification of user ID IBMUSER.
				`KEY    NUM    LEN    DATA`
				`1002   0001   0007   C9C2D4E4E2C5C9`

INMTYPE	8012	1	1 byte	**Data set type**.
				X'80' Data library
				X'40' Program library
				X'04' Extended format sequential data set
				X'01' Large format sequential data set
				
				Example: Specification of data library.
				`KEY    NUM    LEN    DATA`
				`8012   0001   0001   80`

INMUSERP	1029	1	1 to 251 bytes	**The character string specified on the PARM keyword of the TRANSMIT or RECEIVE command.**
				
				Example: Specification of user string ‘PARM1’.
				`KEY    NUM    LEN    DATA`
				`1029   0001   0005   D7C1D9D4F1`

Table 71. Text unit descriptions and format (continued)

Mnemonic	Key (hex)	Number	Length	Data

INMUTILN	1028	1	Length of name	**Name of the utility program that is used in restoring the transmitted data to its original format.**
				
				Currently defined names are:
				
				INMCOPY
				Invokes an internal utility to convert from the transmission format to a sequential file.
				
				IEBCOPY
				Invokes the IEBCOPY utility to reload a partitioned file.
				
				AMSCIPHR
				Invokes the Access Method Services REPRO command to decrypt a file.
				
				Example: Specification of utility program INMCOPY.
				`KEY    NUM    LEN    DATA`
				`1028   0001   0007   C9D5D4C3D6D7E8`

Format of transmitted data

Files sent using the TRANSMIT command contain: control records, data records, and a trailer record (a special type of control record). Several control records begin the file. They are followed by data records, and finally a trailer record. An acknowledgment file is composed of only control records.

Data is actually transmitted in card image format (80-byte records). However, the record descriptions that follow are those of ‘logical records’; card boundaries are ignored. Each control record begins in the byte immediately following the end of the previous record. The first data record begins in the byte immediately following the end of the last control record. The trailer control record begins in the byte immediately following the last data record.

Control records and data records have the same format. The records of the file to be transmitted are broken into segments whose format is shown in the next figure. A segment has a maximum length of 255 bytes, including the 2-byte header. If the length of a record in the file is greater than 253 bytes, the record is sent as multiple segments.

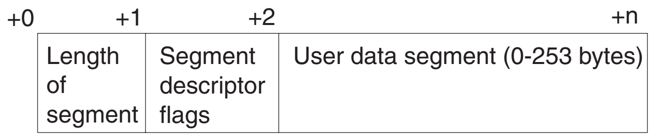


Figure 91. Format of control records and data records

Byte	Length	Contents or meaning
0	1	Length of segment, including 2-byte header (length equals 2 to 255)

Byte	Length	Contents or meaning
1	1	Segment descriptor flags: **X'80'** First segment of original record. **X'40'** Last segment of original record. **X'20'** This is (part of) a control record. **X'10'** This is record number of next record. **X'0F'** Reserved
2	n	User data segment (n = 0 to 253). Control records have a control record identifier (for example, INMR01) in bytes 2-7. Text units generally begin in byte 8.

Control record formats

The following sections describe the format of various control records.

INMR01 -- header record

The INMR01 record is always the first record of a transmission. The identifier of the record is 'INMR01' in bytes 2-7. The remainder of the record (beginning with byte 8) is composed of text units. The text unit keys always present are:

INMFNODE
Origin node name

INMFTIME
Origin timestamp

INMFUID
Origin user ID

INMLRECL
Length of physical control record segments

INMTNODE
Target node name

INMTUID
Target user ID

Text units that can be present are:

INMFACK
Receipt notification requested

INMFVERS
Origin version number

INMNUMF
Number of files in this transmission

INMUSERP
User parameter string

INMR02 -- file utility control record

Each INMR02 record controls a data restoration step. In a given transmission, one or more processes represented by a corresponding number of INMR02 records are required. Utility operations currently supported are: INMCOPY, which converts sequential files to and from the TRANSMIT/RECEIVE format; IEBCOPY, which converts partitioned files to and from sequential files (called *unloaded* files); and AMSCIPHR, which invokes the Access Method Services REPRO command to encrypt and decrypt files.

If more than one INMR02 record is present, they appear in the order in which TRANSMIT processes them, which is the inverse of the order used by RECEIVE. The first record in the data stream represents the last utility operation to be performed by RECEIVE. The INMCOPY utility is always the last or the only utility invoked during a TRANSMIT operation, and as a result appears last in the file.

The text units that are in the INMR02 record describe the output of the utility operation. The input with which it must work is described by the previous INMR02-directed operation or by the INMR03 data description record.

If the transmission contains more than one file, one or more INMR02 records are required for each file in the transmission. The groups of INMR02 records are in the same order as the files in the transmission. The file number field identifies which of the multiple files in the transmission the control record applies.

The identifier for this record is 'INMR02' in bytes 2-7. Bytes 8-11 contain the number of the file in this transmission to which the control record applies. Multiple files in a single transmission are numbered sequentially starting at one. The text units begin in byte 12. Text units always present are:

INMDSORG
File organization

INMLRECL
Logical record length

INMRECFM
Record format

INMSIZE
Approximate size of file in bytes

INMUTILN
Utility program name

Text units that can be present are:

INMBLKSZ
File block size

INMCREAT
Creation date

INMDIR
Number of directory blocks

INMDSNAM
File name

INMEXPDT
Expiration date

INMFM
Filemode number

INMLCHG
Last change date

INMLREF
Last reference date

INMMEMBR
Member name list

INMTERM
Mail file

INMUSERP
User parameter string

INMR03 -- data control record

The INMR03 record immediately precedes the transmitted data and identifies its format.

The identifier for this record is ‘INMR03’ in bytes 2-7; text units begin in byte 8. Text units always present are:

INMDSORG
File organization

INMLRECL
Logical record length

INMRECFM
Record format

INMSIZE
Size of file in bytes

INMR04 -- user control record

The INMR04 record may appear anywhere among the control records. It contains user data to be passed to all installation exits during the RECEIVE operation.

The identifier for this record is ‘INMR04’ in bytes 2-7; text units begin in byte 8. The text unit always present is:

INMUSERP
User parameter string

INMR06 -- trailer control record

The INMR06 record is always the last record in a transmission. This record is used to verify that the transmission is complete.

The identifier for this record is ‘INMR06’ in bytes 2-7. No text units are presently defined for this record.

INMR07 -- notification control record

The INMR07 record indicates notification of a previous transmission. When it appears, the transmission consists of only the INMR01, INMR07, and INMR06 records.

The identifier for this record is ‘INMR07’ in bytes 2-7; text units begin in byte 8. INMFTIME,and either INMDSNAM or INMTERM, but not both, are always present.

INMFTIME
Origin timestamp

INMDSNAM
File name

INMTERM
Data transmitted as a message

Text units that might be present are:

INMERRCD
: Error indication for the receive operation

INMFACK
: Notification ID

INMFFM
: Filemode number

INMUSERP
: User parameter string

Chapter 40. Customizing the TSOLIB command

This section describes ways to customize the TSOLIB command.

Writing exits for the TSOLIB command

TSO/E users issue the TSOLIB command to dynamically link to different versions of load module libraries while in their TSO/E session. TSO/E searches these libraries before those specified in the user's logon procedure. Therefore, the users' logon procedures can be kept simpler. Also, system and application programmers do not need to maintain different user IDs just to switch to different versions of load modules.

TSO/E provides initialization and termination exits that you can use to customize the TSOLIB command. These exits receive control during TSOLIB command processing, as follows:

- The initialization exit receives control before the TSOLIB command processor invokes the parse service routine to syntax check the input parameters.
- The termination exit receives control just before the TSOLIB command terminates processing.

TSO/E-supplied exits

TSO/E does not provide default exit routines for the TSOLIB command.

Entry specifications

The contents of the registers on entry for the initialization and termination exits for the TSOLIB command are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions for the initialization exit

The initialization exit receives the standard exit parameter list. However, no exit-dependent data is passed to the initialization exit. For information about the standard exit parameter list and the parameter entry keys, see “TSO/E standard exit parameter list” on page 32.

Parameter descriptions for the termination exit

The termination exit receives the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. If you provide an initialization exit, the termination exit is passed the same parameter entries for the new command buffer and exit-to-exit communication word that were passed to the initialization exit. Figure 92 shows the exit-dependent data that the termination exit receives beginning at offset +36 (decimal) in the parameter list. The parameter entry is described following the figure.

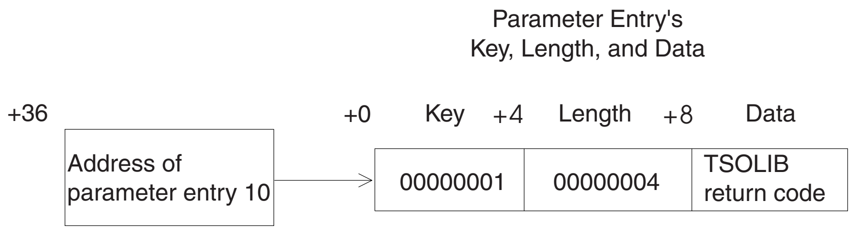


Figure 92. Exit-dependent data for the TSOLIB command termination exit

`TSOLIB Return Code (Parameter Entry 10)`
This parameter entry is the return code from the TSOLIB command processor. For information on the return codes from the TSOLIB command, see *z/OS TSO/E Command Reference*.

Return specifications

The contents of the registers on return from the initialization and termination exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the initialization and termination exits

Table 72 shows the return codes that the initialization and termination exits support.

Table 72. Return codes for the TSOLIB initialization and termination exits

Return code (decimal)	Description
0	Exit processing was successful. TSOLIB processing continues.
12, 16	An error occurred in the exit. The TSOLIB command processor terminates processing. If the exit uses return code 12, it can also pass back an exit reason code to the TSOLIB command processor. For more information, see the notes following this table and "Exit reason code" on page 38. If your exit sets a return code of either 12 or 16, you should consider displaying an informational message to the user. You can use the PUTLINE service routine to issue an informational message. See *z/OS TSO/E Programming Services* for more information.

Note:
1. If an exit returns an undefined return code, the TSOLIB command processor terminates without issuing an error message to the user.
2. If an initialization or termination exit sets a reason code that has a key value of X'03', this reason code is used as the return code from the TSOLIB command. However, if both exits indicate that the reason code is to be used as the return code from the TSOLIB command, the reason code from the termination exit overrides that from the initialization exit.
3. When requesting that the exit reason code be used as the return code from TSOLIB, you must insure that the reason code does not duplicate existing TSOLIB return codes.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return. The exits must be reentrant, refreshable, and reusable.

If the processing done in the initialization exit requires clean-up to be performed, you must write a termination exit. For example, if the initialization exit obtains storage to return a new command buffer to the TSOLIB command processor, you must provide a termination exit to free this storage.

The exits can use any of the TSO/E service routines. For a description of the service routines, see *z/OS TSO/E Programming Services*.

Environment

- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY)

Installing the exits

You must name the exits as follows:

Initialization
IDYTSINI

Termination
IDYTSTER

Link-edit each exit as a separate load module. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see “Installing the standard-format exits” on page 39.

Possible uses

You can use the initialization exit to change the default values of the TSOLIB command. For example, you can cause the COND operand to be the default.

You can use the termination exit to perform clean-up processing. For example, you can free storage that was obtained in the initialization exit.

Part 6. Customizing CLIST and REXX processing

This part describes how you can customize REXX and CLIST processing. You can customize CLIST processing using exits. You can customize REXX processing using exits and replaceable routines. You can also optimize the performance of REXX execs and CLISTs by storing them in the virtual lookaside facility (VLF).

- **CLIST Exits**

  The CLIST language provides a wide range of programming functions. TSO/E provides the following exits that you can use to customize the CLIST language:

  – CLIST built-in function exit:

    The CLIST language provides built-in functions that can be performed on variables, expressions, and character strings. In addition to these built-in functions, TSO/E provides an exit that you can use to add your own CLIST built-in functions. For more information, see "Writing an exit for installation-written built-in functions (IKJCT44B)" on page 495.

  – CLIST statement exit:

    The CLIST language provides statements that let you structure your programs, perform I/O, define and modify variables, and monitor the conditions under which CLISTs execute. In addition to the statements provided by the CLIST language, TSO/E provides an exit that you can use to add your own CLIST statements. For more information, see "Writing an exit for installation-written statements (IKJCT44S)" on page 499.

- **REXX Replaceable Routines and Exits**

  The REXX language provides a wide range of programming functions. REXX has many characteristics similar to CLIST. However, a significant difference between REXX and CLIST is that you can execute CLISTs only in a TSO/E address space. You can execute REXX execs in any MVS address space. TSO/E REXX supports the use of replaceable routines and exit routines to customize REXX processing in different environments.

  – Replaceable Routines

    Replaceable routines allow you to customize REXX processing to call the appropriate system-supplied routines for a language processor environment. TSO/E provides the following types of replaceable routines for REXX processing:
    - Exec load
    - Input/Output
    - Data stack
    - Storage management
    - User ID
    - Message identifier
    - Host command environment

  – REXX Exits

    You can also customize TSO/E REXX using exit routines, which receive control from exit points in TSO/E REXX routines, perform special processing, and return control to the REXX routine. TSO/E provides the following types of exits for REXX processing:

    - TSO/E-Supplied Exits:
      - Pre-environment initialization exit (IRXINITX)

IRXINITX performs processing before a new language processor environment is initialized.

- Post-environment initialization exit (IRXITTS or IRXITMV)

  IRXITTS performs processing after the new language processor environment is initialized for an environment integrated with TSO/E.

  IRXITMV performs processing after the new language processor is initialized for an environment *not* integrated with TSO/E.
- Environment termination exit (IRXTERMX)

  IRXTERMX performs processing before a language processor environment is terminated.

- User-Supplied Exits:
  - Exec processing exit

    Use this exit to prevent the execution of a REXX exec or perform special processing before a REXX exec is executed.
  - Exec initialization exit

    Use this exit to access and update REXX variables.
  - Exec termination exit

    Use this exit to access and update REXX variables.
  - Attention handling exit

    Use this exit to perform special attention processing. It can only be used for an environment integrated with TSO/E.

- **Storing CLISTs and REXX Execs in VLF**

  You can improve performance of CLISTs and REXX execs that the EXEC command finds at the system-level CLIST (SYSPROC) and application-level CLIST (defined using the TSO/E ALTLIB command), by using the virtual lookaside facility (VLF). To improve performance of these CLISTs and REXX execs, define the IKJEXEC class name in the facility VLF PARMLIB member, COFVLFxx. For example,

```
⋮
CLASS NAME (IKJEXEC)
  EDSN(data set name)    /* (your partitioned data set name) */
  EDSN(data set name)    /* (your partitioned data set name) */
⋮
```

  Each eligible data set name (EDSN) entry in that member identifies a data set. The order of the entries is not significant.

  To optimize VLF for CLISTs, you should:

  – Maintain frequently-used CLISTs in separate data set(s).
  – Where possible, place VLF-managed data sets at the front of non-VLF-managed data sets in the SYSPROC concatenation.
  – For non-VLF-managed data sets that occur ahead of VLF-managed data sets in the concatenation, minimize copies of CLISTs with the same name as CLISTs in VLF-managed data sets.
  – Avoid frequent updates to VLF-managed CLISTs.
  – Minimize dynamic changes to CLIST data concatenations.

  When setting up the PARMLIB member, your installation should define a data set name in only one VLF class. For more information about COFVLFxx and about using VLF, see *z/OS MVS Initialization and Tuning Reference*. For more information about the TSO/E ALTLIB command, see *z/OS TSO/E Command Reference*.

If a library is properly defined in a COFVLFxx member and only one system is involved, VLF change notification is automatic. If several systems share the data, VLF change notification might also be automatic. For information about when VLF change notification is automatic, see *z/OS MVS Programming: Authorized Assembler Services Guide*. For more information about the VLFNOTE command, see *z/OS TSO/E Command Reference*.

The system searches VLF for REXX execs invoked implicitly or explicitly by the EXEC command. However, VLF is not searched for REXX execs invoked as functions or subroutines.

For example, suppose you have two REXX execs, A and B, both managed by VLF. REXX exec A is invoked via the EXEC command. REXX exec B is invoked via an external function from REXX exec A. If both REXX execs are changed on one system, the changes to REXX exec A are not seen on the other systems unless VLFNOTE is issued.

Chapter 41. Customizing CLIST processing

This section discusses the two exits you can use to customize CLIST processing:
- CLIST built-in function exit (IKJCT44B)
- CLIST statement exit (IKJCT44S).

Writing an exit for installation-written built-in functions (IKJCT44B)

This section describes the exit you can use to add your own CLIST built-in functions.

The CLIST language provides built-in functions that can be performed on variables, expressions, and character strings. CLIST processing evaluates the variable or expression, if necessary, and then performs the requested function.

To use a CLIST built-in function, specify its name, followed by the variable, expression or character string in parentheses. The variable, expression or character string is called the *argument* of the built-in function.

z/OS TSO/E CLISTs, describes the CLIST built-in functions that are supplied by TSO/E. In addition to these built-in functions, TSO/E provides a built-in function exit that you can use to add your own CLIST built-in functions.

Processing for a CLIST is done in two steps: phase 1, which is performed by the EXEC command processor, and phase 2. In phase 1, the EXEC command processor reads the CLIST records from the input data set and builds an in-storage command procedure. EXEC then places the command procedure that it built on the input stack. This stack is maintained by TSO/E to determine the source of input. Phase 2 processing receives control as each record is removed from the stack. The CLIST built-in function exit receives control during phase 2 processing. When the exit receives control, all symbolic substitution has been performed on the argument of the built-in function.

Names of installation-written built-in functions must begin with the prefix &SYSX and have from 1 to 248 additional characters. Built-in function names must follow the rules for naming symbolic variables described in *z/OS TSO/E CLISTs*.

When CLIST processing encounters a variable with the &SYSX prefix that has not been defined, it passes control to the CLIST built-in function exit, if one exists. Then, if the exit successfully evaluates the built-in function, CLIST processing replaces the built-in function (name and argument) with the result of the evaluation.

Some ways you can use the CLIST built-in function exit include:
- Performing text manipulation, such as determining the prefix or suffix of a string, or reversing the order of the characters in a string
- Performing arithmetic calculations, such as determining the mean of a series of numbers
- Defining *built-in variables* to return a value. A built-in variable is similar to a built-in function, except that it is *not* followed by an argument. For example, you can use the CLIST built-in function exit to define a built-in variable, &SYSXMVS, that returns the level of MVS installed on your system.

Note: The built-in function exit cannot distinguish between built-in functions and built-in variables. Therefore, do not specify an argument after a built-in variable because errors can result.

TSO/E also provides several other exits that you can use to customize CLIST processing:

- Initialization and termination exits allow you to customize the EXEC command. These exits are described in Chapter 31, "Customizing the EXEC command," on page 277.
- The CLIST statement exit allows you to add your own statements to the CLIST language. This exit is described in "Writing an exit for installation-written statements (IKJCT44S)" on page 499.

TSO/E-supplied exit

TSO/E does not provide a default exit routine for installation-written CLIST built-in functions.

Entry specifications

The contents of the registers on entry for the CLIST built-in function exit are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions

The CLIST built-in function exit receives the standard exit parameter list. However, no data is passed for the command buffer (parameter entry 1). That is, the key field has a value of X'00', the length field has a value of X'04', and the data field has a value of X'00'. For a description of the standard exit parameter list, see "TSO/E standard exit parameter list" on page 32.

Figure 93 on page 497 shows the exit-dependent data that the exit receives beginning at offset +36 in the parameter list. Each parameter entry is described following the figure.

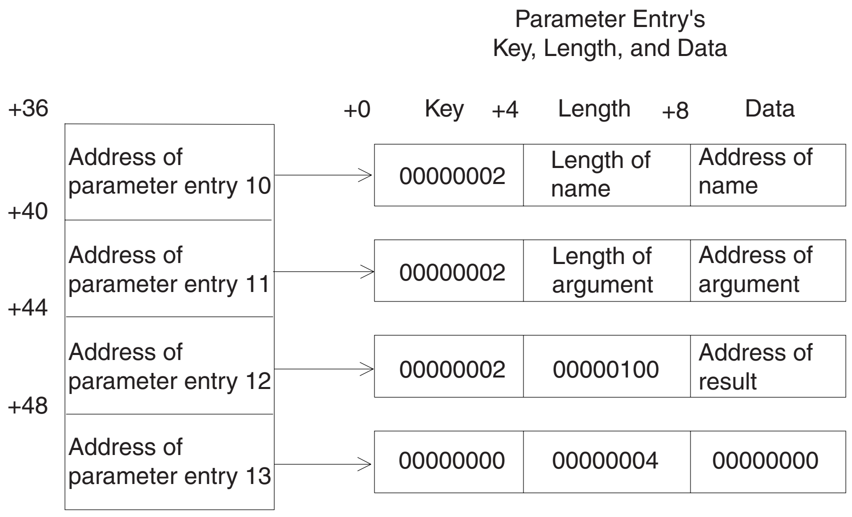


Figure 93. Exit-dependent data for the CLIST built-in function exit

`Name (Parameter Entry 10)`
This parameter entry describes the name of the installation-written built-in function.

`Argument (Parameter Entry 11)`
This parameter entry describes the argument of the built-in function. The argument can be a variable, expression, or character string.

`Result (Parameter Entry 12)`
This parameter describes a 256-byte answer area that is passed to the exit. If your exit successfully evaluates a built-in function, you must place the result into this answer area and update the length field to reflect the length of the result. When the exit returns control, CLIST processing replaces the built-in function with the result.

However, if the result is longer than 256 bytes, your exit must use an alternate buffer to return the result.

`Alternate Buffer Length (Parameter Entry 13)`
This parameter entry describes the length of an alternate buffer used to return the result of evaluating a built-in function. When the result is longer than 256 bytes, and cannot be returned to CLIST processing in the 256-byte answer area, your exit must create an alternate buffer. To pass the result to CLIST processing using an alternate buffer, follow these steps:

1. Obtain the storage for the alternate buffer in subpool 0
2. Place the result into the alternate buffer
3. Update parameter entry 12 as follows:
   - Update the length field to be the actual length of the result that is being passed to CLIST processing
   - Set the data field to the address of the alternate buffer
4. Update parameter entry 13 as follows:
   - Set the key field to X'01' to indicate that the data field contains the actual data for the parameter entry
   - Set the data field to the number of bytes of storage obtained for the alternate buffer

5. Set an exit return code of 4 to indicate to CLIST processing that your exit obtained storage for an alternate buffer. CLIST processing frees the storage obtained for the buffer after it has processed the answer returned by the exit.

Return specifications

The contents of the registers on return from the CLIST built-in function exit must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the CLIST built-in function exit

Table 73 shows the return codes that the CLIST built-in function exit supports.

Table 73. Return codes for the CLIST built-in function exit

Return code (decimal)	Description
0	The exit has successfully evaluated the built-in function and has returned the result in the 256-byte answer area provided. CLIST processing continues.
4	The exit has successfully evaluated the built-in function. The exit has obtained an alternate buffer to return the result of evaluating the built-in function. CLIST processing continues, and frees the storage obtained for the buffer.
8	Exit processing was unsuccessful. The variable was not an installation-written built-in function and could not be evaluated. CLIST processing continues, and initializes the variable to a null value.
12	Exit processing was unsuccessful. The built-in function could not be evaluated. The following action is taken: • CLIST processing issues an error message to the terminal user • Control is passed to the ERROR exit, if one was specified, or CLIST processing terminates • The CLIST return code is set to 604 • The variable &SYSREASON is set to the exit's return code
16	Exit processing was unsuccessful. The built-in function could not be evaluated. The following action is taken: • Control is passed to the ERROR exit, if one was specified, or CLIST processing terminates. However, CLIST processing does *not* issue a message. • The CLIST return code is set to 604 • The variable &SYSREASON is set to the exit's return code If your exit sets a return code of 16, you should consider displaying an informational message to the user. You can use the PUTLINE service routine to issue an informational message. See *z/OS TSO/E Programming Services* for more information.

Note: If the exit returns an undefined return code, CLIST processing terminates without issuing an error message to the user.

Programming considerations

The CLIST built-in function exit must follow standard linkage conventions. It must save the registers on entry and restore the registers when it returns. The exit must be reentrant, refreshable, and reusable.

This exit can use the variable access routine (IKJCT441) to update, create, and retrieve the values of REXX and CLIST variables. For more information about how to use IKJCT441, see *z/OS TSO/E Programming Services*.

Environment

The attributes for the CLIST built-in function exit are:
- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY)

Installing the exit

You must name the CLIST built-in function exit IKJCT44B.

Link-edit the exit as a separate load module. You can link-edit the exit in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exit can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see "Installing the standard-format exits" on page 39.

Writing an exit for installation-written statements (IKJCT44S)

This section describes the exit you can use to add your own CLIST statements.

The CLIST language provides statements that let you structure your programs, perform I/O, define and modify variables, and monitor the conditions under which CLISTs execute. *z/OS TSO/E CLISTs*, describes the CLIST statements that are supplied by TSO/E. In addition to these statements, TSO/E provides an exit that you can use to add your own CLIST statements.

Processing for a CLIST is done in two steps: phase 1, which is performed by the EXEC command processor, and phase 2. In phase 1, the EXEC command processor reads the CLIST records from the input data set and builds an in-storage command procedure. EXEC then places the command procedure that it built on the input stack. This stack is maintained by TSO/E to determine the source of input. Phase 2 processing receives control as each record is removed from the stack. The CLIST statement exit receives control during phase 2 processing.

When CLIST processing encounters a statement that is not one of the CLIST statements supplied by IBM, it passes control to the CLIST statement exit, if one exists. When the exit receives control:
- The statement can be either an installation-written CLIST statement or a TSO/E command
- All symbolic substitution has been performed on the statement

You can use the CLIST statement exit as a fast path for executing TSO/E commands from a CLIST. To avoid the overhead that normally occurs when TSO/E commands are executed from a CLIST, your exit can treat TSO/E commands as installation-written statements. Your exit can directly invoke a TSO/E command processor, using the ATTACH or LINK macro instruction. Depending upon whether the TSO/E command completed successfully, your exit must set the appropriate return code to indicate that an installation-written CLIST statement was processed.

TSO/E also provides several other exits that you can use to customize CLIST processing:

- Initialization and termination exits allow you to customize the EXEC command. These exits are described in Chapter 31, "Customizing the EXEC command," on page 277.
- The CLIST built-in function exit allows you to add your own built-in functions. This exit is described in "Writing an exit for installation-written built-in functions (IKJCT44B)" on page 495.

TSO/E-supplied exit

TSO/E does not provide a default exit routine for installation-written CLIST statements.

Entry specifications

The contents of the registers on entry for the CLIST statement exit are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions

The CLIST statement exit receives the standard exit parameter list. However, no data is passed for the command buffer (parameter entry 1). That is, the key field has a value of X'00', the length field has a value of X'04', and the data field has a value of X'00'. For a description of the standard exit parameter list, see "TSO/E standard exit parameter list" on page 32.

Figure 94 on page 501 shows the exit-dependent data that the exit receives beginning at offset +36 in the parameter list. The parameter entry is described following the figure.

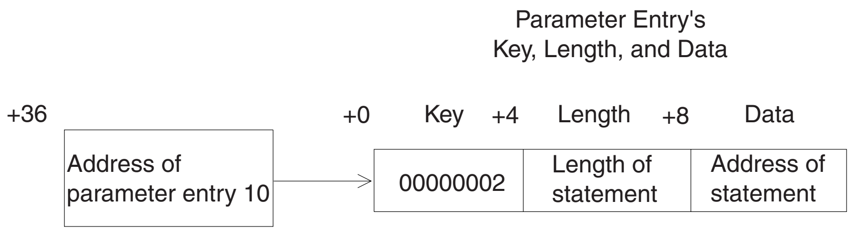


Figure 94. Exit-dependent data for the CLIST statement exit

Statement (Parameter Entry 10)
This parameter entry describes the statement. The statement can be either an installation-written statement or a TSO/E command.

Return specifications

The contents of the registers on return from the CLIST statement exit must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes for the CLIST statement exit

Table 74 shows the return codes that the CLIST statement exit supports.

Table 74. Return codes for the CLIST statement exit

Return code (decimal)	Description
0	The exit has determined that the statement is *not* an installation-written CLIST statement. CLIST processing treats the statement as a TSO/E command.
4	The exit has successfully processed the installation-written CLIST statement.
12	Exit processing for the installation-written CLIST statement was unsuccessful. The following action is taken: • CLIST processing issues an error message to the terminal user • Control is passed to the ERROR routine in the CLIST, if one was specified, or CLIST processing terminates • The CLIST return code is set to 608 • The variable &SYSREASON is set to the exit's return code.
16	Exit processing for the installation-written CLIST statement was unsuccessful. The following action is taken: • Control is passed to the ERROR routine in the CLIST, if one was specified, or CLIST processing terminates. However, CLIST processing does *not* issue a message. • The CLIST return code is set to 608. • The variable &SYSREASON is set to the exit's return code. If your exit sets a return code of 16, you should consider displaying an informational message to the user. You can use the PUTLINE service routine to issue an informational message. See *z/OS TSO/E Programming Services* for more information.

Note: If the exit returns an undefined return code, CLIST processing terminates without issuing an error message to the user.

Programming considerations

The CLIST statement exit must follow standard linkage conventions. It must save the registers on entry and restore the registers when it returns. The exit must be reentrant, refreshable and reusable.

The exit must determine whether the statement should be treated as an installation-written CLIST statement or as a TSO/E command. If the statement is defined by the installation, the exit must perform the necessary processing. Otherwise, the installation must indicate that the statement is a TSO/E command, by setting a return code of 0.

The exit can use the variable access routine (IKJCT441) to update, create and retrieve the value of REXX and CLIST variables. For information on how to use IKJCT441, see *z/OS TSO/E Programming Services*.

Environment

The attributes for the CLIST statement exit are:
- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY)

Installing the exit

You must name the CLIST statement exit IKJCT44S.

Link-edit the exit as a separate load module. You can link-edit the exit in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exit can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see “Installing the standard-format exits” on page 39.

Chapter 42. Customizing REXX processing

This section describes the REXX exits you can use to customize REXX processing. REXX exits are *not* standard-format exits. Some receive non-standard exit parameter lists and some receive no parameter list. Generally, you use TSO/E exits to customize a function or command on a system-wide basis. You use the REXX exits to customize REXX processing on a *language processor environment* basis.

Before you use the REXX exits, review *z/OS TSO/E REXX Reference* to help you understand REXX processing and specifically, language processor environments. This section frequently refers to *z/OS TSO/E REXX Reference*.

Overview of customizing REXX processing

REstructured eXtended eXecutor language (REXX) is a high-level interpretive language that enables you to write programs in a clear and structured way. You can write programs in the REXX language, called *execs*, to perform given tasks or groups of tasks. You can execute REXX execs in TSO/E, ISPF, and in any non-TSO/E address space. REXX execs cannot be invoked authorized in the foreground or the background.

In addition to REXX exits, TSO/E supplies replaceable routines you can use to customize REXX processing and specifically, system services. The following topics give you an overview of the language processor environments and the replaceable routines. For complete information about the language processor environments and the replaceable routines, see *z/OS TSO/E REXX Reference*.

language processor environments

Before the language processor can process an exec, a language processor environment must exist. A *language processor environment* is the environment in which a REXX exec executes. This environment defines how the exec is processed and how the language processor accesses system services. A language processor environment allows the language processor to process (“interpret”) a REXX exec independently of how a specific address space accesses and uses system services.

TSO/E address space

In the TSO/E address space, a default language processor environment is automatically initialized when a user logs on and starts a TSO/E session. The IRXINIT initialization routine initializes language processor environments. When a user invokes ISPF, another language processor environment is initialized. The language processor environment initialized for ISPF is a separate environment from the one that is initialized when the TSO/E session is started. Similarly, if you enter split screen mode in ISPF, another language processor environment is initialized for the second ISPF screen. Therefore, at this point, three separate language processor environments exist. If the user invokes an exec from the second ISPF screen, the exec executes within the language processor environment that was initialized for that second screen. If the user invokes an exec from TSO/E READY mode, the exec executes within the environment that was initialized when the user first logged on.

When the user returns to a single ISPF screen, the language processor environment associated with the second ISPF screen terminates automatically. The IRXTERM termination routine terminates the language processor environment. Similarly, when the user exits from ISPF and returns to TSO/E READY mode, the environment associated with the ISPF screen is terminated. When the user logs off from TSO/E, that language processor environment is then terminated.

Non-TSO/E address spaces

TSO/E provides two programming routines that you can use to run REXX execs, IRXJCL and IRXEXEC.

You can execute a REXX exec in MVS batch by specifying IRXJCL as the program (PGM=) on the JCL EXEC statement.

You can call either IRXJCL or IRXEXEC from a program to execute a REXX exec. You can use the IRXJCL and IRXEXEC routines in any address space. *z/OS TSO/E REXX Reference* describes IRXEXEC and IRXJCL.

Changing the maximum number of language processor environments

The maximum number of environments that the system can initialize in an address space is defined in an environment table called IRXANCHR. The default number of environment table entries is 401. To change the number of environment table entries, you must create a new IRXANCHR load module. TSO/E provides a SYS1.SAMPLIB member called IRXTSMPE, which is an SMP/E user modification (USERMOD) to change the maximum number of language processor environments in an address space. Instructions for using IRXTSMPE are included in the prolog of the IRXTSMPE member in SYS1.SAMPLIB.

Initializing and terminating a language processor environment

The initialization and termination routines (IRXINIT and IRXTERM) are programming routines that you can call to initialize and terminate language processor environments. Although they are primarily intended for use in the non-TSO/E address spaces, you can also use them in TSO/E.

IRXINIT gives you the flexibility to define your own environment, and therefore, *customize* how execs run within the environment. When you call IRXINIT, you specify all the characteristics you want defined for the language processor environment. However, because a language processor environment is always automatically initialized in any address space, you may have no need to use IRXINIT to initialize an environment. If you use IRXINIT to initialize a language processor environment, use IRXTERM to terminate the environment. *z/OS TSO/E REXX Reference* describes IRXINIT and IRXTERM.

Characteristics for a language processor environment

When IRXINIT is called to automatically initialize an environment (such as logon and ISPF screen initialization), it uses default values. TSO/E provides three default *parameters modules* that are load modules containing the default values for initializing language processor environments for TSO/E (READY mode), ISPF, and non-TSO/E address spaces. The parameters modules are:
- IRXTSPRM (for TSO/E)
- IRXISPRM (for ISPF)

- IRXPARMS (for MVS)

You can change the values TSO/E uses to define language processor environments by providing your own load modules. See *z/OS TSO/E REXX Reference* on how to change the default values.

The parameters modules consist of the parameter block (PARMBLOCK), the module name table, the function package table, and the host command environment table. The PARMBLOCK is a control block that is created when an environment is initialized. It contains the values used to define the environment and the addresses of the module name table, the function package table, and the host command environment table.

The following sections briefly describe the fields in the PARMBLOCK that define the characteristics for a language processor environment.

Table 77 on page 510 shows the default values that TSO/E provides in each of the three default parameters modules.

ID An 8-byte character field that is used only to identify the parameter block that IRXINIT creates. The field name is ID. The value must be IRXPARMS.

Version
A 4-byte character field that identifies the version of the parameter block for a particular release and level of TSO/E. The field name is VERSION. The value must be 0200.

Language Code
A 3-byte character field that contains a language code. The field name is LANGUAGE. The language code identifies the language in which REXX messages are displayed. The valid language codes are listed in Table 75.

Table 75. Language codes and their meanings

Language code	Language
CHS	Simplified Chinese
CHT	Traditional Chinese
DAN	Danish
DEU	German
ENP	U.S. English – all uppercase
ENU	U.S. English – mixed case (uppercase and lowercase)
ESP	Spanish
FRA	French
JPN	Japanese (Kanji)
KOR	Korean
PTB	Brazilian Portuguese

Note: To display REXX messages in languages other than U.S. English, you must have the appropriate national language feature of TSO/E installed.

Reserved
A 1-byte field that is reserved.

Module Name Table
A 4-byte field that contains the address of the modulename table. The field

name is MODNAMET. The table contains the names of DDs for reading and writing data and from which to load execs, the names of several replaceable routines, and the names of the following exit routines:
- Exec processing exit (exit for the IRXEXEC routine)
- Exec initialization exit
- Exec termination exit
- Attention handling exit

Table 80 on page 512 shows a summary of the replaceable routines. *z/OS TSO/E REXX Reference* contains the format of the module name table.

Host Command Environment Table
A 4-byte field that contains the address of the host command environment table. The field name is SUBCOMTB. The table contains the names of valid host command environments that execs running in the language processor environment can use. It also contains the names of the routines that handle commands within each host command environment.

Function Package Table
A 4-byte field that contains the address of the function package table for function packages. The field name is PACKTB. The table consists of three parts for user packages, local packages, and system packages. Each part of the table contains a directory of the module names that represent the external functions to be searched.

Token for PARSE SOURCE
An 8-byte character string that contains the value of a token to be used by the PARSE SOURCE instruction. The field name is PARSETOK. The default is blank. This token is the last token of the string that PARSE SOURCE returns. The token you specify is returned in every PARSE SOURCE instruction that is used in the environment.

Flags
A fullword of bits used as flags. The field name is FLAGS. The flags define certain characteristics for the new language processor environment and how the environment and execs executing in the environment operate.

In addition to the flags field, the parameter following the flags is a *mask* field that works together with the flags. The mask field is a string that has the same length as the flags field. Each bit position in the mask field corresponds to a bit position in the flags field. IRXINIT uses the mask field to determine whether the corresponding flag bit should be used or ignored.

Table 76 summarizes each flag. *z/OS TSO/E REXX Reference* describes each of the flags and the bit settings for each flag.

Table 76. Summary of each flag bit

Flag name	Description
TSOFL	Indicates whether the new environment is to be integrated into TSO/E. 0 - The environment is not integrated into TSO/E. 1 - The environment is integrated into TSO/E.
Reserved	This bit is reserved.

Table 76. Summary of each flag bit (continued)

Flag name	Description
CMDSOFL	Specifies the search order used to locate a command that is issued from an exec. 0 - Search for modules first, followed by execs, followed by CLISTs (TSO/E address space only). The ddname used to search for execs is specified in the LOADDD field in the module name table. 1 - Search for execs, followed by modules, followed by CLISTs (TSO/E address space only). The ddname used to search for execs is specified in the LOADDD field in the module name table.
FUNCSOFL	Specifies the search order used to locate functions and subroutines that are called from an exec. 0 - Search the load libraries first. If the function or subroutine is not found, search the exec libraries for an exec. 1 - Search the exec libraries first for an exec. If the exec is not found, search the load libraries.
NOSTKFL	Prevents execs running in the environment from using any data stack functions. 0 - The exec can use any data stack functions. 1 - Requests for data stack functions are processed as though the data stack is empty.
NOREADFL	Prevents execs running in the environment from reading any input file. 0 - Reads from any input file are permitted. 1 - Reads from any input file are not permitted.
NOWRTFL	Prevents execs running in the environment from writing to any output file. 0 - Writes to the output file are permitted. 1 - Writes to the output file are not permitted.
NEWSTKFL	Indicates whether a new data stack is initialized for the new environment. 0 - Do not create a new data stack. 1 - Create a new data stack during the initialization of the new language processor environment.
USERPKFL	Indicates whether the user function packages that are defined for the previous language processor environment are also available to the new environment. 0 - Add the user function packages from the previous environment to the user function packages for the new environment. 1 - Do not add the user function packages from the previous environment to the user function packages for the new environment.

Table 76. Summary of each flag bit (continued)

Flag name	Description
LOCPKFL	Indicates whether the local function packages that are defined for the previous language processor environment are also available to the new environment. 0 - Add the local function packages from the previous environment to the local function packages for the new environment. 1 - Do not add the local function packages from the previous environment to the local function packages for the new environment.
SYSPKFL	Indicates whether the system function packages that are defined for the previous language processor environment are also available to the new environment. 0 - Add the system function packages from the previous environment to the system function packages for the new environment. 1 - Do not add the system function packages from the previous environment to the system function packages for the new environment.
NEWSCFL	Indicates whether the host command environments that are defined for the previous language processor environment (in the host command environment table) are also available to the new environment. 0 - Add the host command environments from the previous environment to the host command environment table for the new environment. 1 - Do not add the host command environments from the previous environment to the host command environment table for the new environment.
CLOSEXFL	Indicates whether the data set as specified in the LOADDD field in the module name table is closed after the exec completes processing or remains open. 0 - The data set is opened once and remains open. 1 - The data set is opened for each load and then closed.
NOESTAE	Indicates whether a recovery ESTAE is established under the new environment. 0 - A recovery ESTAE is established. 1 - A recovery ESTAE is not established.
RENTRANT	Indicates whether the new environment is initialized as reentrant or non-reentrant. 0 - A non-reentrant language processor environment is initialized. 1 - A reentrant language processor environment is initialized.
NOPMSGS	Indicates whether primary messages are printed. 0 - Primary messages are printed. 1 - Primary messages are not printed.

Table 76. Summary of each flag bit (continued)

Flag name	Description
ALTMSGS	Indicates if alternate messages are printed. 0 - Alternate messages are not printed. 1 - Alternate messages are printed.
SPSHARE	Indicates if the subpool specified in SUBPOOL field is shared across MVS tasks. 0 - The subpool is not shared. 1 - The subpool is shared.
STORFL	Indicates whether execs running in the environment can use the TSO/E STORAGE function. 0 - The STORAGE function can be used. 1 - The STORAGE function cannot be used.
NOLOADDD	Indicates whether the data set specified in the LOADDD field in the module name table is to be searched. 0 - Search the ddname, such as SYSEXEC, specified in the LOADDD field. If the exec is not found, search SYSPROC. 1 - Search SYSPROC only. **Note:** SYSPROC is searched only if the language processor environment is integrated into TSO/E. The user can either use the EXECUTIL command to indicate whether to search the exec ddnames or use the ALTLIB command to explicitly activate them. *z/OS TSO/E REXX Reference* describes the EXECUTIL command. *z/OS TSO/E Command Reference* describes the ALTLIB command.
NOMSGWTO	Indicates whether to route REXX messages to a file in an MVS environment. SYSTSPRT is the default file name. 0 - Messages are processed. 1 - Messages are routed to the SYSTSPRT file.
NOMSGIO	Indicates whether to route REXX messages to a JCL listing. 0 - Messages are processed. 1 - Messages are routed to the JCL listing.
Reserved	The remaining bits are reserved.

`Mask`
A fullword of bits used as a mask for the setting of the flag bits. The field name is MASKS. The mask field is a string that has the same length as the flags field. Each bit position in the mask field corresponds to a bit in the same position in the flags field. IRXINIT uses the mask field to determine whether the corresponding flag bit is used or ignored.

`Subpool Number`
A fullword of binary numbers that specifies the number of the subpool in which storage is allocated for the entire language processor environment. The field name is SUBPOOL. If the environment is integrated into TSO/E, this value must be decimal 78.

Address Space Name
An 8-byte character field that specifies the name of the address space. The field name is ADDRSPN. The defaults are MVS (for IRXPARMS), TSO/E (for IRXTSPRM), and ISPF (for IRXISPRM).

X'FFFFFFFFFFFFFFFF'
The end of the parameter block (PARMBLOCK) must be indicated by X'FFFFFFFFFFFFFFFF'.

The following four tables show the default values that TSO/E provides in each of the default parameters modules. In Table 77, the LANGUAGE field contains the language code ENU for U.S. English in mixed case (uppercase and lowercase). The default parameters modules may contain a different language code depending on whether one of the language features has been installed. See Table 75 on page 505 for information about the different language codes.

Note: The LANGUAGE field in the parameters module IRXISPRM (for ISPF) contains nulls. For ISPF, the system uses the language default from the parameters module IRXTSPRM (for TSO/E).

Table 77. Values TSO/E provides in the default parameters modules

Field name	IRXPARMS (MVS)	IRXTSPRM (TSO/E)	IRXISPRM (ISPF)
ID	IRXPARMS	IRXPARMS	IRXPARMS
VERSION	0200	0200	0200
LANGUAGE	ENU	ENU	
PARSETOK			
FLAGS			
TSOFL	0	1	1
CMDSOFL	0	0	0
FUNCSOFL	0	0	0
NOSTKFL	0	0	0
NOREADFL	0	0	0
NOWRTFL	0	0	0
NEWSTKFL	0	0	1
USERPKFL	0	0	0
LOCPKFL	0	0	0
SYSPKFL	0	0	0
NEWSCFL	0	0	0
CLOSEXFL	0	0	0
NOESTAE	0	0	0
RENTRANT	0	0	0
NOPMSGS	0	0	0
ALTMSGS	1	1	0
SPSHARE	0	1	1
STOREFL	0	0	0
NOLOADDD	0	0	0
NOMSGWTO	0	0	0
NOMSGIO	0	0	0
MASKS			
TSOFL_MASK	1	1	1
CMDSOFL_MASK	1	1	0
FUNCSOFL_MASK	1	1	0
NOSTKFL_MASK	1	1	0
NOREADFL_MASK	1	1	0
NOWRTFL_MASK	1	1	0
NEWSTKFL_MASK	1	1	1
USERPKFL_MASK	1	1	0
LOCPKFL_MASK	1	1	0

Table 77. Values TSO/E provides in the default parameters modules (continued)

Field name	IRXPARMS (MVS)	IRXTSPRM (TSO/E)	IRXISPRM (ISPF)
SYSPKFL_MASK	1	1	0
NEWSCFL_MASK	1	1	0
CLOSEXFL_MASK	1	1	0
NOESTAE_MASK	1	1	0
RENTRANT_MASK	1	1	0
NOPMSGS_MASK	1	1	0
ALTMSGS_MASK	1	1	0
SPSHARE_MASK	1	1	1
STOREFL_MASK	1	1	0
NOLOADDD_MASK	1	1	0
NOMSGWTO_MASK	1	1	0
NOMSGIO_MASK	1	1	0
SUBPOOL	0	78	78
ADDRSPN	MVS	TSO/E	ISPF
---	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF

Table 78. Values TSO/E provides in the field name in the module name table

Field name in module name table	IRXPARMS (MVS)	IRXTSPRM (TSO/E)	IRXISPRM (ISPF)
INDD	SYSTSIN	SYSTSIN	
OUTDD	SYSTSPRT	SYSTSPRT	
LOADDD	SYSEXEC	SYSEXEC	
IOROUT			
EXROUT			
GETFREER			
EXECINIT			
ATTNROUT			
STACKRT			
IRXEXECX			
IDROUT			
MSGIDRT			
EXECTERM			
---	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF

Table 79. Values TSO/E provides in the field name in the host command environment table

Field name in host command environment table	IRXPARMS (MVS)	IRXTSPRM (TSO/E)	IRXISPRM (ISPF)
TOTAL	9	11	13
USED	9	11	13
LENGTH	32	32	32
INITIAL	MVS	TSO	TSO
---	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF
Entry 1			
NAME	MVS	MVS	MVS
ROUTINE	IRXSTAM	IRXSTAM	IRXSTAM
TOKEN			
Entry 2			
NAME	LINK	TSO	TSO
ROUTINE	IRXSTAM	IRXSTAM	IRXSTAM
TOKEN			
Entry 3			
NAME	ATTACH	LINK	LINK
ROUTINE	IRXSTAM	IRXSTAM	IRXSTAM

Table 79. Values TSO/E provides in the field name in the host command environment table (continued)

Field name in host command environment table	IRXPARMS (MVS)	IRXTSPRM (TSO/E)	IRXISPRM (ISPF)
TOKEN			
Entry 4			
NAME	CPICOMM	ATTACH	ATTACH
ROUTINE	IRXSTAM	IRXSTAM	IRXSTAM
TOKEN			
Entry 5			
NAME	LU62	CONSOLE	ISPEXEC
ROUTINE	IRXSTAM	IRXSTAM	IRXSTAM
TOKEN			
Entry 6			
NAME	LINKMVS	CPICOMM	ISREDIT
ROUTINE	IRXSTAMP	IRXAPPC	IRXSTAM
TOKEN			
Entry 7			
NAME	LINKPGM	LU62	CONSOLE
ROUTINE	IRXSTAMP	IRXAPPC	IRXSTAM
TOKEN			
Entry 8			
NAME	ATTCHMVS	LINKMVS	CPICOMM
ROUTINE	IRXSTAMP	IRXSTAMP	IRXAPPC
TOKEN			
Entry 9			
NAME	ATTCHPGM	LINKPGM	LU62
ROUTINE	IRXSTAMP	IRXSTAMP	IRXAPPC
TOKEN			
Entry 10			
NAME		ATTCHMVS	LINKMVS
ROUTINE		IRXSTAMP	IRXSTAMP
TOKEN			
Entry 11			
NAME		ATTCHPGM	LINKPGM
ROUTINE		IRXSTAMP	IRXSTAMP
Entry 12			
NAME			ATTCHMVS
ROUTINE			IRXSTAMP
TOKEN			
Entry 13			
NAME			ATTCHPGM
ROUTINE			IRXSTAMP
TOKEN			

Table 80. Values TSO/E provides in the field name in the function package table

Field name in function package table	IRXPARMS (MVS)	IRXTSPRM (TSO/E)	IRXISPRM (ISPF)
USER_TOTAL			
USER_USED	1	1	1
LOCAL_TOTAL	1	1	1
LOCAL_USED	1	1	1
SYSTEM_TOTAL	1	2	2
SYSTEM_USED	1	2	2
LENGTH	8	8	8
---	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF
Entry 1			
NAME	IRXEFMVS	IRXEFMVS	IRXEFMVS

Table 80. Values TSO/E provides in the field name in the function package table (continued)

Field name in function package table	IRXPARMS (MVS)	IRXTSPRM (TSO/E)	IRXISPRM (ISPF)
Entry 2			
NAME	IRXFLOC	IRXEFPCK	IRXEFPCK
Entry 3			
NAME	IRXFUSER	IRXFLOC	IRXFLOC
Entry 4			
NAME		IRXFUSER	IRXFUSER

Replaceable routines

In addition to the REXX exits, TSO/E supplies replaceable routines you can use to customize REXX processing. For exec processing, various system services are used for loading and freeing execs, performing I/O, obtaining and freeing storage, and handling data stack requests. TSO/E provides routines that handle these types of system services, which you can replace. The names of the routines are defined in the module name table for each particular language processor environment. In non-TSO/E address spaces, you can provide your own replaceable routines. In the TSO/E address space, you can provide your own replaceable routines only if you initialize a language processor environment that is not integrated into TSO/E, that is, the TSOFL flag is off.

Your routine can check the request for a system service, change the request if needed, and then call the system-supplied routine to actually perform the service. Your routine can perform the request itself and not call the system-supplied routine. It can also terminate the request for a system service. You can specify the routine name either by:

- Providing your own parameters module(s)
- Calling the IRXINIT routine and passing the name of the routine on the call

Table 81 provides a brief description of the replaceable routines. They are described in detail in *z/OS TSO/E REXX Reference*.

Table 81. Summary of replaceable routines

Replaceable routine	Description
Exec load	This routine is called to load an exec into storage and to free the exec when it is no longer needed.
Input/Output (I/O)	This routine is called to read a record from or write a record to a specified ddname. For example, this routine is called for the SAY instruction, the PULL instruction (when the data stack is empty), and for the EXECIO command. The routine is also called to open and close a data set.
Data stack	This routine is called to handle any requests for data stack services. For example, it is called for the PULL, PUSH, and QUEUE instructions and for the MAKEBUF and DROPBUF commands.
Storage management	This routine is called to obtain and free storage.
User ID	This routine is called to obtain the user ID or terminal ID. The result it obtains is returned by the USERID built-in function.
Message identifier	This routine determines if the message identifier (message ID) is displayed with a REXX error message.

Table 81. Summary of replaceable routines (continued)

Replaceable routine	Description
Host command environment	This routine is called to handle the execution of host commands for a specific host command environment.

TSO/E REXX exits

TSO/E provides the following types of exits for REXX processing:
- TSO/E-supplied exits
- Installation-supplied exits

The TSO/E-supplied exits are default exits. The names of these exits are fixed and cannot be changed. You can either use the default exits or provide your own exits to suit your installation's needs.

For the installation-supplied exits, specify names you have chosen for the exits in the appropriate fields of the module name table.

TSO/E-supplied exits

The following section describes how you can use the TSO/E-supplied exits. It also describes the default processing of the exits if you do not provide your own exits.

- **Pre-environment initialization exit (IRXINITX)**

  This exit is called by the IRXINIT routine. IRXINITX performs exit processing before a new language processor environment is initialized. It is invoked before an environment is initialized and before parameters are evaluated. Use IRXINITX to prevent the initialization of a language processor environment, change parameters used for initializing a language processor environment, or perform special pre-environment initialization processing.

  Default: IRXINITX does not prevent the initialization of a language processor environment, does not alter the parameters used for initializing a language processor environment, and does not perform special pre-environment initialization processing. It sets a return code of 0 and returns.

- **Post-environment initialization exit (IRXITTS or IRXITMV)**

  IRXITTS is called by the IRXINIT routine and performs exit processing after the language processor environment is initialized for an environment integrated into TSO/E. It is invoked after the environment is initialized and after the control blocks, such as the environment block, are set up. Use IRXITTS to perform special processing for a newly-initialized language processor environment.

  Default: IRXITTS does not prevent the initialization of a language processor environment and does not perform any special initialization processing. It sets a return code of 0 and returns.

  IRXITMV is called by the IRXINIT routine and performs exit processing after a language processor environment is initialized for an environment not integrated into TSO/E. It is invoked after the environment is initialized and after the control blocks, such as the environment block, are set up. Use IRXITMV to perform special processing for a newly-initialized language processor environment.

  Default: IRXITMV does not prevent the initialization of a language processor environment and does not perform any special initialization processing. It sets a return code of 0 and returns.

- **Environment termination exit (IRXTERMX)**

IRXTERMX is called by the IRXTERM routine and performs exit processing before a language processor environment is terminated. It is invoked before each language processor environment is terminated. Use IRXTERMX to prevent the termination of a language processor environment or perform any special termination processing for a language processor environment.

Default: IRXTERMX does not prevent the termination of a language processor environment and does not perform any special termination processing. It sets a return code of 0 and returns.

Installation-supplied exits

You provide the following exits by specifying names you have chosen in the appropriate fields of the *module name table*:

- **Exec processing exit**

  This exit is called by the IRXEXEC routine. It is invoked before the exec is loaded, if the exec was not pre-loaded, and before IRXEXEC evaluates any parameters on the call. Use the exit to prevent the execution of a REXX exec or perform special processing before a REXX exec is executed. Specify the exit's name in the IRXEXECX field in the module name table.
- **Exec initialization exit**

  This exit is called by the IRXEXEC routine or a compiler runtime processor. Use the exit to update and access REXX variables. The exit gets control after the REXX variable pool has been initialized for an exec, but before the first clause in the exec is processed. Specify the exit's name in the EXECINIT field in the module name table.
- **Exec termination exit**

  This exit is called by the IRXEXEC routine or a compiler runtime processor. Use the exit to update and access REXX variables. The exit gets control after the last clause in the exec is processed, but before the REXX variable pool is terminated. Specify the exit's name in the EXECTERM field in the module name table.
- **Attention handling exit**

  This exit can only be used for an environment integrated into TSO/E. It is called if an exec is executing and an attention interruption occurs. Use the exit to perform special attention processing. Specify the exit's name in the ATTNROUT field in the module name table.

Entry specifications

The contents of the registers on entry for the **IRXINITX exit** and the **exec processing exit** (the exit that is invoked by IRXEXEC) are:

Register 0
: Same as on entry to the IRXINIT initialization routine (IRXINITX)

  Address of the current environment block (IRXEXEC exit)

Register 1
: For the pre-environment initialization exit (IRXINITX), address of the parameter list passed to the IRXINIT routine

  For the exec processing exit for IRXEXEC, address of the parameter list passed to the IRXEXEC routine

Registers 2–12
: Unpredictable

Register 13
: Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

The contents of the registers on entry for the **IRXITTS exit** (integrated into TSO/E) and the **IRXITMV exit** (not integrated into TSO/E) are:

Register 0
Address of the new environment block

Registers 1–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

The contents of the registers on entry for the **IRXTERMX exit** are:

Register 0
Address of the terminating environment block

Registers 1–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

The contents of the registers on entry for the **exec initialization exit**, **exec termination exit**, and the **attention handling exit** are:

Register 0
Address of the current environment block

Registers 1–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

When an environment is initialized, the system creates the environment block (ENVBLOCK) that contains pointers to several other control blocks. Together, these control blocks define all the characteristics of the environment. The address of the environment block is passed in register 0 in all calls to REXX exits and routines, and in all calls to the REXX compiler runtime processor and compiler interface routines. Note that you can only read information from the environment block or the control blocks to which the environment block points. If you change the values, results are unpredictable. *z/OS TSO/E REXX Reference* contains the format of the various control blocks.

Parameter descriptions

The following REXX exits do *not* receive parameter lists:

- For IRXITTS (TSO/E) and IRXITMV (non-TSO/E):

  Register 0 contains the address of the new environment block. Register 1 does not point to a parameter list.
- For IRXTERMX:

  Register 0 contains the address of the terminating environment block. Register 1 does not point to a parameter list.
- For exec initialization and exec termination:

  Register 0 contains the address of the current environment block. Register 1 does not point to a parameter list.
- For attention handling:

  Register 0 contains the address of the current environment block. Register 1 does not point to a parameter list.

  The environment block pointed to by register 0 contains a field (ENVBLOCK_ATTNROUT_PARMPTR) that contains the address of an attention handling routine control block. The attention handling exit can use this control block to communicate with REXX attention processing. For more information on the format of the attention handling routine control block, see Table 82.

The IRXINITX pre-environment initialization exit and the exec processing exit (exit for IRXEXEC) receive *non-standard* parameter lists. They are described in the following sections.

Attention handling control block

The attention handling exit is invoked when a REXX exec is running and the user presses the attention interruption key (usually the PA1 key). The exit receives control before REXX attention processing issues a prompting message (IRX0920I), which instructs the user to either enter a null line to continue running the exec or enter one of the immediate commands.

When the exit receives control, register 0 contains the address of the language processor environment control block, which is mapped by the IRXENVB mapping macro. The ENVBLOCK_ATTNROUT_PARMPTR field contains the address of a control block, which the exit may optionally use to communicate with REXX attention processing. Table 82 shows the format of the attention handling routine control block.

Table 82. Format of the attention handling routine control block

Offset (decimal)	Number of bytes	Field name	Description
0	2	LEN	Length of this control block
2	2	VERS	Version number

Table 82. Format of the attention handling routine control block (continued)

Offset (decimal)	Number of bytes	Field name	Description
4	1	FLAG1	Flag word: • X'80' — May be set by an exit to tell the REXX attention processor to suppress the REXX attention prompting message. • X'40' — May be set by an exit to tell the REXX attention processor that the HE (halt execution) immediate command should not be allowed. When this flag is set, HE is treated as an incorrect response to the REXX attention prompt message.
5	3	-----	Reserved
8	4	EXITRC	Exit return code field

Table 83 lists the valid return codes that may be returned from the attention handling exit using the EXITRC field.

Table 83. Valid return codes for the attention handling exit routine

Return code (decimal)	Description
0	Exit processing was successful. No attention handling requests have been specified by the exit. REXX attention processing should continue with normal attention processing.
4	Exit processing was successful. The attention handling exit has set special request flags for REXX attention processing.
8	Exit processing was successful. REXX should continue processing without performing any additional attention processing.

Parameters for the IRXINITX exit

The parameter list for IRXINITX is the same as the parameter list for the IRXINIT initialization routine. TSO/E REXX passes the address of an environment block in register 0. In register 1, TSO/E REXX passes the address of a parameter list, which consists of a list of addresses. Each address in the parameter list points to a parameter.

TSO/E REXX passes the addresses of at least the first seven parameters. The addresses that point to parameter 8 and parameter 9 are optional. The high-order bit of the last address in the parameter list is set to 1 to indicate the end of the parameter list. Table 84 on page 519 describes the parameter list that IRXINITX receives.

Table 84. Parameters for the IRXINITX exit

Parameter	Number of bytes	Description
Parameter 1	8	This parameter specifies the function to be performed: • **INITENVB** -- to initialize a new environment. • **FINDENVB** -- to obtain the address of the environment block for the current non-reentrant environment. FINDENVB returns the address of the environment block in register 0 and in parameter 6. It does not initialize a new environment.
Parameter 2	8	The name of the *parameters module*, which contains the values for initializing the new environment. On the call to the IRXINIT initialization routine, the caller may have passed a blank in this field. Therefore, IRXINIT assumes that all the fields in the parameters module are null. The parameters module is described in "Characteristics for a language processor environment" on page 504. IRXINIT provides two ways in which you can pass parameter values; the parameters module and the address of an in-storage parameter list, which is parameter 3.
Parameter 3	4	The address of an *in-storage parameter list*, which is an area in storage containing parameters that are equivalent to the parameters in the parameters module. The format of the in-storage list is identical to the format of the parameters module. This parameter may be 0. If the address is 0, IRXINIT assumes that all fields in the in-storage parameter list are null.
Parameter 4	4	The address of a user field. IRXINIT does not use or check this pointer or the field. You can use this field for your own processing.
Parameter 5	4	A 4-byte field that is reserved.
Parameter 6	4	This parameter is only used for output by IRXINIT and should not be altered by this exit. It contains the address of the environment block. If you use the FINDENVB parameter to locate an environment, this parameter contains the address of the environment block for the current non-reentrant environment. If you use INITENVB to initialize a new environment, IRXINIT returns the address of the environment block for the newly created environment in this parameter. For either FINDENVB or INITENVB, IRXINIT also returns the address of the environment block in register 0. This parameter lets higher-level languages obtain the environment block address to examine information in the environment block.
Parameter 7	4	This parameter is only used for output by IRXINIT and should not be altered by this exit. IRXINIT returns a reason code in this field, which indicates why the requested function did not complete successfully. *z/OS TSO/E REXX Reference* describes the reason codes that may be returned.

Table 84. Parameters for the IRXINITX exit (continued)

Parameter	Number of bytes	Description
Parameter 8	4	Parameter 8 is an optional parameter that lets you specify how REXX obtains storage in the language processor environment. Specify 0 if you want the system to reserve a default amount of storage workarea. If you want to pass a storage workarea to IRXINIT, specify the address of an *extended parameter list*. The extended parameter list consists of the address (a fullword) of the storage workarea and the length (a fullword) of the workarea, followed by X'FFFFFFFFFFFFFFFF'. For more information on how REXX obtains workarea storage, see *z/OS TSO/E REXX Reference*.
Parameter 9	4	This parameter is only used for output by IRXINIT and should not be altered by this exit. It is a 4-byte field that IRXINIT uses to return the return code.

Parameters for the exec processing exit for the IRXEXEC routine

When the exec processing exit for the IRXEXEC routine gets control, register 1 points to a parameter list. This parameter list consists of a list of addresses. Each address in the parameter list points to a parameter. Table 85 describes the parameter list that the exec processing exit receives.

Table 85. Parameters for the exec processing exit

Parameter	Number of bytes	Description
Parameter 1	4	The address of the exec block (EXECBLK). The exec block is a control block that describes the exec to be loaded. It contains information needed to process the exec, such as the DD from which the exec is to be loaded and the name of the initial host command environment when the exec starts executing. This parameter can be 0 if the exec is pre-loaded and the address of the pre-loaded exec is passed in parameter 4. If this parameter and parameter 4 are both specified, the value in parameter 4 is used and this parameter is ignored.
Parameter 2	4	The address of the arguments for the exec. The arguments are arranged as a vector of address/length pairs followed by X'FFFFFFFFFFFFFFFF'. There is no limit to the number of arguments passed to the exec.

Table 85. Parameters for the exec processing exit (continued)

Parameter	Number of bytes	Description
Parameter 3	4	A fullword of bits that are used as flags. Only bits 0, 1, and 2 are used. The remaining bits are reserved. Bits 0, 1, and 2 are mutually exclusive. • Bit 0 - If the bit is set on, the exec was invoked as a “command”, that is, it was not invoked from another exec as an external function or subroutine. • Bit 1 - If the bit is set on, the exec was invoked as an external function (a function call). • Bit 2 - If the bit is set on, the exec was invoked as a subroutine. • Bit 3 - Set this bit on if you want IRXEXEC to return *extended return codes* in the range 20001–20099. If a syntax error occurs, IRXEXEC returns a value in the range 20001–20099 in the evaluation block, regardless of the setting of bit 3. If bit 3 is on and a syntax error occurs, IRXEXEC returns with a return code in the range 20001–20099 that matches the value returned in the evaluation block. If bit 3 is off and a syntax error occurs, IRXEXEC returns with return code 0. For more information about how REXX returns information about syntax errors, see *z/OS TSO/E REXX Reference*.
Parameter 4	4	The address of the *in-storage control block* (INSTBLK). The in-storage control block defines the structure of a pre-loaded exec in storage. It contains pointers to each record in the exec and the length of each record. This parameter is specified if the caller of the IRXEXEC routine has pre-loaded the exec. Otherwise, this parameter is 0.
Parameter 5	4	A 4-byte field that contains the address of the CPPL, if IRXEXEC was called from the TSO/E address space. If the caller of IRXEXEC does not pass the address of the CPPL, TSO/E builds the CPPL without a command buffer. The CPPL address is required in the TSO/E address space. If IRXEXEC is called from a non-TSO/E address space, this parameter is 0.
Parameter 6	4	The address of an evaluation block (EVALBLOCK). IRXEXEC uses the evaluation block to return the result from the exec that was specified on either the RETURN or EXIT instruction. The value may be 0, if the exec does not return a result or the caller of IRXEXEC plans to use the IRXRLT (get result) routine to get the result or the result is to be ignored. *z/OS TSO/E REXX Reference* describes IRXRLT.

Table 85. Parameters for the exec processing exit (continued)

Parameter	Number of bytes	Description
Parameter 7	4	The address of an 8-byte field that defines a work area. In the 8-byte field, the: • First four bytes contain the address of the work area • Second four bytes contain the length of the work area The work area is passed to the language processor to use for executing the exec. If the work area is too small, IRXEXEC returns with a return code of 20 and a message indicates an error. The minimum length required for the work area is X'1800' bytes. If you do not want to pass a work area, specify an address of 0. IRXEXEC will obtain storage for its work area or will call the replaceable storage routine specified in the GETFREER field (in the module name table) for the environment, if you provided a storage routine.
Parameter 8	4	The address of a user field. IRXEXEC does not use or check this pointer or the user field. You can use this field for your own processing. If you do not want to use a user field, specify an address of 0.

Return specifications

The contents of the registers on return from all the REXX exits must be:

Registers 0
Same as on entry, except for IRXINITX. See description below.

Registers 1–14
Same as on entry

Register 15
Return code

For the IRXINITX exit, register 0 contains the same values passed to the IRXINIT initialization routine. IRXINIT uses register 0 to locate the previous environment block, reentrant or non-reentrant. Therefore, altering register 0 controls how IRXINIT locates the previous environment block. If you change register 0 and do not restore it, TSO/E uses the new value to locate the previous environment block.

Return codes for the REXX exits

Table 86 shows the return codes that the IRXINITX, IRXITTS, IRXITMV, IRXTERMX, and the exec processing exit support.

Table 86. Return codes for the REXX exits

Return code (decimal)	Description
0	Exit processing was successful. REXX processing continues.
Non-zero	Exit processing was unsuccessful. REXX processing terminates and sets register 15 to 20.

Table 87 shows the return codes the exec initialization and exec termination exits support.

Table 87. Return codes for the exec initialization and termination exits

Return code (decimal)	Description
0	Exit processing was successful. REXX processing continues.
Non-zero	Exit processing was unsuccessful. The exec is not executed. REXX issues a message that indicates a failure in a system service.

Note that the attention handling exit does not support any return codes in register 15. It does support a return code placed in the attention handling routine control block. See "Attention handling control block" on page 517 for more information on the attention handling control block.

Programming considerations

The REXX exits must follow standard linkage conventions. They must have attributes of AMODE(31) and RMODE(ANY). They must save the registers on entry and restore the registers when they return. They must not be APF-authorized. They must be reentrant and refreshable.

The exits can use any of the TSO/E service routines if they are integrated into TSO/E. In non-TSO/E address spaces, the exits cannot use the TSO/E service routines, such as PARSE, SCAN, GETLINE, and PUTLINE. For a description of the TSO/E service routines, see *z/OS TSO/E Programming Services*.

Any of the TSO/E REXX service routines, such as IRXIC, IRXSUBCM, and IRXEXCOM, are available in non-TSO/E and TSO/E address spaces. For a description of the REXX service routines, see *z/OS TSO/E REXX Reference*.

Environment

The attributes for all the REXX exits are:
- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY)
- ASC Mode: Primary
- Task Mode
- Reentrant, refreshable

Installing the exits

TSO/E supplies default exit routines as described in "TSO/E-supplied exits" on page 514. The names of these default exit routines are fixed and cannot be changed:

Pre-environment initialization exit
IRXINITX

Post-environment initialization exits
IRXITTS (for TSO/E), IRXITMV (for non-TSO/E)

Environment termination exit
IRXTERMX

For the following exits, specify the name you have chosen for them in the module name table:
- IRXEXECX field for the exec processing exit for the IRXEXEC routine

- ATTNROUT field for the attention handling exit
- EXECINIT field for the exec initialization exit
- EXECTERM field for the exec termination exit

The IRXINITX, IRXITTS, IRXITMV, and IRXTERMX exits must be link-edited with the IRXINIT initialization routine.

Link-edit the exit for IRXEXEC, the attention handling exit, and the exec initialization and exec termination exits as separate load modules or as aliases of other load modules. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. All the exits can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

If you define any of the TSO/E-supplied REXX exits or the installation-defined exits, and if these exits are intended for use with the TSO REXX environment created by the TSO TMP (that is, they are defined in the REXX default parameters module IRXTSPRM), then these exits must be placed in an APF-authorized library. However, note that they *should not* be link-edited APF-authorized, and they do not receive control APF-authorized.

For more information about using the LPA, LNKLST, or STEPLIB, see “Installing the standard-format exits” on page 39.

Possible uses

The following describes how you can use the REXX exits.

IRXINITX

- Use this exit to prevent language processor environments from being initialized. For example, if you want to prevent more than *n* number of environments from being initialized, use IRXINITX. When the number reaches *n*, IRXINITX can return a non-zero return code to prevent the initialization.
- You can also use IRXINITX to alter the parameters used for initializing a language processor environment. For example, if you want to prevent REXX execs from using the STORAGE function, use IRXINITX to set a flag to indicate that the STORAGE function is not allowed under the language processor environment.

IRXITTS

- Use this exit to perform special initialization processing for a language processor environment that is integrated into TSO/E. For example, if you want to keep track of the number of REXX execs that have been executed under the language processor environment, use IRXITTS to initialize the count to zero.

IRXITMV

- Use this exit to perform special initialization processing for a language processor that is not integrated into TSO/E. For example, if you want to use a control block to communicate between host commands, use IRXITMV to initialize the control block and point to it from the user field in the environment block.

IRXTERMX

- Use this exit to perform special termination processing before a language processor environment is terminated. If you use a control block to communicate between host commands, use IRXTERMX to free the storage for the control block.

Exec processing (IRXEXEC exit)

- Use this exit to perform special processing before the IRXEXEC routine executes the REXX exec. For example, if you want to prevent certain REXX execs from writing output, use the exit to set a flag indicating that no output is allowed.

Exec initialization

- Use this exit to access and update REXX variables. For example, you can use it to initialize variables.

Exec termination

- Use this exit to access and update REXX variables. For example, you can use it to obtain the values of certain variables upon the completion of a REXX exec.

Attention handling

Use this exit to perform special attention processing. For example, you can use this exit to:

- Log off the user immediately after an attention interruption occurs.
- Invoke the EXECUTIL HI command or the HI function of IRXIC to halt the interpretation of the exec
- Request that REXX attention processing not display the attention prompting message
- Disable the use of the HE immediate command during REXX attention processing.

Part 7. Support for a REXX compiler

This part describes the support that TSO/E provides for the installation and execution of a REXX compiler.

- Routines and Interfaces to Support a REXX Compiler

  TSO/E provides a defined interface to support the installation and execution of a REXX compiler runtime processor. This support consists of routines and interfaces that TSO/E REXX uses during the execution of compiled execs under a compiler runtime processor.

  Central to TSO/E REXX compiler support is the compiler programming table. TSO/E REXX uses the compiler runtime processor name stored in the compiled REXX exec to locate the entry for the compiler runtime processor in the compiler programming table. The compiler programming table entry contains the name of the compiler runtime processor and the names of up to four optional compiler interface routines. TSO/E REXX uses the compiler runtime processor to run compiled execs. During the execution of a compiled exec, TSO/E REXX invokes compiler interface routines (if installed) to perform specialized processing.
- Programming Routines for a REXX compiler runtime processor

  TSO/E provides various programming routines that support a REXX compiler runtime processor. These routines are:
  - IRXERS - a REXX compiler programming routine that searches for and runs an external routine. For more information on the search order for external routines, refer to *z/OS TSO/E REXX Reference*.
  - IRXHST - a REXX compiler programming routine that searches for and runs a host command. For more information on locating host commands, refer to *z/OS TSO/E REXX Reference*.
  - IRXRTE - a REXX compiler programming routine that searches for and invokes a REXX exit routine. For more information on REXX exit routines, refer to *z/OS TSO/E REXX Reference*.

  In addition, you can use the GETEVAL function of the IRXRLT programming service to obtain the evaluation block for an external function or subroutine. These routines and the GETEVAL function of IRXRLT are intended for use only by a compiler runtime processor. For more information on the IRXRLT programming service, see *z/OS TSO/E REXX Reference*.

Chapter 43. Routines and interfaces to support a REXX compiler

This chapter discusses the characteristics of a compiled REXX exec and the routines and interfaces that TSO/E provides to support a REXX compiler, including:
- The compiler programming table
- The compiler runtime processor
- The four compiler interface routines:
  - compiler interface initialization routine
  - compiler interface termination routine
  - compiler interface load routine
  - compiler interface variable handling routine

Overview of routines and interfaces to support a REXX compiler

TSO/E REXX defines a format for compiled REXX execs so that TSO/E REXX can distinguish between compiled and interpreted execs. TSO/E REXX also provides a defined interface for an installation to install a REXX compiler runtime processor.

Compiled execs are executed by a compiler runtime processor. To initiate run-time processing of a compiled REXX exec, TSO/E uses a compiler programming table to identify the run-time processor and up to four interface routines. You can modify the compiler programming table to identify routines for a compiler runtime processor, if a compiler runtime processor is installed. Each of the four compiler interface routines are optional and, if installed, can provide special processing for the initialization and termination of the compiler runtime processor, loading of compiled REXX execs, and accessing REXX variables.

How REXX identifies a compiled exec

During REXX exec processing, TSO/E REXX determines whether an exec iscompiled or interpreted. TSO/E REXX will recognize an exec as compiled if the exec meets the following three criteria:
- The length of the first record must be at least 20 bytes
- The string 'EXECPROC' is in columns 5–12 of first record
- The first non-blank in columns 1–4 of the first record is not a comment delimiter

If an exec meets these criteria, TSO/E REXX determines the name of a compiler runtime processor from columns 13–20 of the first record.

You might find that some CLISTs and interpreted execs meet these criteria and are, therefore, incorrectly executed as compiled execs. There are several ways to correct this problem, including:
- Shift everything in the first record one column to the right. This leaves the string 'EXECPROC' in the first record, but not in the expected position (columns 5–12) for a compiled exec.
- Add a comment as the first record of the REXX exec or CLIST. The record that contains 'EXECPROC' remains intact as the second record.

  Note: Interpreted execs that are stored in data sets allocated to SYSPROC must contain the REXX exec identifier in the first record. For more information on the REXX exec identifier, see *z/OS TSO/E REXX Reference*.

The compiler programming table

The compiler programming table is a control block that TSO/E REXX uses to obtain information about a compiler runtime processor, including the names of up to four optional compiler interface routines. When TSO/E REXX initializes the first language processor environment in the address space, TSO/E REXX loads the IRXCMPTM module as the compiler programming table. After the compiler programming table is loaded, it is used for all compiled executions in the current and any subsequent language processor environments.

The IRXCMPTM module that TSO/E provides in LINKLIB contains entries that define the routines associated with the IBM REXX Alternate Library compiler runtime processor. From release z/OS V1R9, the IBM REXX Alternate Library is included in the base z/OS product. (The routine names used by the IBM REXX Alternate Library are the same names as used by the IBM REXX Library compiler runtime processor, therefore this version of IRXCMPTM will also work with the IBM REXX Library.)

TSO/E provides source for a sample compiler programming table in SYS1.SAMPLIB member IRXREXX4. IRXREXX4 is not intended to be used as shipped in SYS1.SAMPLIB. Refer to the installation documentation of your compiler for the requirements for the IRXCMPTM module. If you wish to install another REXX compiler, you can create your own compiler programming table using IRXREXX4 as a model. After you create the source for the compiler programming table, assemble and link-edit the table as a load module IRXCMPTM. You can place IRXCMPTM in any library that is accessible by the MVS LOAD system service.

TSO/E provides a mapping macro, IRXCMPTB, for the compiler programming table in SYS1.MACLIB. Table 88 and Table 89 present the format of the compiler programming table.

Table 88. compiler programming table header information

Offset (decimal)	Number of bytes	Field name	Description
0	4	FIRST	Address of the first entry
4	4	TOTAL	Total number of entries
8	4	USED	Number of entries used
12	4	LENGTH	Length of each entry
16	8	—	Reserved
24	8	COMPGMTB_FFFF	X'FFFFFFFFFFFFFFFF'

Table 89. compiler programming table entry information

Offset (decimal)	Number of bytes	Field name	Description
0	8	RTPROC	Name of the compiler runtime processor
8	8	COMPINIT	Name of the compiler interface initialization routine
16	8	COMPTERM	Name of the compiler interface termination routine

Table 89. compiler programming table entry information (continued)

Offset (decimal)	Number of bytes	Field name	Description
24	8	COMPLOAD	Name of the compiler interface load routine
32	8	COMPVAR	Name of the compiler interface variable handling routine
40	16	STORAGE	Four words of storage that can be used by a REXX compiler runtime processor. For example, a REXX compiler runtime processor might use these storage words as anchors for its control block structure.

Figure 95 shows the sample compiler programming table shipped in SYS1.SAMPLIB member IRXREXX4. XXXRTPRC is the name of the compiler runtime processor. XXXRTXLD is the name of the compiler interface load routine, and XXXRTXVH is the name of compiler interface variable handling routine. XXXRTXIN is the name of the compiler interface initialization routine, and XXXRTXTR is the compiler interface termination routine. If any of these routines are not needed by the compiler being installed, the names can be left blank to tell TSO/E REXX not to invoke these interface points for the given compiler.

```
IRXCMPTM CSECT ,
IRXCMPTM AMODE 31
IRXCMPTM RMODE ANY
IRXCMPTB_HEADER       DS    0CL32
IRXCMPTB_FIRST        DC    AL4(FIRST_ENTRY)
IRXCMPTB_TOTAL        DC    F'1'
IRXCMPTB_USED         DC    F'1'
IRXCMPTB_LENGTH       DC    F'56'
                      DC    X'0000000000000000'
IRXCMPTB_FFFF         DC    X'FFFFFFFFFFFFFFFF'
FIRST_ENTRY           DS    0CL56
FIRST_ENTRY_RTPROC    DC    C'XXXRTPRC'
FIRST_ENTRY_COMPINIT  DC    C'XXXRTXIN'
FIRST_ENTRY_COMPTERM  DC    C'XXXRTXTR'
FIRST_ENTRY_COMPLOAD  DC    C'XXXRTXLD'
FIRST_ENTRY_COMPVAR   DC    C'XXXRTXVH'
FIRST_ENTRY_STORAGE   DC    4F'0'
         END IRXCMPTM
```

Figure 95. Sample compiler programming table

The compiler runtime processor

When TSO/E REXX encounters a compiled REXX exec, TSO/E REXX passescontrol to the appropriate compiler runtime processor to run the exec. Prior to the first invocation of a compiled REXX exec in the first language processor environment in the address space, TSO/REXX loads the appropriate compiler runtime processor, saves the location of thecompiler runtime processor, then invokes the compiler runtime processor. On subsequent invocations of compiled REXX execs, and in subsequent language processor environments, TSO/E REXX uses the saved location of the loaded compiler runtime processor to pass control to the compiler runtime processor.

A compiler runtime processor receives control in the same recovery environmentas the REXX interpreter; TSO/E REXX establishes ESTAE recovery and attention handling routines based on the characteristics of the language processor environment. The compiler runtime processor must issue all messages relating to language processing. For more information on REXX recovery and attention processing, see *z/OS TSO/E REXX Reference*.

When the compiler runtime processor receives control, it must pass control to the exec initialization routine (EXECINIT) and exec termination routine (EXECTERM) at the appropriate times. The programming routine IRXRTE must be used to pass control to these routines.

Table 90 describes the results required from a compiler runtime processor.The results vary according to how the compiled exec was invoked under the compiler runtime processor.

Table 90. compiler runtime processor expected results

Invocation Method (compiled exec)	**Returned results (compiled exec)**			
	EXIT/RETURN Without Expression	**EXIT/RETURN With Expression**	**Language Error**	**Processing Error**
Subroutine	Set return code to 0. The compiler runtime processor must not obtain or complete an EVALBLOK.	Set return code to 0. The compiler runtime processor must use the GETEVAL function of IRXRLT to obtain an EVALBLOK. The compiler runtime processor must then use the results from the execution of the compiled exec to complete the EVALBLOK.	Set return code to 200*nn*, where *nn* is greater than or equal to 1 and less than or equal to 99. The compiler runtime processor must not obtain or complete an EVALBLOK.	Set return code to 20, 100, or 104; set abend and reason codes as appropriate. The compiler runtime processor must not obtain or complete an EVALBLOK.
Function	For a RETURN without expression, set the return code to 20045. Return code 20045 is a special case of return code 200nn. For an EXIT without expression, set the return code to 0.	Set return code to 0. The compiler runtime processor must use the GETEVAL function of IRXRLT to obtain an EVALBLOK. The compiler runtime processor must then use the results from the execution of the compiled exec to complete the EVALBLOK.	Set return code to 200*nn*, where *nn* is greater than or equal to 1 and less than or equal to 99. The compiler runtime processor must not obtain or complete an EVALBLOK.	Set return code to 20, 100, or 104; set abend and reason codes as appropriate. The compiler runtime processor must not obtain or complete an EVALBLOK.
Command	Set return code to 0. The compiler runtime processor must use the GETEVAL function of IRXRLT to obtain an EVALBLOK. The compiler runtime processor must then complete the EVALBLOK with a result of 0.	Set return code to 0. The compiler runtime processor must represent the results from the compiled exec execution as a number in string format. If the result string will fit in a fullword, the compiler runtime processor must use the GETEVAL function of IRXRLT to obtain an EVALBLOK. The compiler runtime processor must then complete the EVALBLOK with the result string. If the result string will not fit in a fullword, then the compiler runtime processor must set the return code to 20026 and must not obtain or modify an EVALBLOK.	Set return code to 200*nn*, where *nn* is greater than or equal to 1 and less than or equal to 99. The compiler runtime processor must not obtain or complete an EVALBLOK.	Set return code to 20, 100, or 104; set abend and reason codes as appropriate. The compiler runtime processor must not obtain or complete an EVALBLOK.

Entry specifications

The contents of the registers on entry to the compiler runtime processor are:

Register 0
Address of an environment block

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Entry point address

Parameters for the compiler runtime processor

In register 1, TSO/E REXX passes the address of a parameterlist, which consists of a list of addresses. Each address in the parameter list points to a parameter. TSO/E REXX passes all parameters on the call. The high-order bit of the last address in the parameter list is set to 1. Table 91 lists the parameters for the compiler runtime processor.

Table 91. Parameters for a compiler runtime processor

Parameter	Number of bytes	Description
Parameter 1	4	EXECBLK address. On entry to the compiler runtime processor, this parameter contains the address of the REXX exec block (EXECBLK) that is used by IRXLOAD. The exec block is a control block that describes the exec to be loaded. For more information on the exec block parameter for IRXLOAD, see *z/OS TSO/E REXX Reference*.
Parameter 2	4	Exec arguments. On entry to the compiler runtime processor, this parameter contains the address of a series of address/length pairs, which describe the arguments for the exec. A double word of X'FFFFFFFFFFFFFFFF' delineates the end of the pairs. For more information on REXX exec arguments, see *z/OS TSO/E REXX Reference*.
Parameter 3	4	A fullword of flag bits. For more information on flag bits, see the IRXEXEC parameters in *z/OS TSO/E REXX Reference*.
Parameter 4	4	In-storage control block address. The in-storage control block contains a series of address/length pairs, which REXX uses to describe the structure of a loaded exec in storage. The in-storage control block is initialized by IRXLOAD or the compiler interface load routine (if installed) before a compiler runtime processor receives control. For more information on the in-storage control block, see *z/OS TSO/E REXX Reference*.

Table 91. Parameters for a compiler runtime processor (continued)

Parameter	Number of bytes	Description
Parameter 5	4	CPPL address. Specifies the address of the command processor parameter list (CPPL). The CPPL contains addresses of control blocks that TSO/E uses in various programming services. For more information on CPPL's, see *z/OS TSO/E Programming Services*.
Parameter 6	4	Address of a user field. When a program calls IRXEXEC to invoke a compiled REXX exec, the program can pass the address of a user field. IRXEXEC passes the user field address to the compiler runtime processor in this parameter. For more information on the user field, see *z/OS TSO/E REXX Reference*.
Parameter 7	4	Environment block address. On entry, this parameter contains the address of the REXX environment block with which the compiler programming table is associated. This parameter is identical to the address in register 0. For more information on the REXX environment block, see *z/OS TSO/E REXX Reference*.
Parameter 8	4	compiler runtime processor entry address. Specifies the address of the entry in the compiler programming table for the compiler runtime processor.
Parameter 9	4	compiler runtime processor return code. On exit, the compiler runtime processor must set this parameter to a return code that indicates the completion status of the compiler runtime processor. Table 92 lists the return codes for the compiler runtime processor.
Parameter 10	4	compiler runtime processor abend and reason codes. On exit, if a return code of 100 or 104 is provided, the compiler runtime processor must set this parameter to the appropriate abend and reason codes that provide specific information about processing that was not successful. The abend and reason codes are the same as those returned by IRXEXEC. For more information on abend and reason codes for IRXEXEC, see *z/OS TSO/E REXX Reference*.

Return specifications

On return from the compiler runtime processor, the contents of registers 0–14 must be the same as on entry.

Return codes

Table 92 lists the return codes issued by the compiler runtime processor.

Table 92. Return codes from a REXX compiler runtime processor

Return code (decimal)	Description
0	Processing was successful. Table 90 on page 532 shows the expected results from the compiler runtime processor.
20	Processing was not successful. The compiler runtime processor issued an error message that describes the error.
100	Processing was not successful. A system abend occurred during the execution of the compiler runtime processor. Parameter 10 must contain the abend code and the reason code describing the error.

Table 92. Return codes from a REXX compiler runtime processor (continued)

Return code (decimal)	Description
104	Processing was not successful. A user abend occurred during the execution of the compiler runtime processor. Parameter 10 must contain the abend code and the reason code describing the error.
20001 - 20099	Processing was successful. However, the compiler runtime processor detected a syntax error in the compiled exec. The return code value is 20000 plus the value of the REXX error number. REXX error numbers are described in *z/OS TSO/E REXX Reference*.

Programming considerations

The compiler runtime processor must follow standard linkage conventions.It must save the registers on entry and restore the registers when it returns. The compiler runtime processor must be reentrant.

Environment

The attributes for the compiler runtime processor are:
- State: Problem Program
- Key: 8
- AMODE(31), RMODE(ANY)
- ASC mode: Primary
- Task Mode
- Reentrant

compiler interface routines

During various stages of processing a compiled REXX exec, TSO/E REXXinvokes a compiler interface routine, if installed, to perform special processing. The compiler runtime processor is not required to use the compiler interface routines. However, you must install those compiler interface routines that the compiler runtime processor requires. You indicate to TSO/E REXX that a compiler interface routine is not required by specifying a module name of eight blanks in the appropriate field of the compiler programming table entry. The four compiler interface routines are:

- compiler interface initialization routine — Initializes a compiler runtime processor
- compiler interface termination routine — Terminates a compiler runtime processor
- compiler interface load routine — Performs specialized processing to service a request to load or free a compiled exec
- compiler interface variable handling routine — Performs specialized processing to service a request to access REXX variables

compiler interface initialization routine

This routine, if installed, receives control to initialize a compiler runtime processor before the compiler runtime processor is invoked for the first time. TSO/E REXX invokes a compiler interface initialization routine once for each compiler runtime processor that runs in a REXX language processor environment.

Entry specifications

The contents of the registers on entry to the compiler interface initialization routine are:

Register 0
: Address of an environment block

Register 1
: Address of the parameter list

Registers 2–12
: Unpredictable

Register 13
: Address of a register save area

Register 14
: Return address

Register 15
: Entry point address

Parameter list for the compiler interface initialization routine

In register 1, TSO/E REXX passes the address of a parameter list,which consists of a list of addresses. Each address in the parameter list points to a parameter. TSO/E REXX passes all parameters on the call. The high-order bit of the last address in the parameter list is set to 1. Table 93 lists the parameters for the compiler interface initialization routine.

Table 93. Parameter list for the compiler interface initialization routine

Parameter	Number of bytes	Description
Parameter 1	4	Environment block address. On entry, this parameter contains the address of the REXX environment block with which the compiler programming table is associated. This parameter is identical to the address in register 0. For more information on the REXX environment block, see *z/OS TSO/E REXX Reference*.
Parameter 2	4	compiler runtime processor entry address. Specifies the address of the entry in the compiler programming table for the compiler runtime processor.
Parameter 3	4	compiler interface initialization routine return code. On exit, the compiler interface initialization routine must set this parameter to a return code that indicates the completion status of the compiler interface initialization routine. Table 94 on page 537 lists the return codes for the compiler interface initialization routine.

Return specifications

On return from the compiler interface initialization routine, the contents of registers 0–14 must be the same as on entry.

Return codes

Table 94 on page 537 lists the return codes issued by the compiler interface initialization routine.

Table 94. Return codes from the compiler interface initialization routine

Return code (decimal)	Description
0	Processing was successful. TSO/E REXX can now pass control to the compiler runtime processor.
20	Processing was not successful. TSO/E REXX will not give control to the associated compiler runtime processor. TSO/E REXX will not execute any compiled REXX exec that uses the associated compiler runtime processor.

Programming considerations

The compiler interface initialization routine must follow standard linkage conventions.It must save the registers on entry and restore the registers when it returns. The compiler interface initialization routine must be reentrant.

Environment

The attributes for the compiler interface initialization routine are:
- State: Problem Program
- Key: 8
- AMODE(31), RMODE(ANY)
- ASC mode: Primary
- Task Mode
- Reentrant

Compiler interface termination routine

This routine, if installed, receives control at the termination of a REXX language processor environment.

Entry specifications

The contents of the registers on entry to the compiler interface termination routine are:

Register 0
Address of an environment block

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Entry point address

Parameter list for the compiler interface termination routine

In register 1, TSO/E REXX passes the address of a parameter list, whichconsists of a list of addresses. Each address in the parameter list points to a parameter. TSO/E

REXX passes all parameters on the call. The high-order bit of the last address in the parameter list is set to 1. Table 95 lists the parameters for the compiler interface termination routine.

Table 95. Parameter list for the compiler interface termination routine

Parameter	Number of bytes	Description
Parameter 1	4	Environment block address. On entry, this parameter contains the address of the REXX environment block with which the compiler programming table associated. This parameter is identical to the address in register 0. For more information on the REXX environment block, see *z/OS TSO/E REXX Reference*.
Parameter 2	4	compiler runtime processor entry address. Specifies the address of the entry in the compiler programming table for the compiler runtime processor.
Parameter 3	4	Compiler interface termination routine return code. This parameter is reserved for future use. TSO/E REXX initializes this parameter to zero and does not inspect the parameter on return from the compiler interface termination routine.

Return specifications

On return from the compiler interface termination routine, the contents of registers 0–14 must be the same as on entry.

Return codes

The return code parameter in the compiler interface termination routine is reserved for future use.The compiler interface termination routine must not modify the return code parameter; TSO/E REXX does not inspect the return code parameter.

Programming considerations

The compiler interface termination routine must follow standard linkage conventions.It must save the registers on entry and restore the registers when it returns. The compiler interface termination routine must be reentrant.

Environment

The attributes for the compiler interface termination routine are:
- State: Problem Program
- Key: 8
- AMODE(31), RMODE(ANY)
- ASC mode: Primary
- Task Mode
- Reentrant

Compiler interface load routine

IRXLOAD will pass control to the compiler interface load routine in either of two cases:

- After the REXX language processor reads a compiled REXX exec into storage.
- When the REXX language processor makes a request to free the in-storage control block that was created by an earlier request to the compiler interface load routine.

Note: This section discusses the interaction between the compiler interface load routine and the IBM-supplied IRXLOAD routine.

For compiled execs, IRXLOAD will call the compiler interface load routine, if installed, before IRXLOAD builds the in-storage control block and after IRXLOAD has obtained all information required by the compiler interface load routine.

One of the inputs (parameter 5) to the compiler interface load routine is a group of blocks containing the compiled REXX exec. The compiler interface load routine must create and initialize an in-storage control block from the group of blocks, preferably above 16 MB in virtual storage. For more information about the in-storage control block, see *z/OS TSO/E REXX Reference*.

Entry specifications

The contents of the registers on entry to the compiler interface load routine are:

Register 0
Address of an environment block

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Entry point address

Parameter list for the compiler interface load routine

In register 1, the calling program (IRXLOAD) will pass the address of aparameter list, which consists of a list of addresses. Each address in the parameter list points to a parameter. IRXLOAD will pass all parameters on the call. The high-order bit of the last address in the parameter list is set to 1. Table 96 on page 540 lists the parameters for the compiler interface load routine.

Table 96. Parameter list for the compiler interface load routine

Parameter	Number of bytes	Description
Parameter 1	8	Function requested. On entry, this parameter contains the function requested of the compiler interface load routine The function specification must be in uppercase, left-justified, and padded on the right with blanks. Acceptable values are: **"LOAD "** Specifies that the compiler interface load routine is to load an exec into storage. **"FREE "** Specifies that the compiler interface load routine is to free the exec represented by the in-storage control block specified in parameter 8. For more information on the LOAD and FREE functions, see the descriptions for the LOAD and FREE functions of IRXLOAD in *z/OS TSO/E REXX Reference*.
Parameter 2	4	EXECBLK address. On entry to the compiler interface load routine, this parameter contains the address of the REXX exec block (EXECBLK) that is used by IRXLOAD. The exec block is a control block that describes the exec to be loaded. For more information on the exec block parameter for IRXLOAD, see *z/OS TSO/E REXX Reference*.
Parameter 3	4	Record format. On entry, this parameter specifies the format of records in the blocks passed to this routine in parameter 5. Possible values for this parameter are 'F ' for fixed-length records and 'V ' for variable-length records. Variable-length records will not span across blocks.
Parameter 4	4	Record length. On entry, this parameter specifies the length of each record for fixed-length records, or the maximum record length for variable-length records. Each variable-length record contains a record descriptor word (RDW). The first two bytes of the RDW indicate the actual length of the record, including the RDW.
Parameter 5	4	Address of a vector of address/length pairs. Each address/length pair contains the address and length of a block of data that contains the statements of the exec. A double word of X'FFFFFFFFFFFFFFFF' delineates the end of the pairs.
Parameter 6	4	Environment block address. On entry, this parameter contains the address of the REXX environment block with which the compiler programming table is associated. This parameter is identical to the address in register 0. For more information on the REXX environment block, see *z/OS TSO/E REXX Reference*.
Parameter 7	4	compiler runtime processor entry address. Specifies the address of the entry in the compiler programming table for the compiler runtime processor.

Table 96. Parameter list for the compiler interface load routine (continued)

Parameter	Number of bytes	Description
Parameter 8	4	In-storage control block address. The in-storage control block contains a series of address/length pairs, which REXX uses to describe the structure of a loaded exec in storage. For more information on the in-storage control block, see *z/OS TSO/E REXX Reference*. When IRXLOAD invokes the compiler interface load routine to load a compiled exec, the compiler interface load routine should create an in-storage control block and place the control block address in this parameter. IRXLOAD will consider this parameter to be valid only when the return code from the compiler interface load routine is zero. When IRXLOAD invokes the compiler interface load routine to free storage for the REXX exec, this parameter contains the address of the in-storage control block that the compiler interface load routine previously created, and is to free. For complete details on in-storage control blocks, see *z/OS TSO/E REXX Reference*.
Parameter 9	4	Compiler interface load routine return code. On exit, the compiler interface load routine must set this parameter to a return code that indicates the completion status of the compiler interface load routine. Table 97 lists the return codes issued by the compiler interface load routine.

Return specifications

On return from the compiler interface load routine, the contents of registers 0–14 must be the same as on entry.

Return codes

Table 97 lists the return codes issued by the compiler interface load routine.

Table 97. Return codes from the compiler interface load routine

Return code (decimal)	Description
0	Processing was successful. If the requested function was LOAD, parameter 8 contains the address of the created in-storage control block. If the requested function was FREE, the in-storage control block specified in parameter 8 has been freed.
4	Processing was successful. However, the compiler interface load routine did not create an in-storage control block. IRXLOAD will create an in-storage control block.
20	Processing was not successful. A severe error has occurred. The compiler interface load routine should issue a message to accompany this return code. IRXLOAD will propagate a return code of 20 to the caller of IRXLOAD.

Programming considerations

The compiler interface load routine must follow standard linkage conventions.It must save the registers on entry and restore the registers when it returns. The compiler interface load routine must be reentrant.

Environment

The attributes for the compiler interface load routine are:
- State: Problem Program
- Key: 8
- AMODE(31), RMODE(ANY)
- ASC mode: Primary
- Task Mode
- Reentrant

Compiler interface variable handling routine

The compiler interface variable handling routine, if installed, receives control whenever an external routine or host command requests access to REXX variables using IKJCT441 or IRXEXCOM.

Entry specifications

The contents of the registers on entry to the compiler interface variable handling routine are:

Register 0
Address of an environment block

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Entry point address

Parameter list for the compiler interface variable handling routine

In register 1, the calling program passes the address of aparameter list, which consists of a list of addresses. Each address in the parameter list points to a parameter. The high-order bit of the last address in the parameter list is set to 1. Table 98 on page 543 lists the parameters for the compiler interface variable handling routine.

Table 98. Parameter list for the compiler interface variable handling routine

Parameter	Number of bytes	Description
Parameter 1	1	Variable handling function request. On entry to the compiler interface variable handling routine, this parameter contains a one-character field corresponding to the shared variable request code (SHVCODE) used by IRXEXCOM. For more information on shared variable request codes, see *z/OS TSO/E REXX Reference*. In addition, this routine must support the function 'n', Fetch Next With Mask. TSO/E uses the Fetch Next With Mask function to search through all variables known to the language processor. These variables include stem variables that have been assigned a value. The output from this function is expected to be the next variable that begins with the specified mask.
Parameter 2	4	The address of the variable name to be manipulated. This is an input parameter for the following functions: **Function** **SHVCODE** **Set Variable** 'S','s' **Fetch Variable** 'F','f' **Drop Variable** 'D','d' **Fetch Private** 'P' For the Fetch Next ('N') and Fetch Next With Mask ('n') functions, this parameter must be set on output to the address of the next variable name.
Parameter 3	4	Length of variable name. Specifies the length of the string pointed to by the address in parameter 2.
Parameter 4	4	Address of the value for the variable. This is an input parameter for the Set Variable function ('S','s') and an output parameter for the following functions: **Function** **SHVCODE** **Fetch Variable** 'F','f' **Fetch Next With Mask** 'n' **Fetch Private** 'P' **Fetch Next** 'N' This parameter is not used for the Drop Variable ('D','d') function.
Parameter 5	4	Length of the value for the variable. Specifies the length of the value pointed to by the address in parameter 4.

Table 98. Parameter list for the compiler interface variable handling routine (continued)

Parameter	Number of bytes	Description
Parameter 6	4	Work block extension address. On entry, this parameter contains the address of the work block extension. The work block extension contains the WORKEXT_RTPROC field, which can be used by the compiler runtime processor as an anchor for resources that are specific to a particular compiled exec.
Parameter 7	4	compiler runtime processor entry address. Specifies the address of the entry in the compiler programming table for the compiler runtime processor.
Parameter 8	4	Environment block address. On entry, this parameter contains the address of the REXX environment block with which the compiler programming table is associated. This parameter is identical to the address in register 0. For more information on the REXX environment block, see *z/OS TSO/E REXX Reference*.
Parameter 9	1	Shared variable function return code (SHVRET). On output, the compiler interface variable handling routine must set this parameter to the appropriate value for the SHVRET field. The values returned in this parameter for the Fetch Next With Mask function must be identical to those returned for the Fetch Next function. For a list of appropriate values for the SHVRET field, see *z/OS TSO/E REXX Reference*.
Parameter 10	4	Compiler interface variable handling routine return code. On exit, the compiler interface variable handling routine must set this parameter to a return code that indicates the completion status of the compiler interface variable handling routine. Table 99 on page 545 lists the return codes for the compiler interface variable handling routine.
Parameter 11	4	Fetch next mask. This parameter is optional, and is used only with the Fetch Next With Mask function ('n'). If this parameter is provided, it specifies an address of a mask to be used to search for the next variable or stem. The mask can be a character string that meets the naming conventions for simple variables or variable stems. The mask cannot identify a compound variable. The compiler interface variable handling routine must return a variable whose name begins with the mask provided. A parameter value of zero indicates that no mask is provided.
Parameter 12	4	Fetch next mask length. This parameter is optional, and may be used only in conjunction with parameter 11. If the parameter is specified, the value represents the length of the mask provided in parameter 11. This parameter should be ignored if the value in parameter 11 is zero.

Return specifications

On return from the compiler interface variable handling routine, the contents of registers 0–14 must be the same as on entry.

Return codes

Table 99 lists the return codes issued by the compiler interface variable handling routine.

Table 99. Return codes from the compiler interface variable handling routine

Return code (decimal)	Description
0	Processing was successful.
4	Processing was not successful. Insufficient storage was available.
8	Processing was not successful. The name that was passed in parameter 2, or created by a symbolic substitution on parameter 2, is too long.
12	Processing was not successful. The name that was passed in parameter 2, or created by a symbolic substitution on parameter 2, is incorrect because it begins with a character that is not valid.
20	Processing was not successful.

Programming considerations

The compiler interface variable handling routine must follow standard linkage conventions.It must save the registers on entry and restore the registers when it returns. The compiler interface variable handling routine must be reentrant.

Environment

The attributes for the compiler interface variable handling routine are:
- State: Problem Program
- Key: 8
- AMODE(31), RMODE(ANY)
- ASC mode: Primary
- Task Mode
- Reentrant

Chapter 44. Programming routines for a REXX compiler runtime processor

This chapter discusses the programming routines that TSO/E Release 3.1 introduced to support a compiler runtime processor. These programming routines include: These routines are intended for use only by a REXX compiler runtime processor.
- The external routine search routine
- The host command search routine
- The exit routing routine

Overview of programming routines for a REXX compiler runtime processor

TSO/E provides programming routines that support a compiler runtime processor. These routines enable a compiler runtime processor to search for and run an external routine, host command, or exit routine.

Environment for the programming routines

The programming routines must run in an environment with the following characteristics:
- State: Problem Program
- Key: 8
- AMODE(31), RMODE(ANY)
- ASC mode: Primary
- Task mode

External routine search routine (IRXERS)

IRXERS is a programming routine that searches for and runs an external routine. IRXERS allows a compiler runtime processor to pass control to an external routine by a direct interface. A compiler runtime processor that uses IRXERS leaves the implementation of the external routine search and invocation to TSO/E REXX. For more information on the search order for REXX external routines, refer to *z/OS TSO/E REXX Reference*.

Entry specifications

The contents of the registers on entry to IRXERS are:

Register 0
Address of an environment block (optional)

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Entry point address

Parameters for IRXERS

You can pass the address of an environment block in register 0. In register 1, the compiler runtime processor must pass the address of a parameter list, which consists of a list of addresses. Each address in the parameter list points to a parameter.

The first five parameters are required. The addresses that point to parameter 6 and parameter 7 are optional. If IRXERS does not find the high-order bit set on in either the address for parameter 5, or in the addresses for parameters 6 or 7, which are optional parameters, IRXERS does not invoke the specified routine and returns with a return code of 32 in register 15. See Table 101 on page 550 for more information on return codes. The high-order bit of the last address in the parameter list must be set to 1. Table 100 lists the parameters for the external routine search routine.

Table 100. Parameters for the external routine search routine

Parameter	Number of bytes	Description
Parameter 1	8	Function requested. On entry to IRXERS, this parameter contains the function requested of the external routine search routine. The function specification must be in uppercase, left-justified, and padded on the right with blanks. Acceptable values are: **"EXTSUB "** Specifies that the external routine that is being requested is a subroutine. The subroutine is not required to return an EVALBLOK. For a successfully run subroutine that does not return an EVALBLOK, the EVALBLOK address is set to 0 and the return code is set to 0. **"EXTBRSUB"** Specifies that the external routine that is being requested is a subroutine and will be given control through a branch instruction. The external routine will be invoked using standard register linkage conventions as described for external function parameter lists in *z/OS TSO/E REXX Reference*. **"EXTFCT "** Specifies that the external routine that is being requested is a function. The function is required to return an EVALBLOK. For a successfully run function that does not return an EVALBLOK, the EVALBLOK address is set to 0 and the return code is set to 4. **"EXTBRFCT"** Specifies that the external routine that is being requested is a function and will be given control through a branch instruction. The external routine will be invoked using standard register linkage conventions as described for external function parameter lists in *z/OS TSO/E REXX Reference*.

Table 100. Parameters for the external routine search routine (continued)

Parameter	Number of bytes	Description
Parameter 2	4	Address of the external routine name. For the "EXTFCT " and "EXTSUB " functions, this parameter specifies the address of the external routine name for the requested external routine. The name must not include the opening left parenthesis that identifies the routine as a function, if that is the type of routine being invoked. For the "EXTBRFCT" and "EXTBRSUB" functions, this parameter specifies the address of the external routine that is to be given control. IRXERS will branch to this address after building the parameter list for the specified routine.
Parameter 3	4	Length of the external routine name. Specifies the length of the external routine name pointed to by parameter 2. IRXERS ignores this parameter if "EXTBRFCT" or "EXTBRSUB" is specified in parameter 1.
Parameter 4	4	Address of the arguments for the external routine. Specifies the address of a set of address/length pairs, which hold the arguments for the external routine. These arguments must be in the format expected by an external routine. See the mapping macro IRXARGTB in *z/OS TSO/E REXX Reference*, for a description of the argument list format.
Parameter 5	4	Address of an EVALBLOK. On return from IRXERS, this parameter contains the address of an EVALBLOK (if any) that IRXERS returned after an external routine successfully completed. An address of zero indicates that IRXERS did not receive an EVALBLOK.
Parameter 6	4	Address of a REXX environment block. Specifies the address of the REXX environment block under which the request is to be performed. If the compiler runtime processor supplies a non-zero parameter, IRXERS considers this parameter to be a valid environment block address. If this parameter is zero or omitted, IRXERS obtains the environment block address from register 0 as described in *z/OS TSO/E REXX Reference*. This parameter is optional.
Parameter 7	4	Return code. The return code parameter is optional. On return from IRXERS, this parameter (if supplied) contains the return code for IRXERS. Register 15 will contain the same value as this parameter, if this parameter is provided.

Return specifications

On return from the external routine search routine, the contents of the registers will be:

Registers 0–14
: Same as on entry

Register 15
: Return code

Return codes

Table 101 lists the return codes issued by the external routine search routine.

Table 101. Return codes from the external routine search routine

Return code (decimal)	Description
0	Processing was successful. IRXERS located the external routine, and the external routine returned control with a return code of 0 in register 15. If EXTFCT or EXTBRFCT was specified, the address of the EVALBLOK is available in parameter 5.
4	Processing was successful. IRXERS located the external routine, and the external routine returned control with a return code of 0 in register 15. However, EXTFCT or EXTBRFCT was specified, and no EVALBLOK was returned by the external routine.
8	Processing was successful. IRXERS located the external routine, and the external routine returned with a non-zero return code in register 15.
12	Processing was not successful. IRXERS attempted to create an EVALBLOK, but insufficient virtual storage was available.
16	Processing was not successful. IRXERS could not locate the specified routine.
20	Processing was not successful. An error message may accompany this return code.
28	Processing was not successful. IRXERS was unable to locate a language processor environment. Verify that you passed a valid environment block address.
32	Processing was not successful. The parameter list is not valid. The parameter list contains either too few or too many parameters, or the high-order bit of the last address in the list is not set to 1 to indicate the end of the parameter list.
100	Processing was not successful. IRXERS located and passed control to the external routine. However, an abend occurred in the external routine.

Host command search routine (IRXHST)

IRXHST is a programming routine that searches for and runs a host command. IRXHST allows a compiler runtime processor to pass control to a host command through a direct interface. A compiler runtime processor that uses IRXHST leaves the implementation of the host command search and invocation to TSO/E REXX. For more information on the search order for host commands, refer to *z/OS TSO/E REXX Reference*.

IRXHST also allows a compiler runtime processor to set and clear the ETMODE flag, based on the OPTIONS ETMODE or OPTIONS NOETMODE instructions. The ETMODE function of IRXHST sets the ETMODE flag, and the NOETMODE function of IRXHST clears the ETMODE flag. For more information about OPTIONS ETMODE and OPTIONS NOETMODE, see *z/OS TSO/E REXX Reference*.

Entry specifications

The contents of the registers on entry to IRXHST are:

Register 0
Address of an environment block (optional)

Register 1
Address of the parameter list passed by the caller

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Entry point address

Parameters for IRXHST

In register 1, the compiler runtime processor must pass the address of a parameter list, which consists of a list of addresses. Each address in the parameter list points to a parameter. The first six parameters are required. The addresses that point to parameter 7 and parameter 8 are optional. If IRXHST does not find the high-order bit set on in either the address for parameter 6, or in the addresses for parameters 7 or 8, which are optional parameters, IRXHST does not invoke the specified routine and returns with a return code of 32 in register 15. See Table 103 on page 553 for more information on return codes. The high-order bit of the last address in the parameter list must be set to 1. Table 102 lists the parameters for the host command search routine.

Table 102. Parameters for the host command search routine

Parameter	Number of bytes	Description
Parameter 1	8	Function requested. On entry to IRXHST, this parameter contains the function requested of the host command search routine. The function name must be in uppercase, left-justified, and padded on the right with blanks. Acceptable values are: **"HOSTCMD "** Specifies that IRXHST will search for and invoke a host command. **ETMODE** Specifies that IRXHST sets the ETMODE flag. **NOETMODE** Specifies that IRXHST clears the ETMODE flag.
Parameter 2	8	Host command environment name. Specifies the name of the host command environment that is in effect for the compiled REXX exec that is running. The name must be in uppercase, left-justified, and padded on the right with blanks. The host command environment name should correspond to an entry in the host command environment table. This parameter is only used for the HOSTCMD function. It is ignored for the ETMODE and NOETMODE functions.

Table 102. Parameters for the host command search routine (continued)

Parameter	Number of bytes	Description
Parameter 3	4	Address of host command string. Specifies the address of a string to be run by the host command environment. IRXHST passes the string as is to the host command environment routine that corresponds to the host command environment specified in parameter 2. Storage for the command buffer must be managed (allocated and freed) by the program that calls IRXHST. This parameter is only used for the HOSTCMD function. It is ignored for the ETMODE and NOETMODE functions.
Parameter 4	4	Host command string length. Specifies the length of the command string that is pointed to by the address in parameter 3. This parameter is only used for the HOSTCMD function. It is ignored for the ETMODE and NOETMODE functions.
Parameter 5	4	Command output buffer address. Specifies the address of an area to hold the result of the command. This result will be a character representation of the binary return code issued by the host command. It is recommended that this area be 20 bytes. If parameter 2 is not defined in the host command environment table, IRXHST returns the character representation of -3. The compiler runtime processor that calls IRXHST should properly set the REXX special variable RC. This parameter is only used for the HOSTCMD function. It is ignored for the ETMODE and NOETMODE functions.
Parameter 6	4	Output area length. Specifies the length of the output area that is pointed to by the address in parameter 5. This parameter is only used for the HOSTCMD function. It is ignored for the ETMODE and NOETMODE functions.
Parameter 7	4	Address of a REXX environment block. Specifies the address of the REXX environment block under which the request is to be performed. If the compiler runtime processor supplies a non-zero parameter, IRXHST considers this parameter to be a valid environment block address. If this parameter is zero or omitted, IRXHST obtains the environment block address from register 0 as described in *z/OS TSO/E REXX Reference*. This parameter is optional.
Parameter 8	4	Requested function return code. The return code parameter is optional. On return from IRXHST, this parameter (if supplied) contains the return code for IRXHST. See Table 103 on page 553 for information on return codes issued by the host command search routine. Register 15 will contain the same value as this parameter, if this parameter is provided.

Return specifications

On return from the host command search routine, the contents of the registers will be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes

Table 103 lists the return codes issued by the host command search routine.

Table 103. Return codes from the host command search routine

Return code (decimal)	Description
0	Processing was successful. For the HOSTCMD function, IRXHST located the host command and the host command returned a return code of 0 in register 15. For the ETMODE and NOETMODE functions, IRXHST set or cleared the ETMODE flag successfully.
20	Processing was not successful. For the HOSTCMD function, IRXHST could not locate the specified host command. IRXHST returns -3 in the command output buffer. The command string specified in parameters 3 and 4 may be incorrect, the requested command could not be located in the search order, or the host command environment routine is not defined in the host command environment table. This return code could also indicate a not valid function (parameter 1) passed to IRXHST. Valid functions are HOSTCMD, ETMODE and NOETMODE.
28	Processing was not successful. IRXHST could not locate a language processor environment. The command output area is not modified. Verify that you passed a valid environment block address.
32	Processing was not successful. The parameter list is not valid. The parameter list contains either too few or too many parameters, or the high-order bit of the last address in the list is not set to 1 to indicate the end of the parameter list. The command output buffer is not modified.
100	Processing was not successful. An abend occurred in the specified host command. IRXHST returns the abend code in the command output buffer. The compiler runtime processor should set the REXX special variable RC to this abend code.
nn	Processing was successful. The specified host command environment routine returned a non-zero return code. *nn* is the return code from the host command environment routine.

Exit routing routine (IRXRTE)

IRXRTE is a programming routine that locates and invokes a REXX exit. IRXRTE provides a way for a compiler runtime processor to invoke REXX exit routines. A compiler runtime processor that uses IRXRTE leaves the implementation of exit routing to TSO/E REXX. For information about REXX exit routines, see *z/OS TSO/E REXX Reference*.

Entry specifications

The contents of the registers on entry to IRXRTE are:

Register 0
Address of an environment block (optional)

Register 1
Address of the parameter list passed by the caller

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Entry point address

Parameters for IRXRTE

In register 1, the compiler runtime processor must pass the address of a parameter list, which consists of a list of addresses. Each address in the parameter list points to a parameter. The first three parameters are required. The addresses that point to parameter 4 and parameter 5 are optional. If IRXRTE does not find the high-order bit set on in either the address for parameter 3, or in the addresses for parameters 4 or 5, which are optional parameters, IRXRTE does not invoke the specified routine and returns with a return code of 32 in register 15. See Table 105 on page 555 for more information on return codes. The high-order bit of the last address in the parameter list must be set to 1. Table 104 lists the parameters for the exit routing routine.

Table 104. Parameters for the exit routing routine

Parameter	Number of bytes	Description
Parameter 1	8	Function requested. On entry to IRXRTE, this parameter contains the function requested of the exit routing routine. The function name must be in uppercase, left-justified, and padded on the right with blanks. Acceptable values are: **"EXECINIT"** Specifies that the EXECINIT exit routine is to be run. For more information on the EXECINIT exit, see *z/OS TSO/E REXX Reference*. **"EXECTERM"** Specifies that the EXECTERM exit routine is to be run. For more information on the EXECTERM exit, see *z/OS TSO/E REXX Reference*.
Parameter 2	8	Exit routine parameter list address. Specifies the address of the parameter list for the requested exit routine. If the exit does not require parameters, the address in this parameter must be set to zero. For a discussion of the parameters for the specified exit, see *z/OS TSO/E REXX Reference*.
Parameter 3	4	Exit routine return code. On return from IRXRTE, this parameter contains the return code value from the requested exit. This value only has meaning if the return code from IRXRTE is zero.

Table 104. Parameters for the exit routing routine (continued)

Parameter	Number of bytes	Description
Parameter 4	4	Address of a REXX environment block. Specifies the address of the REXX environment block under which the request is to be performed. If the compiler runtime processor supplies a non-zero parameter, IRXRTE considers this parameter to be a valid environment block address. If this parameter is zero or omitted, IRXRTE obtains the environment block address from register 0 as described in *z/OS TSO/E REXX Reference*. This parameter is optional.
Parameter 5	4	Return code. The return code parameter is optional. On return from IRXRTE, this parameter (if supplied) contains the return code for IRXRTE. Register 15 will contain the same value as this parameter, if this parameter is provided.

Return specifications

On return from the exit routing routine, the contents of the registers will be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes

Table 105 lists the return codes issued by the exit routing routine.

Table 105. Return codes from the exit routing routine

Return code (decimal)	Description
0	Processing was successful. IRXRTE located the exit, passed control to the exit, and the exit ran to completion. The return code from the exit is available in parameter 3.
4	Processing was not successful. The module name table for the current environment did not have an entry for the exit requested. Verify that the environment block address specified in parameter 4 is correct and the module name table contains the name of the exit you specified.
20	Processing was not successful. The error may have occurred because: • A compiled exec is not executing • The requested function is not supported.
28	Processing was not successful. A language processor environment could not be located. Verify that the environment block address specified in parameter 4 is correct.
32	Processing was not successful. The parameter list contained too few or too many parameters, or the high-order bit of the last parameter was not set to 1 to indicate the end of the parameter list.

Part 8. Session Manager

Before Session Manager can be used in an installation, you may have to:

- Modify a SYS1.PARMLIB member, either TSOKEY00 or IKJPRM00
- Modify or create logon procedures tailored for Session Manager

Chapter 45, “Setting up a Session Manager environment,” on page 559 presents the modifications necessary to support a Session Manager environment and discusses system considerations for using Session Manager.

Chapter 46, “Customizing Session Manager,” on page 563 describes the format of the IBM-supplied default environment module, ADFMDFLT, and its stream definitions. The chapter discusses modifying a default environment using a CLIST or the AMASPZAP service aid and presents ways of providing multiple default environments using different logon procedures for groups of users.

This chapter also describes the three Session Manager exits: initialization, stream monitoring, and termination. You can use these exits to:

- Monitor and intercept commands
- Retain a log of a user's TSO/E session
- Determine how long it takes a command to execute
- Determine at what time certain operations were performed

A functional description of the three exits and their specifications is followed by programming considerations, including installing the exits, their environment, and possible uses. “Possible uses” on page 579 provides information about changing the stream monitoring exit dynamically and shows an example initialization exit.

Chapter 45. Setting up a Session Manager environment

Before you can use the Session Manager at your installation, you may have to:
- Modify a SYS1.PARMLIB member, either TSOKEY00 or IKJPRM00
- Modify or create logon procedures tailored for Session Manager

This chapter presents the modifications necessary to support a Session Manager environment.

SYS1.PARMLIB changes

To support a Session Manager environment, you may have to modify the following SYS1.PARMLIB members (or any installation-specified alternate members) to use specific types of terminals and improve performance:
- For VTAM - TSOKEY00

Two defaults in member TSOKEY00 are:
- SCRSIZE - 480
- BUFRSIZE - 132

Note:

1.

Figure 96. Recommendations

2. Increase SCRSIZE to correspond to the type of terminals installed. For example, specify 3440 for a 3278 Model 4
3. Increase BUFRSIZE to 1000 or greater. Although the buffer size is not critical, a larger size should improve performance.

Logon procedure changes

When initialized, Session Manager loads the default environment module specified in the PARM field on the EXEC statement of the logon procedure. Session Manager loads a default environment module, (ADFMDFLT) if a module name is not specified in the PARM field. The default environment module contains the tables and data necessary for building a user's Session Manager screen layout, PFK definitions, and so on.

If you want to use a single default environment for all TSO/E users, you can modify the IBM-supplied default environment module to set up the installation-dependent environment. If a single environment is not satisfactory for all users, you can create multiple logon procedures, each one specifying a different default environment module.

To enable the use of Session Manager, you may have to create new logon procedures or modify the EXEC statement in existing procedures. The EXEC statement can have the following format:

```
//[stepname]  EXEC  PGM=ADFMDF03[,
// PARM= 'SM ( tmpname  , Y , default-module-name )[,] tso-cmd]'
               IKJEFT01   N   ADFMDFLT
```

Explanation:

- PGM=ADFMDF03 - attach the Session Manager initialization task, ADFMDF03, instead of the TMP. After initialization is complete, attach the TMP.
- *tmpname* - the name of the terminal monitor program (TMP)
- IKJEFT01 - the name of the TMP that TSO/E provides
- Y - the TMP is attached APF-authorized
- N - the TMP is attached non-APF-authorized
- *default-module-name* - the name of an installation-written default environment module
- ADFMDFLT - the name of the default environment module TSO/E provides
- *tso-cmd* - a TSO/E command with any associated operands and parameters

A sample logon procedure for Session Manager is:

```
//SMPROC  EXEC  PGM=ADFMDF03,DYNAMNBR=30,
//  PARM='SM(IKJEFT01,Y),EXEC ''SYS1.PRD.CLIST(SMLOGON)'''
//SYSPROC  DD   DSN=SYS1.TSO.CLIST,DISP=SHR
⋮
/*
```

Note: The data set SYS1.PRD.CLIST is an installation-defined CLIST data set.

Session Manager environment considerations

The following topics describe system considerations for using Session Manager.

Avoiding a system interlock condition

In a Session Manager environment with Session Manager active (not running in a full-screen environment, for example, ISPF not active) a system interlock occurs if a task running above the Session Manager task sets the Session Manager task nondispatchable and issues TGETs or TPUTs. For example, the SMF time limit installation exit, IEFUTL, should not issue TGETs or TPUTs because the Session Manager task is set nondispatchable prior to the exit getting control.

Converting Session Manager CLISTs

TSO/E installs the following Session Manager CLISTs into SYS1.ADFMAC1:

- ADFSPLT
- ADFSETUP
- ADFVSPLT

Recommendation: Copy the CLISTs into the installation-defined production CLIST data set. If your production CLIST has a RECFM of VB, run CLIST ICQSMC00. ICQSMC00, located in ICQ.ICQSAMP, converts the CLIST data set members shipped to you from a RECFM of FB to a RECFM of VB.

Deleting SYS1.SMLIB

SYS1.SMLIB is no longer required. Session Manager is link-edited into SYS1.LINKLIB.

Recommendation: Delete SYS1.SMLIB and remove all linklist library and similar references to it.

Chapter 46. Customizing Session Manager

The Session Manager module contains the tables and data needed for building the TSO/E user's Session Manager screen layout, PF key definitions, and other items. A copy of the assembler source of ADFMDFLT is in SYS1.SAMPLIB(ADFDFLTX). You can either modify it, reassemble it, and create a new load module for another default environment; or, if you choose, code your own default environment module.

If you want to use one default environment for all your TSO/E users, you can modify ADFMDFLT to refine the environment for your installation's particular requirements. If you find that one environment does not satisfy the requirements of all users, you can create multiple logon procedures, each one specifying a different installation-supplied default environment module.

This section describes the streams that comprise the default environment module, how to modify a default environment, and how to provide multiple default environments. For information about using the Session Manager streams, see *z/OS TSO/E User's Guide*.

Stream definitions

The parameters that define the streams are located at approximately X'1808' in ADFMDFLT. The label on the stream definitions is ADFIDSTR. The data at that location is a fullword initialized to X'00000009', indicating nine streams are defined for Session Manager. Table 106 shows the default stream definitions.

Table 106. Default stream definitions

Stream name	Stream size (in bytes)	Maximum lines	Lines per IDB	Stream type	Header length	Stream header	Stream flags
TSOIN	8192	300	5	1	32	*** THIS IS THE TSOIN STREAM ***	0000 0000
TSOUT	147456	4000	5	2	26	*** TSO SESSION OUTPUT ***	0000 0000
SMIN	8192	300	5	1	32	*** THIS IS THE SMIN STREAM ***	0000 0000
SMOUT	4096	150	5	2	33	*** THIS IS THE SMOUT STREAM ***	0000 0000
HEADER	1024	50	5	0	33	*** THIS IS THE HEADER STREAM ***	8000 0000
MESSAGE	1024	50	5	2	34	*** THIS IS THE MESSAGE STREAM ***	4000 0000
EXTRA1	32768	400	5	2	33	*** THIS IS THE EXTRA1 STREAM ***	0000 0000

Table 106. Default stream definitions (continued)

Stream name	Stream size (in bytes)	Maximum lines	Lines per IDB	Stream type	Header length	Stream header	Stream flags
EXTRA2	1024	100	5	0	33	*** THIS IS THE EXTRA2 STREAM ***	0000 0000
EXTRA3	1024	100	5	0	33	*** THIS IS THE EXTRA3 STREAM ***	0000 0000

Stream definition descriptions

Each stream is defined by eight fullwords that follow the stream number. The fullwords contain:
- Stream name pointer
- Stream size
- Maximum lines
- Lines per IDB
- Stream type
- Stream header length
- Stream header pointer
- Stream flags

Stream name pointer

This fullword points to an 8-byte field containing the name of the stream. The name is left-justified and padded with blanks.

Stream size

This fullword contains the length of the stream (in hexadecimal). It is the total number of bytes available in the stream for text. A stream wraps if the maximum number of bytes has been reached and the stream is defined as one that can wrap. The bit setting for the stream flags in byte eight determines the ability of a stream to wrap.

Stream management overhead can be determined by using the following formula:

$$\frac{\text{number of lines per index descriptor block}}{\text{maximum number of lines in the stream}} + 1$$

Index Descriptor Blocks (IDBs) are part of an indexing mechanism Session Manager uses for referencing stream data.

For example, you can determine the number of bytes of stream management overhead for the SMOUT stream as follows. This stream is defined as 4096 bytes long, 150 lines maximum, and 5 lines per IDB. Using the above formula results in the following:

$$\frac{150}{5} + 1 = 30 + 1 = 31 \text{ IDBs}$$

The 31 IDBs can actually index 155 lines of data. Using the QUERY.STREAMS command with the IBM-supplied defaults would show this.

Because IDBs are eight bytes long, the overhead is:

```
31 IDBs x 8 bytes per IDB = 248 bytes overhead
```

Thus, 248 bytes of stream management overhead are obtained in addition to the text portion size. The two physical storage requirements really define one logical stream consisting of an overhead area and a text area.

For a discussion of IDBs and performance considerations, see "Changing the number of lines per IDB" on page 567.

Maximum lines

This fullword contains the total number of lines, in hexadecimal, that can be entered into the stream. If the total number of lines in the stream exceeds the stream's value for the maximum number of lines, the stream wraps (overlays) the previous stream entries. Note that a stream wraps if the maximum number of lines is reached, even if the byte capacity of that stream is not exceeded.

Lines per IDB

This fullword contains the number (in hexadecimal) of logical lines for which an index descriptor block (IDB) is responsible. Each IDB is 8 bytes long.

Stream type

This fullword is initialized to one of the following values to define the type of stream:

0 Defines an EXTRA stream type. IBM supplies the HEADER, EXTRA2, and EXTRA3 streams for this type by default. Session Manager does not actively use the EXTRA2 and EXTRA3 streams. Optionally, the user can direct data to an EXTRA stream using Session Manager commands.

1 Defines an INPUT stream type. IBM supplies the TSOIN and SMIN streams for this type by default. Session Manager uses the INPUT streams for TSO/E command input (TSOIN) and Session Manager input (SMIN).

2 Defines an OUTPUT stream type. IBM supplies the TSOOUT, SMOUT, MESSAGE, and EXTRA1 streams for this type by default. Session Manager uses the OUTPUT streams for any output generated by:
- TSO/E commands (TSOOUT)
- Session Manager commands (SMOUT)
- SNAPSHOT commands (EXTRA1).

Session Manager does not actively use the MESSAGE stream. The user can direct output to the MESSAGE stream using Session Manager commands.

Header length

This fullword is initialized to the length of the stream header (in hexadecimal).

Stream header pointer

This fullword contains a pointer to the stream header. The stream header is a string of 26 to 34 characters that becomes the first record in the stream. When scrolling through a stream, the header identifies the name of the stream, making it easier to identify.

Stream flags

This fullword is defined as a flag field. Only two bits are defined. Bit 0 indicates if the stream wraps. Bit 1 indicates if the alarm sounds when data is entered into the stream. The remaining 30 bits are reserved.

Modifying a default environment

To modify a default environment, you can use:
- A CLIST
- The AMASPZAP service aid

Using a CLIST to modify a default environment

If you want to make several modifications to the screen layout and PF key definitions, you can supply the TSO/E EXEC command as a TMP parameter on the EXEC statement of the logon procedure. This command could execute an installation CLIST that issues Session Manager commands to modify the defaults. This CLIST could also execute another CLIST, the name of which is prefixed by the person's TSO/E user ID (using the &SYSUID built-in function) to allow the user to further modify the Session Manager environment.

While using a CLIST is the easiest way to modify the defaults, there are two important considerations:
- The only changes that you can make are those done by Session Manager commands. You cannot modify stream sizes using this method.
- If the user issues the Session Manager RESET command, then the default environment is set up without executing the CLIST again.

Using AMASPZAP to modify the defaults module

For simplicity, assume that the name of the IBM-supplied default environment module you are using is ADFMDFLT. If you want to modify some of the streams to make them larger or smaller, use AMASPZAP to modify ADFMDFLT. While you can modify the streams using this method, be sure you do not change the logical data structure of the module.

Simply altering ADFMDFLT limits the extent to which you can modify the Session Manager default environment:
- You cannot provide different environments for different users this way. Any change you make affects all TSO/E users whose logon procedures load ADFMDFLT.
- There is little you can change without changing the data structure of the module. The screen layout and PF key definitions would be difficult to modify.

The following topic describes the changes you can make to the stream definitions using the AMASPZAP service aid.

Changing stream definitions

When altering streams using the AMASPZAP service aid, keep the following information in mind:
- Session Manager has nine streams, indicated by the fullword at label ADFIDSTR in module ADFMDFLT. It is initialized to X'00000009'. Do not change this value. If it is changed, unpredictable results occur.
- The following stream names are required:
  - TSOIN
  - TSOOUT
  - SMIN
  - SMOUT
  - EXTRA1
  - EXTRA3

  - HEADER

  You can change the maximum number of lines, lines per IDB, stream size, and stream header, but you cannot change the stream names, stream type, or stream flags.
- The remaining two streams, MSG and EXTRA2, are not required. You may change any definitions for these two streams except their stream names.
- The stream headers must remain the same length as currently defined.

Changing stream size

Changing the stream size causes the particular stream to reside in more or less storage than the original. If the size is increased, more data is entered into a stream; if the size is decreased, less is entered. If you do not change the size of the default TSOIN stream, it will probably wrap. Increasing the size to hold all TSO/E commands and data a user enters during a session obviously has a disadvantage: more storage is required. However, decreasing the size causes the stream to wrap sooner, so some stream data is lost by overlaying or wrapping.

Changing the maximum number of lines

Changing the maximum number of lines in a stream also has an effect on the frequency of wrapping. If the maximum number of lines is changed to a high number, a stream could wrap because the number of bytes entered exceeded the stream capacity but wasn't close to the maximum number of lines. If the maximum number of lines is set quite low, the stream could wrap with a large amount of stream capacity unused.

Changing the number of lines per IDB

The number of lines per index descriptor block (IDB), as defined in the default, is set to five for all streams. You can decrease the number of IDBs created by increasing the number of lines per IDB. The net result is a decrease in the amount of overhead (less storage for IDBs), but an increase in search time when scrolling through a stream. Decreasing the number of lines per IDB increases the total number of IDBs and the storage required for them. But the efficiency of stream accessing improves when there are fewer lines per IDB.

For a discussion about calculating the number of IDBs, see “Stream size” on page 564.

Providing multiple default environments

If you want to provide multiple default environments for your installation's TSO/E users, you can provide a logon procedure that specifies a different default environment module for each environment you want.

IBM supplies a copy of the assembler source of ADFMDFLT in SYS1.SAMPLIB(ADFDFLTX). You can either modify it, reassemble it, and create a new load module for another default environment, or, if you choose, code your own default environment module.

Formatting a default environment module

The header data of the default environment is the module entry point. The module also includes three tables and, conditionally, three exit routines. For more information about the Session Manager exit routines, see “Writing Session Manager exits” on page 571.

The module attribute type must be refreshable and reentrant.

The default environment module contains four basic areas:
- Header area
- Stream definitions table
- Function definitions table
- Session Manager commands table

The tables contain addresses of data that can be placed anywhere in the default environment module except in the header area or in the tables, themselves.

Figure 97 shows one possible configuration for a default environment module that places the data pointed to by the tables at the end of the module.

```
Header area               (116 bytes)
                          (Specified as EP for the module)

COMMAND TABLE:
Number of command strings (4 bytes)
Total number of bytes for
all command strings       (4 bytes)
Command string entry      (8 bytes)
Command string entry      (8 bytes)
      .
      .
      .

STREAM TABLE:
Number of stream entries  (4 bytes)
Stream entry              (32 bytes)
Stream entry              (32 bytes)
      .
      .
      .

FUNCTION TABLE:
Number of functions       (4 bytes)
Function entry            (32 bytes)
Function entry            (32 bytes)
      .
      .
      .

Stream header
Stream name
Stream name
Command string
Command string
Function name
Function name
```

Figure 97. Default environment module structure

1. The 116-byte **header area** must be pointed to by the entry point name for the environment because it contains required data and addresses at fixed offsets into the module.

   The first 8 bytes must be the characters ‘ADFMDFLT’ to ensure that the default CSECT is the entry point in the load module. If any other CSECT is the entry point, the user receives an error message and enters TSO/E without the Session Manager.

   The second 8 bytes can be a time stamp or zeros.

   The next 76 bytes are not used. (In the IBM-supplied ADFMDFLT, this area is used for the copyright statement.)

The addresses of the installation management exit routines you can link-edit into this module start at 92 bytes, (offset X'5C'). If no exits are present, these twelve bytes must be zero.

The addresses of the three tables in the default CSECT start at 104 bytes, (offset X'68').

Figure 98 shows the storage declarations for the 116-byte header area, followed by a pictorial representation.

```
OFFSET     LABEL

           ADFMDFLT     CSECT
  0                     DC CL8'ADFMDFLT'
  8                     DC CL8'&SYSDATE'
 10                     DS CL76
 5C                     DC V(inst-exit-1)
 60                     DC V(inst-exit-2)
 64                     DC V(inst-exit-3)
 68                     DC AL4(ADFIDCMD)
 6C                     DC AL4(ADFIDSTR)
 70                     DC AL4(ADFIDFUN)
```

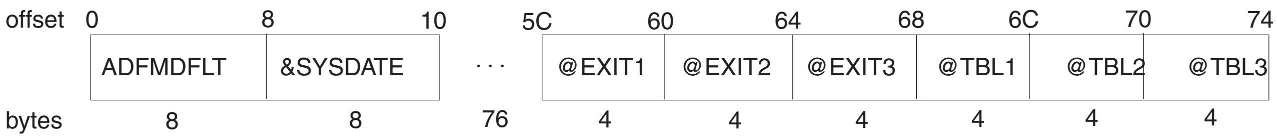


Figure 98. Default environment header

2. The **stream definitions table** begins with a fullword containing the number of streams. The remainder of the table contains contiguous 32-byte entries, one entry for each stream. Each entry contains the following information:
   - Pointer to the stream name
   - Number of bytes in the stream
   - Maximum number of lines in the stream
   - Number of lines per IDB
   - Stream type
   - Length of stream header line
   - Pointer to the stream header line
   - Flags for stream attributes

   ```
   LABEL

   ADFIDSTR     DC A(# streams)

   stream       DC AL4(stream-name-addr)
   entry#1      DC A(# bytes)
                DC A(# lines)
                DC A(# lines/IDB)
                DC A(stream-type)   0=extra
                                    1=input
                                    2=output
                DC A(header-length)
                DC A(header-addr)
                DC BL4('xx0...0')     1x...=non-wrappable
                                      x1...=alarm is on
   ```

   You must provide entries in this table for TSOIN, TSOOUT, SMIN, and SMOUT, because some Session Manager commands have operands that default to these particular stream names. Furthermore, if you intend to use the

IBM-supplied defaults for the screen layout and PF key definitions, you must also have table entries for EXTRA1, EXTRA3, and HEADER. Optionally, you can add other stream definitions.

See "Stream definition descriptions" on page 564 for considerations in defining the number of bytes, number of lines, and number of lines per IDB.

You can place the actual stream names and stream header to which this table points anywhere in the default environment module, provided they are not within the header or any of the three tables.

3. The **function definitions table** begins with a fullword containing the number of functions. The remainder of the table contains contiguous 28-byte entries, one entry for each function. Each entry contains the following information:
   - Pointer to the function name
   - Pointer to the input stream name for this function
   - Pointer to the output stream name for this function
   - Intensity of output lines
   - Pointer to the copy stream name for this function
   - Intensity of copied lines
   - Flags for function attributes

   ```
   LABEL

   ADFIDFUN     DC A(# functions)

   function     DC AL4(function-name-addr)
   entry#1      DC AL4(input-stream-name-addr)
                DC AL4(output-stream-name-addr)
                DC A(output-intensity)   0=non-display
                                         1=normal
                                         2=high
                DC AL4(copy-stream-name-addr)
                DC AL4(copy-intensity)   0=non-display
                                         1=normal
                                         2=high
                DC BL4('xx0...0')      1x...=alarm on output
                                       x1...=alarm on input
   ```

   Session Manager supports three functions and the entries must be in the order: TSO, SM, MSG. You must provide entries in this table for each. Optionally, you can add other function definitions, but Session Manager supports them only with the CHANGE.FUNCTION and QUERY.FUNCTION commands.

   You can place the actual function names and stream names to which this table points anywhere in the default environment module, provided they are not within the header or any of the three tables.

4. The **Session Manager commands table** has the commands stacked in the SM input stream. These commands initialize the screen layout and PF key definitions. The table begins with a fullword containing the number of command strings, followed by a fullword containing the total number of bytes taken up by all of the command strings. (This number does not include the first two fullwords of the table). The remainder of the table contains contiguous eight-byte entries, one entry for each command string. Each entry contains the length of the command string (four bytes), followed by the four-byte command string address.

   Figure 99 on page 571 shows the storage declarations for the Session Manager commands table, followed by a pictorial representation.

```
LABEL

ADFIDCMD     DC F'#entries'
             DC F'#bytes'

command1     DC F'#bytes'
entry #1     DC AL4(command-string-addr)

command2     DC F'#bytes'
entry #2     DC AL4(command-string-addr)
```

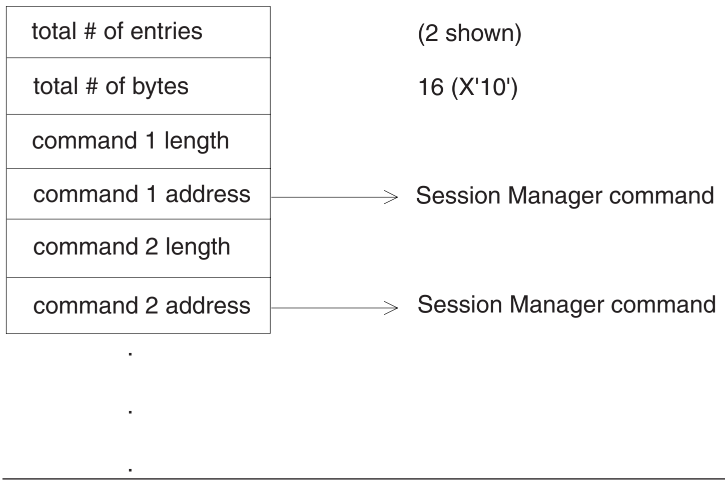


Figure 99. Session Manager Commands Table

Session Manager picks up each of these command strings and stacks them in the SM function's input stream during initialization. After Session Manager is initialized, these commands are executed.

You can put an entry in this table for any Session Manager command that defines the screen layout, PF keys, the definitions of streams, functions, terminal characteristics, and so on.

Note: You must place entries in the Session Manager commands table, otherwise there will be no screen layout in which users can enter commands or see output. The only available alternative for users would be using the CLEAR key, which allows them to enter Session Manager commands separated by commas or blanks.

You can place the actual command strings to which this table points anywhere in the default environment module, provided they are not within the header or any of the three tables.

Writing Session Manager exits

You can use three Session Manager exits to monitor users' interaction with the system while they are using Session Manager:
- Initialization exit
- Stream monitoring exit
- Termination exit

You can use these exits to:
- Monitor and intercept certain commands
- Retain a log of a user's TSO/E session
- Determine how long it takes a command to execute

- Determine at what time certain operations were performed
- Provide multiple default environments

For detailed information, see "Possible uses" on page 579.

Functional description

Following is a description of the three exits.

Initialization exit
Use the initialization exit to indicate to Session Manager which streams you want to monitor. You also indicate whether Session Manager should log line-mode output while users are executing full-screen programs. The initialization exit receives control from the Session Manager initialization program, ADFMDF0A. This exit does not hold any locks.

Stream monitoring exit
Use the stream monitoring exit to monitor the different Session Manager streams and perform required processing. The stream monitoring exit receives control, holding the local lock, from either of the following Session Manager programs:
- ADFIMPUT, which puts a line into a stream
- ADFIMGET, which gets a line of input from a stream

For information about locks and serialization, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Before ADFIMPUT puts a line into a stream (excluding the TSO/E input stream), it determines if the stream is being monitored. If it is, ADFIMPUT invokes the stream monitoring exit routine, then puts the line into the stream.

Whenever ADFIMGET retrieves the next line of input from a stream, it determines if that stream is being monitored. If it is, ADFIMGET invokes the stream monitoring exit routine. By waiting to invoke the exit after the line is retrieved from the input stream instead of invoking the exit when the line was put into the stream, the exit routine receives the command that will now execute, not an input line that is being stacked into the input stream.

Termination exit
Use the termination exit to perform required processing before Session Manager ends. The termination exit receives control from the Session Manager initialization program (ADFMDF0A) holding the local lock. The exit gets control whenever the user enters either the following:
- Session Manager END command
- TSO/E LOGOFF command

TSO/E-supplied exits

TSO/E does not provide default exit routines for any of the three Session Manager exits.

Entry specifications

For the Session Manager initialization, stream monitoring, and termination exits, the contents of the registers on entry are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

See the following topics for a description of the parameter lists for each exit.

Parameter descriptions for initialization exit

On entry, register 1 contains the address of the following parameter list:

Offset	Length	Value
+0	4	Pointer to the user ID. The user ID is left-justified and padded with blanks.
+4	4	Pointer to installation data. The installation data is one fullword of zeros. It is reserved for the exit's use. Session Manager saves its contents across all calls to the stream monitoring exit.
+8	4	Pointer to a bit mapping. The bit mapping is one byte in length and is initially set to zero. The bits represent the different streams that you can monitor. The initialization exit sets the bits to indicate which streams are to be monitored and if line mode output is to be logged while in a full-screen program. The individual bits are defined as follows: **`1... ....`** TSO/E input stream **`.1.. ....`** TSO/E output stream **`..1. ....`** Session Manager input stream **`...1 ....`** Session Manager output stream **`.... 1...`** MSG output stream **`.... .1..`** Log line mode output when in a full-screen program **`.... ..11`** Reserved The particular stream names themselves are not monitored, but the input/output streams of particular functions are monitored. For example, if the initialization exit sets the “TSO/E input stream” bit, the TSO/E function's input stream will be monitored. This is not necessarily the TSOIN stream because the user can issue the CHANGE.FUNCTION command to make the TSO/E input stream something else.

Note: You need to save the address of the bit mapping in the installation data fullword if you want to change the stream monitoring selections dynamically. For an example of changing the exits dynamically to monitor and intercept certain commands, see "Possible uses" on page 579.

Parameter descriptions for stream monitoring exit

On entry, register 1 contains the address of the following parameter list:

Offset	Length	Value
+0	4	Pointer to the user ID. The user ID is left-justified and padded with blanks.
+4	4	Pointer to installation data. The installation data is one fullword and is reserved for the installation's use. This fullword contains one of the following: • Zeros • The data that the initialization exit put there • The data that the last call to the stream monitoring exit put there Session Manager saves the installation data across calls to the exits.
+8	4	Pointer to a bit mapping. The bit mapping is one byte in length. It indicates an output stream to which stream the line is being written or an input stream from which a line is being read. **Note:** Although the entire bit mapping is initialized, only one stream bit is set when a command is processed to indicate the current function being performed. **1... ....** TSO/E input stream **.1.. ....** TSO/E output stream **..1. ....** Session Manager input stream **...1 ....** Session Manager output stream **.... 1...** MSG output stream **.... .1..** Log line mode output when in a full-screen program **.... ..11** Reserved The bits do not represent particular stream names. They represent only the input or output stream of a particular function. For example, if the first bit is set, the TSO/E function's input stream is monitored. This is not necessarily the TSOIN stream because the user can issue the CHANGE.FUNCTION command to make the TSO/E input stream something else.

Offset	Length	Value
+12	4	Pointer to a time stamp. The time stamp is the two fullwords returned by a STORE CLOCK instruction that ADFIMPUT executes prior to calling the exit. The 31st bit of the first word represents 1.048576 seconds.
+16	4	Pointer to the length of control data. The length is a 2-byte field that represents the number of Session Manager control characters in the line being written to the stream. The length may be zero.
+20	4	Pointer to the length of the data. The length is a 2-byte field that represents the number of text characters being written to the stream.
+24	4	Pointer to the control data. The control data indicates special characteristics of the line that must be handled when the line is viewed by a window on the screen. The control data bits are defined as follows:
		`1... ....` The line is displayed with high intensity.
		`.1.. ....` The line is non-display.
		`..11 111.` Reserved
		`.... ...1` The line is to be treated as TPUT ASIS.
		The first two bits cause the line to be displayed at either high intensity or non-display.
		If the last bit is on, Session Manager performs only minimal editing on the line before it displays the line (the same editing as TPUT ASIS). Normally, Session Manager performs the same editing as TPUT EDIT. For more information about editing, see *z/OS TSO/E Programming Services*.
+28	4	Pointer to the text of the line being written to the stream.
		If you change the text of the line itself (change it to blanks or different characters), you must not change the length of the line.

For a discussion of performance considerations associated with writing a stream monitoring exit record, see "Possible uses" on page 579.

Parameter descriptions for termination exit

On entry, register 1 contains the address of the following parameter list:

Offset	Length	Value
+0	4	Pointer to the user ID. The user ID is left-justified and padded with blanks.
+4	4	Pointer to installation data. The installation data is one fullword and is reserved for the installation's use. Session Manager saves its contents across calls to all three exits.

Offset	Length	Value
+8	4	Pointer to a time stamp. The time stamp is the two fullwords returned by a STORE CLOCK instruction that ADFIMPUT executes prior to calling the termination exit. The 31st bit of the first word represents 1.048576 seconds.

For a description of using the time stamp, see "Possible uses" on page 579.

Return specifications

The contents of the registers upon return must be the same as on entry. Register 15 is not checked for a return code.

Programming considerations

This topic describes general considerations for writing the exits. For general exit use considerations, see "Possible uses" on page 579.

Each exit must follow standard linkage conventions. It must save the registers on entry and restore the registers when it returns.

In the initialization exit, specify the streams you want to monitor. The initialization exit gets control holding no locks. It can run holding a lock, but it must return control holding no locks. To use the stream monitoring exit, you must have an initialization exit.

The stream monitoring exit holds the local lock and receives control from either ADFIMPUT or ADFIMGET. The stream monitoring exit must not release the local lock because Session Manager uses it for serialization. If the exit releases the lock, the results are unpredictable.

The length fields, control data, and text that the stream monitoring exit receives constitute the actual line that either ADFIMPUT or ADFIMGET is processing. The stream monitoring exit must not change the length fields or the results are unpredictable. The exit can change the control data or the text as long as the lengths remain the same. Neither ADFIMPUT or ADFIMGET checks the validity of these parameters when the exit returns control.

The termination exit gets control holding the local lock. The termination exit must not release the local lock because Session Manager uses it for serialization. If the exit releases the lock, the results are unpredictable.

In the three exits' parameter lists, one fullword (installation data) is reserved for the exit routines' use. Session Manager saves the information in this fullword across all calls to all three exits. You can use this fullword to pass information between the exits.

ADFIMPUT is called each time a line is put into any stream. This is a mainline path, but there is a negligible increase in the number of instructions in ADFIMPUT to determine if it should invoke the stream monitoring exit and then invoke the exit. You should consider the following:

- Depending upon the streams you are monitoring, ADFIMPUT could call the stream monitoring exit every time it puts a line into a stream. If this happens, and the exit has a considerable amount of function, performance could be degraded.

- To reduce the number of unnecessary invocations of the stream monitoring exit, the exit can dynamically change the mapping that defines the streams being monitored.
- If you monitor one group of users differently from other groups, the overhead of monitoring one group does not have to affect others. You can supply different sets of exits in the groups' default environment modules.

If a user defines the same stream as the input or output stream of two or three functions, the following can occur:

- ADFIMPUT calls the stream monitoring exit for *every* line that is going into a monitored stream, even if the line is being put into the stream as a result of another function. For example, suppose you are monitoring MSG output and the user has TSOOUT as both the TSO/E and MSG output stream. The stream monitoring exit gets control for all TSO/E output because all of the lines have a target stream equal to the MSG output stream.
- The bit mapping that the stream monitoring exit receives indicates only the stream for which a match was found between the target stream and the function's stream. (In the example described above, the bit mapping always indicates MSG output, even though many of the lines are TSO/E output lines.)

Environment

- State: Supervisor
- Key: 1
- AMODE and RMODE considerations

  The three Session Manager exits can run in either 24-bit or 31-bit addressing mode and can reside above or below 16 MB in virtual storage. Modules residing below 16 MB in virtual storage (RMODE=24) can have an AMODE of 24, 31, or ANY. Exits that reside in the same load module must have the same addressing mode.

  If you link-edit exits with AMODE(24) into either the default environment module or the ADFMDF0A load module, you must include the following statement on the SYSLIN DD statement of the link-edit step. This ensures that the AMODE of the load module is 24.

  ```
  MODE AMODE(24),RMODE(24)
  ```

  If you link-edit exits with AMODE(31) into either the default environment module or the ADFMDF0A load module, you do not have to add a MODE statement on the link-edit step.

  For information about 31-bit addressing, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Installing the exits

Naming requirements for the exits depend on how you link-edit the exits. You can link-edit any or all of the exits into the ADFMDF01 load module. If you link-edit the exits into ADFMDF01, you must name the exits as follows:

Initialization exit
: ADFEXIT1

Stream monitoring exit
: ADFEXIT2

Termination exit
: ADFEXIT3

You can link-edit any or all of the exits into any default environment module. The default environment module that TSO/E provides is named ADFMDFLT.

Your installation may create different default environment modules for different users. You can also link-edit the exits into any of these modules. For information about creating your own environment modules, see “Providing multiple default environments” on page 567.

If you link-edit the exits into a default environment module, there is no restriction on the names of the exits or on the name of the default environment module. However, offsets into the default CSECT must contain the addresses of the exits as follows:
- Offset X'5C' must contain the address of the initialization exit
- Offset X'60' must contain the address of the stream monitoring exit
- Offset X'64' must contain the address of the termination exit

Any initialization, stream monitoring, or termination exit you provide within the default environment module overrides the corresponding exit named ADFEXIT1, ADFEXIT2, or ADFEXIT3 in the ADFMDF01 load module.

Note: To have a stream monitoring exit, you must have an initialization exit. A termination exit is not required.

You can place the exits in the same or different load modules, however, all exits in the same load module must have the same AMODE. No other restrictions pertain to where you can place the exits. You can provide ADFEXIT1, ADFEXIT2, and ADFEXIT3 in ADFMDF01, then, for some default environments, you can override any or all of these exits with different exits to customize the monitoring of certain classes of users. For an example of monitoring certain classes of users differently, see “Possible uses” on page 579.

Figure 100 on page 579 shows how Session Manager locates the addresses of the exits during initialization.

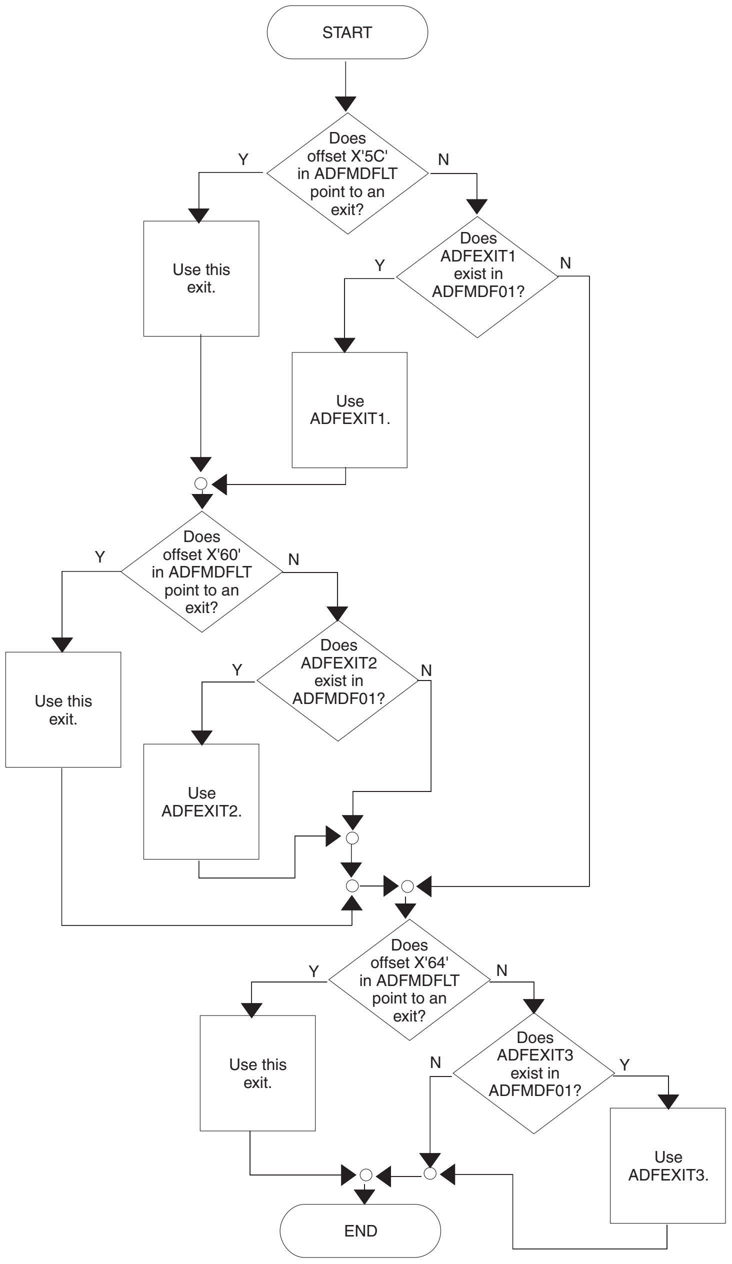

START
Does offset X'5C' in ADFMDFLT point to an exit?
Y
N
Use this exit.
Does ADFEXIT1 exist in ADFMDF01?
Y
N
Use ADFEXIT1.
Does offset X'60' in ADFMDFLT point to an exit?
Y
N
Use this exit.
Does ADFEXIT2 exist in ADFMDF01?
Y
N
Use ADFEXIT2.
Does offset X'64' in ADFMDFLT point to an exit?
Y
N
Use this exit.
Does ADFEXIT3 exist in ADFMDF01?
N
Y
Use ADFEXIT3.
END

Figure 100. Overview of how Session Manager determines which exit to use

Possible uses

The Session Manager exits provide you with a wide range of monitoring capabilities. You can monitor all users in the same way, monitor each class of users differently, or group various classes of users together and monitor the groups differently. Use the Session Manager exits to:

- Monitor all of the users with the same logon procedure in the same way
- Monitor and intercept certain commands
- Retain a log of a user's session
- Determine how long it takes a command to run or determine at what time certain operations were performed

The following describes these four possible uses and provides an example initialization exit.

- Monitor all of the users with the same logon procedure in the same way

  You can use multiple exits with logon procedures to provide different default environments for certain classes of users and to monitor groups of users in the same way.

  For example, suppose your installation has two logon procedures for Session Manager:
  - LOG1 - specifies the default environment module ENV1
  - LOG2 - specifies the default environment module ENV2

  If you had three Session Manager exits (EX1, EX2, and EX3), you could link-edit them into the ENV1 and ENV2 modules. All Session Manager users would be monitored by the exits regardless of whether they log on with the LOG1 or LOG2 procedure.

  To monitor users who use the LOG2 procedure differently, you could write a second stream monitoring exit (NEWEX2). Link-edit NEWEX2 into the ENV2 default environment module or incorporate NEWEX2 directly into the ENV2 module and reassemble ENV2. Then, users who log on with the LOG1 procedure are monitored by the EX1, EX2, and EX3 exits; users who use the LOG2 procedure are monitored by the EX1, NEWEX2, and EX3 exits.

- Monitor and intercept certain commands

  You can monitor the various streams to intercept particular commands and perform some action based on the user and the command. You can monitor and intercept:
  - TSO/E commands and output
  - Program input and output
  - Session Manager commands and output
  - Messages that users receive

  For example, you can monitor TSO/E input and restrict specific users or groups of users from using certain commands. When the exit intercepts the command, it can change it to blanks or some other characters. If you eliminate commands in this way, you must not change the length of the line.

  If you monitor an input stream, you should be aware that the command may not be ready to execute because other commands may be stacked ahead of it. At this point, the input line is being placed into the input stream. If you want the exit to perform an action on that specific command, detect the command as it is copied to the output stream. At that point, Session Manager is returning the command to TSO/E in order for it to execute.

  If you want the exit to intercept a specific command as it is copied to the output stream, you do not have to monitor every line of output. Monitor only the input until the exit detects the command. The exit can then start monitoring the output until the command is copied to the output stream. The exit can perform the required action and then go back to monitoring input only.

  To dynamically change which streams are being monitored, use the initialization and stream monitoring exits. The initialization exit can save the address of the stream mapping in the "installation data" fullword. The stream monitoring exit can change the stream mapping that defines which streams are being monitored. For example, if you are monitoring TSO/E input and detect a particular command, you can start monitoring the TSO/E output stream. When the next command is retrieved for TSO/E input, you can then go back to monitoring only TSO/E input.

- Retain a log of a user's session

To monitor the input, output, and message streams and write each line to a data set, use the stream monitoring exit. To create a hard copy of the data set, use the termination exit.

The stream monitoring exit must not do any I/O operations. To prevent performance degradation, the initialization exit can attach another task during initialization to handle I/O requests from the stream monitoring exit. Whenever the stream monitoring exit is ready to write a record, it can post the I/O task and pass it a buffer with the record. The stream monitoring exit can then return to ADFIMPUT, which lets Session Manager continue while the I/O task is asynchronously writing the record.

The task that the initialization exit attaches runs without holding the local lock; therefore, the stream monitoring exit must not release the local lock. If the stream monitoring exit releases the local lock, unpredictable serialization problems within Session Manager can occur.

- Determine how long it takes a command to run or determine at what time certain operations were performed.

  The stream monitoring exit receives a time stamp with each line that it receives. You can use the time stamp to:

  – Monitor the length of time it takes for particular commands to run

    The stream monitoring exit can monitor the input. When it intercepts a specific command, it can save the time stamp and begin monitoring the output. When it detects the READY mode message, it can determine how long the command took to execute by calculating the difference between the saved time stamp and the time stamp provided with the READY mode message line.

  – Provide a time stamp on lines that you save for subsequent hard copy listings.

    If you save a TSO/E session for subsequent hard copy, you can convert the time stamp that the stream monitoring exit receives for each line to a time stamp that can be printed. You can then include this with each line. When you review the session in hard copy, you can see the time at which certain operations were performed.

Example initialization exit

Figure 101 on page 582 shows an example initialization exit. The exit indicates to Session Manager that line mode output should be logged while in a full-screen program.

```
**********************************************************************
*                                                                    *
* MODULE NAME = ADFEXIT1                                             *
*                                                                    *
* DESCRIPTIVE NAME = INSTALLATION MANAGEMENT INITIALIZATION EXIT    *
*                                                                    *
* FUNCTION = EXAMPLE EXIT OF HOW TO INDICATE TO SESSION MANAGER     *
*           TO LOG LINE MODE OUTPUT WHILE IN FULL-SCREEN PROGRAMS  *
*                                                                    *
* INPUT =                                                            *
*      REG 1 - POINTER TO A LIST OF ADDRESSES OF THE FOLLOWING      *
*              PARAMETERS:                                           *
*                1) USERID                                           *
*                2) INSTALLATION DATA                                *
*                3) STREAM AND OUTPUT AND LOGGING INDICATOR BIT:    *
*                     BIT 0 - MONITOR TSO/E INPUT STREAM            *
*                     BIT 1 - MONITOR TSO/E OUTPUT STREAM           *
*                     BIT 2 - MONITOR SESSION MANAGER INPUT STREAM  *
*                     BIT 3 - MONITOR SESSION MANAGER OUTPUT STREAM *
*                     BIT 4 - MONITOR MSG OUTPUT STREAM             *
*                     BIT 5 - TURN ON LOGGING OF LINE MODE OUTPUT   *
*                             WHILE IN FULL-SCREEN PROGRAMS         *
*      REG 14 - RETURN ADDRESS TO SESSION MANAGER MAINLINE          *
*      REG 15 - ENTRY POINT ADDRESS                                  *
*                                                                    *
* PROCESSING =                                                       *
*      1) RETRIEVE ADDRESS OF THIRD PARAMETER                        *
*      2) TURN ON BIT 5 TO INDICATE TO TURN ON LOGGING OF OUTPUT    *
*      3) RETURN TO SESSION MANAGER MAINLINE                         *
*                                                                    *
**********************************************************************
ADFEXIT1  CSECT                  INSTALLATION INITIALIZATION EXIT
          USING  *,15
          STM    14,12,12(13)
          L      1,8(1)          3RD PARM IS PTR TO INDICATOR BITS
          OI     0(1),B'00000100' BIT 5 INDICATES TO LOG ISPF OUTPUT
          LM     14,12,12(13)
          BR     14              RETURN TO SESSION MANAGER MAINLINE
          END
```

Figure 101. Initialization exit to log line mode output while in full-screen programs

Part 9. Information Center Facility.

The TSO/E Information Center Facility.enables you to implement an MVS-based information center at your installation. The Information Center Facility. contains two conversational, panel-driven interfaces. One enables users to access products and services and the other enables administrators to maintain the facility. You, the system programmer, can customize the Information Center Facility. in many ways to suit the specific needs of your installation.

This part describes the tasks that you do to prepare the Information Center Facility. for use, customize it, and diagnose problems you may cause when you make changes. To help you understand how to perform these tasks, this part includes background information about the structure of the Information Center Facility.Information Center Facility..

For an overview of the products and services available through the Information Center Facility., see *z/OS TSO/E General Information*. That document also shows the primary panels for the users and the administrators of the Information Center Facility.

Chapter 47. Preparing the Information Center Facility for use

After you have installed TSO/E and done any conversion that might be required, you can prepare the Information Center Facilityfor use at your installation. This section describes the tasks that you may want to do before making the Information Center Facility available to your users. As background information, this section describes the structure of the Information Center Facility and lists the products that users can access through it.

The tasks that you may want to do before making the Information Center Facility available for use are:

- Changing the location of program libraries
- Identifying CLISTs and REXX execs to VLF
- Making products available
  - Products supported by the Information Center Facility
  - General consideration for all products
  - Considerations for specific products.
- Creating or tailoring application definitions for users and user groups
- Changing Information Center Facility defaults
- Defining printers to Information Center Facility
- Making installation changes available
- Making performance decisions for the Information Center Facility names service
- Estimating space requirements
- Setting the APPC/MVS administration dialog defaults

Information Center Facility structure

The Information Center Facility consists of the following elements that you can change to suit your installation's needs:

- **Panels**

  The Information Center Facility panels are written using Interactive System Productivity Facility (ISPF) panel definition. Panel definitions are programmed descriptions that define the content and format of the panels. During customization, you are primarily concerned with the following types of panels: menu, non-display, tutorial, and HELP panels. The IBM-supplied panels are members of the partitioned data set named ICQ.ICQPLIB. The Application Manager panels, ICQAMED1, ICQAMED4 and ICQAMED5, reside in that data set as model panels. The models do not contain all of the information that users see on panels. Application Manager dynamically generates menu panels and the panels for the DESCRIBE option by taking information from the Application Manager tables in data set ICQ.ICQCMTAB and displaying it on the model panels.
- **CLISTs and REXX Execs**

  The Information Center Facility is written in the TSO/E CLIST and REXX languages. The CLISTs and REXX execs reside in the partitioned data set ICQ.ICQCCLIB.
- **Messages**

The Information Center Facility displays messages on its panels. The messages use the ISPF format for message definition. The messages are located in the partitioned data set ICQ.ICQMLIB. Each member of ICQ.ICQMLIB contains up to ten messages.

- **Tables**

  The Information Center Facility stores information in two-dimensional arrays called tables and accesses the information using the ISPF table services. The Information Center Facility generates the names of the tables. Each table is stored as a member of a partitioned data set. The members have the same names as the tables they contain. The following partitioned data sets contain tables and are created during the installation of TSO/E:
  - ICQ.ICQTLIB
  - ICQ.ICQAATAB
  - ICQ.ICQABTAB
  - ICQ.ICQANTAB
  - ICQ.ICQAPTAB
  - ICQ.ICQGCTAB
  - ICQ.ICQAMTAB
  - ICQ.ICQCMTAB

  Application Manager dynamically creates two table data sets for each defined group level of application definitions and one data set for each defined private level of application definitions. See “Naming conventions: application definition tables and installation files” on page 654.

- **Applications**

  When you are using Application Manager to define products or services to the Information Center Facility, you need to describe the products or services in terms of three types of applications: panel, function, and environment. The first Application Manager panel lists the applications included in the Information Center Facility. You use the different types of applications for the following purposes:

  Panel To define the external interface that users see. Panels enable users to link to other panels, invoke functions, and display HELP text.

  ICQADMIN (TSO/E - Administration) is an example of a panel application. ICQADMIN specifies the information to be displayed on the administrator's primary panel and defines the linkage to the services listed on that panel.

  Function

  To provide the method for invoking the product or service, such as a CLIST, command, program, or menu (excluding the menu panels that Application Manager generates). A function can also contain the support information that environments contain. However, if the support information applies to more than one function, you may want to describe the information in an environment.

  ICQNEWSA (TSO/E News - Maintenance) is an example of a function application. ICQNEWSA contains the following invocation command used to invoke the news service:

  ```
  CMD(%ICQANC00)
  ```

  ICQNEWSA does not define any variables itself, but refers to the ICQENVIRON environment, which defines several variables.

  Environment

  To provide support information, such as:

  - Commands for setup, invocation, and termination
  - Data set allocations for associated panels, messages, tables, and load modules
  - ISPF shared variables.

  ICQENVIRON (TSO/E Information Center Facility environment) is an example of an environment application. ICQENVIRON defines the date and language variables that are used by products and services in the Information Center Facility.

- **Installation files**

  These files are sequential data sets or members of partitioned data sets that contain the information that Application Manager needs to define an application to the Information Center Facility. The information in installation files corresponds to the input fields on the Application Manager panels used to define applications. Using Application Manager, installations have a choice of defining applications by providing information on the Application Manager panels or by providing installation files. For examples of installation files, see "Example using installation files when adding a service" on page 631.

- **Skeletons**

  Skeleton are models containing some type of data that is used repeatedly. Each skeleton is stored as a member of a partitioned data set. The Information Center Facility contains only one skeleton and that is stored in the partitioned data set, ICQ.ICQSLIB. Education services use the skeleton to create Interactive Instructional Presentation System (IIPS) registration requests. You will probably not use skeletons to customize the Information Center Facility.

Information Center Facility libraries

The following table summarizes information about the Information Center Facility libraries that contain the elements you may want to change. For information about the block size for these libraries, see the appropriate installation manual.

Table 107. Information Center Facility libraries

IBM-Supplied Data Set Name	Ddname	Description	RECFM	LRECL
ICQ.ICQPLIB	ISPPLIB	Panel library	FB	80
ICQ.ICQMLIB	ISPMLIB	Message library	FB	80
ICQ.ICQSLIB	ISPSLIB	Skeleton library	FB	80
ICQ.ICQTLIB	ISPTLIB ICQTABL	General table library	FB	80
ICQ.ICQAATAB	ICQAATAB	Names table library	FB	80
ICQ.ICQABTAB	ICQABTAB	Administrator courses table library	FB	80
ICQ.ICQANTAB	ICQANTAB	Administrator news table and text library	FB	80
ICQ.ICQAPTAB	ICQAPTAB	Printer support table library	FB	80
ICQ.ICQAMTAB	ICQAMTAB	Administrator Application Manager table library	FB	80
ICQ.ICQCMTAB	ICQCMTAB	User Application Manager table library	FB	80

Table 107. Information Center Facility libraries (continued)

IBM-Supplied Data Set Name	Ddname	Description	RECFM	LRECL
ICQ.ICQGCTAB	ICQGCTAB	IIPS/IIAS Registration table library	FB	80
ICQ.ICQCCLIB	SYSPROC	User and administrator CLIST and REXX exec library	FB	80

Changing the location of program libraries

The Information Center Facilitycontains several dialog functions that are coded as programs. The load modules for these programs (ICQAMLD0, ICQAMLF0, ICQAMLI0, ICQGCL00, ICQGCL10, ICQCALN1, and ICQCAL00) are link-edited into SYS1.CMDLIB when TSO/E is installed. You can concatenate the load modules to the ISPF link library (ddname=ISPLLIB, RECFM=U) or move them into:

- The link pack area (LPA), which makes them available to all of your users. When you install the modules, you must re-IPL your system to make them available to users.
- A system link library such as SYS1.LINKLIB, which makes them available to all of your users.
- A logon STEPLIB, which is helpful for limited use and for testing. However, the use of a STEPLIB for all of your users can result in extra search time required to locate and invoke the modules.

For more information about LPA, SYS1.LINKLIB, and STEPLIB, see *z/OS MVS Initialization and Tuning Reference*.

Identifying CLISTs and REXX execs to VLF

The Information Center Facility contains several dialogs that are coded as CLISTs or REXX execs. You can improve the performance of the CLISTs/REXX execs the EXEC command finds at the system-level library (SYSPROC) and application-level library (defined using the TSO/E ALTLIB command), by using the virtual lookaside facility (VLF). To improve performance, define the IKJEXEC class name in the VLF PARMLIB member, COFVLFxx.

```
⋮
CLASS NAME (IKJEXEC)
  EDSN(ICQ.ICQCCLIB)     /* (IBM-supplied) */
⋮
```

Each eligible data set name (EDSN) entry in that member identifies a data set. The order of the entries is not significant. For more information about COFVLFxx, see *z/OS MVS Initialization and Tuning Reference*.

Making products available

The Information Center Facility enables users to access a variety of products and services. The Information Center Facility does not actually contain any licensed products. To use licensed products, you must order them separately.

This topic lists the products you can access through the Information Center Facility and describes the general and product-specific tasks that you must do to make products available to users.

Products supported by the Information Center Facility

As shipped by IBM, the Information Center Facility supports the following IBM licensed programs with menu panels, HELP panels, tutorials, and dialogs:

- APL and APL-based associated products:
  - APL2® (5668-899): A Programming Language 2
  - VS/APL (5748-AP1): Virtual Storage/A Programming Language
  - APLDI-II (5796-PNJ): APL Data Interface II
  - ADRS-II (5796-PLN): A Departmental Reporting System II with Business Graphics (ADRS-II/BG)
  - Info Center/1 (5668-897): Information Center/1
  - FPS (5798-CXP): Financial Planning System - TSO

    When using the Information Center Facility, you can access the FPS routines only through ADRS-II or Info Center/1. Note that IBM no longer provides service for FPS. Consult the following publications for information about installing the FPS routines or ADRS with the BG feature and the FPS routines under ADRS-II:
    - *A Departmental Reporting System User's Guide*
    - *A Departmental Reporting System Systems Guide*
    - *A Departmental Reporting System II Business Guide*
    - *The Financial Planning System - TSO Systems Guide*
- Other associated products:
  - AS (5767-001): Application System
  - GDDM/PGF
  - IBM BASIC/MVS (5665-948)
  - IIPS/IIAS: Interactive Instructional Presentation System (5668-012)/Interactive Instructional Authoring System (5668-011)
  - PS/TSO (5665-346): Personal Services/TSO
  - QMF™ (5668-972): Query Management Facility™
  - TIF (5665-339): The Information Facility

For a description of the products and services that users can access through the Information Center Facility as shipped by IBM, see *z/OS TSO/E General Information*.

General consideration for all products

As shipped by IBM, the status of all products and services displayed on Information Center Facility menu panels is unavailable to users. If a product or service is to be used at an installation, the status of the applications that define it, must be changed to available. An Information Center Facility administrator can use Application Manager panels to change the status of applications to available. For a description of how to use Application Manager panels to make an application available, see *z/OS TSO/E Administration*.

Considerations for specific products

APL2, VS/APL, GDDM/PGF, IIPS, and APLDI-II have additional set-up tasks that you may want to do before making them available to users.

Making APL2 available

For APL2 to be available, you must ensure that the status indicator is set to **available** and the variable QCLAPL2 is set to **yes**. For APL2 to be unavailable, only the status indicator needs to be set to **unavailable**. When shipped by IBM, QCLAPL2 is set to **yes** by the Application Manager function ICQAPL2. Do not change the value of the variable QCLAPL2. If you change the value to **no**, administrators will not be able to make APL2 available by only setting the status indicator to **available**.

Making VS/APL available

For VS/APL to be available, you must ensure the status indicator is set to **available** and the variable QCLVSAPL is set to **yes**. For VS/APL to be unavailable, only the status indicator needs to be set to **unavailable**. When shipped by IBM, QCLVSAPL is set to **yes** by the Application Manager function ICQVSAPL. Do not change the value of the variable QCLVSAPL. If you change the value to **no**, administrators will not be able to make VS/APL available by only setting the status indicator to **available**.

Naming GDDM/PGF libraries

Set the following variables in the non-display panel ICQSIECG to the appropriate library names:

Table 108. Variables for GDDM/PGF libraries

Set:	To:
&QCGICU	Library name containing the ICU load module
&QCGISE	Library name containing the ISE load module
&QCGVSE	Library name containing the VSE load module
&QCGSYMBL	Library name containing the default symbols.

Changing the names of IIPS data sets

The Information Center Facility uses a default high-level qualifier of IIPS for IIPS data sets. If your installation specifies a different qualifier when installing IIPS, change the Information Center Facilitydefault. To change the default:

1. Select the COURSES option on the primary selection panel for Information Center Facility administrators.
2. Select the DEFAULTS option on the next panel you see (the INFORMATION CENTER COURSES panel).
3. When the COURSES-MODIFY ADMINISTRATION DEFAULTS panel is displayed, enter the high-level qualifier your installation uses.

The Information Center Facility uses the IISBATCH program to process registration requests. The BCONFIG member of the IIPS.OS.CTLCARD data set contains control statements for IISBATCH. (Your installation might use a qualifier other than IIPS in the data set name.) Before you can register students in a course using the Information Center Facility, BCONFIG must contain the statement DISKnn=YES, where nn corresponds to the number of the data set in which the course resides. The DISKnn statement identifies the course data set to IISBATCH. If the statement is omitted, registration fails. Therefore, your installation needs to ensure that BCONFIG contains a DISKnn=YES statement for each data set that might contain a course in which students can request registration using the Information Center Facility.

Simplifying END PF key use in APLDI-II

On a system-wide basis, you can modify APL Data Interface II (APLDI-II) to alter the way users leave APLDI and return to an invoking program, such as the Information Center Facility.

The unmodified version of APLDI requires the user to press the END PF key twice when responding to an APLDI prompt. Pressing the END PF key the first time (or entering a null line) terminates the APLDI “SELECTION:” prompt. Pressing the END PF key again saves the user's work in a CONTINUE workspace, terminates APL, and returns to the invoking program. After you install this modification, users can press the END PF key only once to leave APLDI, save the workspace, terminate APL, and return to the invoking program.

To make the modification, enter APL and change line 2 of the GF function in APLDI as described in the function ICQINP of workspace ICQUPDTS.

Creating and tailoring application definitions

Information Center Facility administrators and users can define and invoke up to three levels of application definitions. A system-wide level is supported for applications that are defined for an entire system. A group level is supported for user groups to create their own application definitions and to override system applications with definitions tailored to a particular group. A user (or private) level is supported for individual users to create their own application definitions and to override group and system application definitions. All levels can be implemented in the Application Manager definition dialogs and all levels can be active at invocation time. See *z/OS TSO/E Administration*.

Changing Information Center Facility defaults

You can change the default values for particular variables in the Information Center Facility. The default values for Information Center Facilityproducts and services are specified in non-display panels and in the ICQENVIRON environment. To change the default value for a variable defined in a non-display panel, you:
- Edit the non-display panel
- Find the first occurrence of the variable
- Change the value of the variable.

To change the default value for a variable defined in the ICQENVIRON environment, use Application Manager. For information about how to use Application Manager to change a default value for a variable, see *z/OS TSO/E Administration*.

This section describes how to change the system defaults that are used when setting up a user profile in the Information Center Facility and how to change the date format, which is used by several services and products in the Information Center Facility. It also describes the default variables that you can change for specific Information Center Facility products and services. It does not include all of the variables that are initially set in each non-display panel, but only those that you might want to change. The variables apply to the following products and services:
- Names
- Interface to GDDM/PGF
  - Libraries that contain program load modules
  - Interactive Chart Utility (ICU)
  - Image Symbol Editor (ISE)
  - Vector Symbol Editor (VSE)
- Education services
  - Names of the interactive and batch execution programs
  - Starting point for searching education services data sets
- Enrollment

- VS APL, APL2, APLDI-II, and ADRS-II
- Application Manager

Changing the system default ISPF profile

The Information Center Facility uses the system default ISPF profile supplied by ISPF/PDF. ISPF/PDF maintains a set of modifiable values on behalf of each user. Each ISPF/PDF user can modify these defaults by selecting the PDF profile option (normally option 0). The enroll process of the Information Center Facility creates this default set of values on behalf of the user being enrolled. The administrator, during the enroll process, can modify the defaults for the user being enrolled. The administrator can also modify the initial set of defaults supplied by ISPF/PDF using the SYSDEF option on the administration main menu panel. When the administrator modifies the initial defaults supplied by ISPF/PDF, a local copy of the profile is maintained by the Information Center Facility. The defaults supplied by ISPF/PDF are then no longer used. See *z/OS TSO/E Administration* for more information.

Changing the date format

Variables QCCDFMTX and QCCDFMT, defined in the ICQENVIRON environment application, contain the date format used in date processing for the names and news services, Application Manager, and APLDI-II. Using Application Manager panels, you can set QCCDFMT to the date format that you want users to see. QCCDFMT can contain special language characters. Also using Application Manager panels, you must set QCCDFMTX to the equivalent date format using English characters (“mm” for month, “dd” for day, and “yy” for year). The Information Center Facility issues an error message if the first two characters of the date format in QCCDFMTX are not English characters.

Changing variables for names

The variables for changing the Information Center Facilitynames function are located in the non-display panel ICQSIECA. Table 109 shows the variables you can change and their default values.

Table 109. Variables for names

Variable	Contents	Default value
QCANTXT	The name of the user's nickname data set. The TRANSMIT and RECEIVE commands search this data set for nicknames. The names function adds :ALTCTL. tags for the other nickname data sets to this data set so that TRANSMIT and RECEIVE search the other data sets, if needed. If you change the USRCTL parameter of the Interactive Data Transmission Facility, you must change the value of the QCANTXT variable. The USRCTL parameter changes the name of the user's nickname data set. **Note:** The name you supply for QCANTXT must not be a fully-qualified data set name. The names function automatically prefixes the user ID to the name of the data set.	NAMES.TEXT

Table 109. Variables for names (continued)

Variable	Contents	Default value
QCAPNTXT	The name of the nickname data set for the private directory. **Note:** The name you supply for QCAPNTXT must not be a fully-qualified data set name. The names function automatically prefixes the user ID to the name of the data set.	ICQNAMES.TEXT
QCAPPRI	The primary space allocation of the nickname data set for the private directory.	5
QCAPSEC	The secondary space allocation of the nickname data set for the private directory.	1
QCAPUNIT	The allocation units for the nickname data set for the private directory.	TRACKS
QCAPRECL	The logical record length of the nickname data set for the private directory.	255
QCAPBKSZ	The block size of the nickname data set for the private directory.	2550
QCAPRCFM	The record format of the nickname data set for the private directory.	F B
QCAPINCR	The percentage by which space management increases the space allocated to the nickname data set for the private directory when it is running out of space. For more information on space management, see "Changing the defaults for managing data set space" on page 220.	50
QCAMNTXT	The name of the nickname data set for the master directory. If you specify a data set name, it must be a fully-qualified name without quotes.	ICQ.MASTER.NAMES.TEXT
QCAMPRI	The primary space allocation of the nickname data set for the master directory.	20
QCAMSEC	The secondary space allocation of the nickname data set for the master directory.	2
QCAMUNIT	The allocation units for the nickname data set for the master directory.	TRACKS
QCAMRECL	The logical record length of the nickname data set for the master directory.	255
QCAMBKSZ	The block size of the nickname data set for the master directory.	2550
QCAMRCFM	The record format of the nickname data set for the master directory.	F B
QCAMINCR	The percentage by which space management increases the space allocated to the nickname data set for the master directory when it is running out of space. For more information on space management, see "Changing the defaults for managing data set space" on page 220.	50

Table 109. Variables for names (continued)

Variable	Contents	Default value
QCAALTC	The name of another nickname data set that is to be searched if a name is not found in the user, private, or master nickname data sets.	null
QCACATAB	The name of the private table library.	ICQNAMES.DIR
QCAPRI	The primary space allocation of the private table library.	10
QCASEC	The secondary space allocation of the private table library.	5
QCADIR	The number of directory blocks for the private table library.	2
QCAUNIT	The allocation units for the private table library.	TRACKS
QCALRECL	The logical record length of the private table library.	80
QCABLKSZ	The block size of the private table library.	3120
QCARECFM	The record format of the private table library.	F B
QCASTAT	Use this variable to indicate whether the names directory tables should: • Remain open after they are initially opened (OPEN). • Be closed after each use and reopened when they are needed again (CLOSE). For more information about the considerations for leaving the tables open, see "Making performance decisions for names service" on page 611.	CLOSE

Changing variables for the APPC/MVS administration dialog

The Information Center Facilityprovides an interface to the APPC/MVS administration dialog. You can customize the APPC/MVS administration dialog by changing certain variables contained in the non-display panel ICQASE00. Table 110 shows the variables you can change and their default values.

Table 110. Variables for APPC/MVS administration dialog

Variable	Contents	Default value
QASSDIN	Allocation attributes for the data set used as input to APPC/MVS administration	BLKSIZE(3100) SPACE(50,10)
QASSDOUT	Allocation attributes for the data set used for SYSOUT output from APPC/MVS administration	BLKSIZE(3100) SPACE(50,10)
QASSDPRT	Allocation attributes for the data set used for SYSPRINT output from APPC/MVS administration	BLKSIZE(3100) SPACE(50,10)
QASSDATA	Allocation attributes for the data set used to contain the JCL for an ASCH transaction program profile or to contain the scheduler information for a non-ASCH profile.	BLKSIZE(3100) SPACE(50,10)

Table 110. Variables for APPC/MVS administration dialog (continued)

Variable	Contents	Default value

QASTSPE	A map of transaction scheduler exits. The value on the left is the value used in the transaction scheduler field name. The value on the right is the program/module name of the transaction scheduler exit that is called by APPC/MVS administration to check the syntax of the scheduler data. For example, if the scheduler name is XYZ and the exit to check the syntax is XYZEX01, specify 'XYZ,XYZEX01'.
	
	You can have more than one transaction scheduler, but ASCH,ASCH is required. To specify more than one transaction scheduler, separate them with a + as in the following example:
	`ASCH,ASCH +`
	`ABC,ABCEXIT +`
	`XYZ,XYZEX01`
		ASCH,ASCH

QASCLASS	The available classes for an ASCH-scheduled transaction program profile. These classes are defined in one or more ASCHPMxx parmlib members. A list of these classes appears when a user enters an * (asterisk) next to the SCHEDULER CLASS keyword. For more information about the ASCHPMxx parmlib members, see *z/OS MVS Planning: APPC/MVS Management*.	none
QASMODDF	The data set name that contains JCL models. As an aid to the person adding transaction program profiles scheduled by ASCH, sample models can be created and placed in the data set named here.	ICQ.*.*
QASTPDEF	The name of the default VSAM KSDS that contains transaction program profiles.	SYS1.APPCTP
QASSIDEF	The name of the default VSAM KSDS that contains side information.	SYS1.APPCSI
QASDBDEF	The name of the default VSAM KSDS that retrieves or updates data base tokens.	SYS1.APPCTP

Temporary storage data files

When you enter the APPC/MVS administration dialog utility, four temporary files are created. When you exit the utility, these files are erased. These four temporary files are:

```
<user prefix>.TEMP.SYSSDIN
<user prefix>.TEMP.SYSSDOUT
<user prefix>.TEMP.SYSSDPRT
<user prefix>.TEMP.SYSSDATA
```

Setting the CANCEL PF key

When you enter the APPC/MVS administration dialog utility, your PF12 key is set to CANCEL, if a PF key has not been previously set to CANCEL. When you exit the APPC/MVS administration dialog, the PF12 key, if it was set to CANCEL, returns to its original value.

REQAPPC command

In MVS/ESA SP Version 4 Release 1, you can set the REQAPPC variable on or off. You can set the REQAPPC variable by entering `REQAPPC ON` or `REQAPPC OFF` from the command line of any APPC/MVS administration panel.

When REQAPPC is set on, the APPC/MVS administration panel will not allow you to add, modify, or copy ASCH transaction program profiles. APPC/MVS needs to be present to successfully process ASCH transaction program profiles.

When REQAPPC is set off, the administrator can add, modify, or copy ASCH transaction program profiles. If you add, modify, or copy ASCH transaction program profiles, the JCL syntax is *not* checked. Because JCL syntax is not checked, set the Activation Status to inactive when you are adding, modifying, or copying ASCH transaction program profiles. When you migrate to MVS/ESA SP Version 4 Release 2, you can set the Activation Status to active and JCL syntax checking will occur.

In MVS/ESA SP Version 4 Release 2 or later, the REQAPPC command will have no effect on adding, modifying, or copying transaction program profiles. The transaction program profile JCL syntax checking will always occur.

Changing variables for the interface to GDDM/PGF

The Information Center Facility provides an interface to the Graphic Data Display Manager (GDDM/PGF). Certain variables for the interface are located in the non-display panel ICQSIECG. You can change the settings of variables for the:
- Libraries that contain program load modules
- Interactive Chart Utility (ICU)
- Image Symbol Editor (ISE)
- Vector Symbol Editor (VSE)

Changing variables for libraries that contain program load modules

Table 111 shows the variables you can change and their default values.

Table 111. Variables for libraries that contain program load modules for GDDM/PGF

Variable	Contents	Default value
QCGICU	The name of the library that contains the ICU load module.	SYS1.GDDMLOAD
QCGISE	The name of the library that contains the ISE load module.	SYS1.GDDMLOAD
QCGVSE	The name of the library that contains the VSE load module.	SYS1.GDDMLOAD
QCGSYMBL	The name of the library that contains the default symbol set.	SYS1.GDDMSYM

Changing variables for the interactive chart utility

The Interactive Chart Utility uses the following files:
- Data file
- Format file
- Graphic data format (GDF) file

To customize the allocation of the files that the Interactive Chart Utility uses, change the default settings for the variables shown in Table 112 on page 597.

Table 112. Variables for the interactive chart utility

Variable	Contents	Default value
QCGPRI1	The primary space allocated to the data file.	5
QCGSEC1	The secondary space allocated to the data file.	5
QCGDIR1	The number of directory blocks for the data file.	10
QCGUNIT1	The allocation units for the data file.	TRACKS
QCGRECL1	The logical record length of the data file.	400
QCGBKSZ1	The block size of the data file.	400
QCGRCFM1	The record format of the data file.	F
QCGPRI2	The primary space allocated to the format file.	5
QCGSEC2	The secondary space allocated to the format file.	5
QCGDIR2	The number of directory blocks for the format file.	10
QCGUNIT2	The allocation units for the format file.	TRACKS
QCGRECL2	The logical record length of the format file.	400
QCGBKSZ2	The block size of the format file.	400
QCGRCFM2	The record format of the format file.	F
QCGPRI6	The primary space allocated to the GDF file.	5
QCGSEC6	The secondary space allocated to the GDF file.	5
QCGDIR6	The number of directory blocks for the GDF file.	10
QCGUNIT6	The allocation units for the GDF file.	TRACKS
QCGRECL6	The logical record length of the GDF file.	400
QCGBKSZ6	The block size of the GDF file.	400
QCGRCFM6	The record format of the GDF file.	F

Changing variables for the image symbol editor

The Image Symbol Editor uses the following files:
- Image file
- Object file

To customize the allocation of the files that the Image Symbol Editor uses, change the default settings for the variables shown in Table 113.

Table 113. Variables for the image symbol editor

Variable	Contents	Default value
QCGPRI3	The primary space allocated to the image file.	5
QCGSEC3	The secondary space allocated to the image file.	5
QCGDIR3	The number of directory blocks for the image file.	10
QCGUNIT3	The allocation units for the image file.	TRACKS
QCGRECL3	The logical record length of the image file.	400
QCGBKSZ3	The block size of the image file.	400
QCGRCFM3	The record format of the image file.	F
QCGPRI4	The primary space allocated to the object file.	5
QCGSEC4	The secondary space allocated to the object file.	5
QCGDIR4	The number of directory blocks for the object file.	10

Table 113. Variables for the image symbol editor (continued)

Variable	Contents	Default value
QCGUNIT4	The allocation units for the object file.	TRACKS
QCGRECL4	The logical record length of the object file.	400
QCGBKSZ4	The block size of the object file.	400
QCGRCFM4	The record format of the object file.	F

Changing variables for the vector symbol editor

To customize the allocation of the vector file, change the default settings for the variables shown in Table 114.

Table 114. Variables for the vector symbol editor

Variable	Contents	Default value
QCGPRI5	The primary space allocated to the vector file.	5
QCGSEC5	The secondary space allocated to the vector file.	5
QCGDIR5	The number of directory blocks for the vector file.	10
QCGUNIT5	The allocation units for the vector file.	TRACKS
QCGRECL5	The logical record length of the vector file.	400
QCGBKSZ5	The block size of the vector file.	400
QCGRCFM5	The record format of the vector file.	F

Changing variables for education services

Variables for the customization of education services are located in the non-display panel ICQSIECB. You can change the settings of variables for the:
- Names of the interactive and batch execution programs
- Starting point for searching education services data sets

Changing the names of the interactive and batch execution programs

The following variables for the names of the IIPS interactive and batch execution programs are set in the non-display panel ICQSIECB.
- Variable PGM is set to the default value for the name of the interactive program, IIS.
- Variable PROGM is set to the default value for the name of the batch program, IISBATCH.

If the programs are named using the default, you do not have to make any changes. However, if the programs are not installed with these names, you must change the settings of PGM and PROGM in panel ICQSIECB.

Changing the starting point for searching education services data sets

If you allocate a new data set for a course, you must change the starting point that education services uses when searching the education services data sets. When searching for a course, education services searches through the education data sets sequentially in ascending order, based on the low-level qualifier. The low-level qualifier for education services data sets is DISK*nn*, where *nn* is a number lower than 69. The variable DISKNO on panel ICQSIECB contains the lowest number (the *nn* part of the low-level qualifier) currently in use in an education services data set name.

When you allocate a new education services data set, set the number in the low-level qualifier to one less than the lowest number currently in use. After you allocate the data set, change the value of DISKNO to the number of the low-level qualifier for the new data set.

Figure 102 shows an example of adding a new education services data set. Assuming that the lowest number in a low-level qualifier for an existing education services data set is 67, the name of the new data set is ICQ.CBT.DISK66. You must set DISKNO to 66, so that education services starts searching for a course in the new data set.

Existing data sets:
IIPS.CBT.DISK69
IIPS.CBT.DISK68
IIPS.CBT.DISK67
New data set:
IIPS.CBT.DISK66
DISKNO = 66

Figure 102. Example of adding a new education services data set

Changing variables for enrollment

The non-display panel ICQSIE00 initializes the following:
- Application ID
- Name of the ISPF profile data set
- Volume for the ISPF profile data set
- Data set name for the UADS
- Generic profile for the user

ICQ, the default value of the application ID, is set in variable QAEAPPL. If you change the value of QAEAPPL, also change the variable NEWAPPL on the PROC statement of CLIST ICQICF.

ISPF.PROFILE, the default value of the ISPF profile data set name, is set in variable QAEPROF. If you change QAEPROF, also change the variable PROFILE on the PROC statement of CLIST ICQICF.

The variable QAEVOL contains the number of the volume on which the ISPF profile data set is to be allocated. The default value is blank, which permits the data set to be allocated on any volume. If you want the ISPF profile data set allocated on a particular volume, set QAEVOL to the volume number.

The variable QAEUADSD contains the data set name for the UADS. The default is SYS1.UADS. You must specify a fully-qualified data set name. The data set is created when you install TSO/E and must exist to enroll users in the UADS.

Use the variable QAEGENER to specify whether you want a generic profile created for the users you enroll. Specify Y (yes) or N (no). The default is N. If you set QAEGENER to Y, the defaults for the RACF ADDSD command are:

Field	Variable	Default
UACC	QAEUACC	None
OWNER	QAEOWNER	If blank, it is the user ID of the person you are enrolling.
AUDIT	QAEAUDIT	FAILURES
AUDIT-ACCESS-LEVEL	QAEAUDLV	READ
LEVEL	QAELEVEL	0

If you want additional keywords on the RACF ADDSD command, you must change the ADDSD command in the CLIST ICQAEC00.

Changing variables for VS APL, APL2, APLDI-II, and ADRS-II

The non-display panel ICQSIECR contains variables for VS APL, APL2, APLDI-II, and ADRS-II. You can change the values of the variables to provide defaults that match your own requirements. If you have both VS APL and APL2 installed, consider using the same defaults for the equivalent VS APL and APL2 variables. Using the same defaults, simplifies maintenance and provides similar results for the two products.

If you change any defaults on panel ICQSIECR, also reset the values for the variables TESTDATE and TESTTIME in ICQSIECR. When a user enters VS APL, APL2, ADRS, or APLDI, ICQSIECR is invoked.

Note: Do not code &SYSDATE and &SYSTIME to reset TESTDATE and TESTTIME. &SYSDATE and &SYSTIME provide the current date and time and using them would ensure that TESTDATE and TESTTIME never matched the saved values.

Changing the default values for VS APL

For VS APL, you can change the default values for the variables shown in Table 115.

Table 115. Variables for VS APL

Variable	Contents	Default value
QCLAPV	The APL product variable.	APL
QCLAPT	The product name that appears on interface panels.	VS APL
QCLAPSM	The default for the SESSION MGR parameter. You can set the variable to ON or OFF. If you set the variable to ON, the VS APL Session Manager is used. If you set it to OFF, the VS APL Session Manager is not used.	OFF
QCLAPQ	The default for the QUIET parameter. You can set the variable to null or QUIET. If you set the variable to null, normal terminal output is displayed on invocation. If you set the variable to QUIET, terminal output is suppressed until the first time input is needed from the terminal.	null

Table 115. Variables for VS APL (continued)

Variable	Contents	Default value
QCLAPHL	The default for the HILIGHT parameter. You can set the variable to the following values: • ON to highlight all lines • OFF to turn off all highlighting • INPUT to highlight only input lines • OUTPUT to highlight only output lines	OFF
QCLAPAS	The default for the AISIZE parameter. The value is the amount of storage, in kilobytes, to be reserved for input lines stacked by the alternate input auxiliary processor. You can specify a value from 1 to 9999.	1
QCLAPFS	The default for the FREESIZE parameter. The value is the amount of virtual storage, in kilobytes, to be reserved for MVS/TSO functions that the auxiliary processor uses. You can specify a value from 64 to 9999.	64
QCLAPSR	The default for the SHRSIZE parameter. The value is the amount of storage, in kilobytes, to be reserved for the shared variable processor. You can specify a value from 4 to 9999.	4
QCLAPWS	The default for the WSSIZE parameter. The value is the amount of contiguous virtual storage, in kilobytes, to be reserved for the user's active workspace. You can specify a value from 32 to 9999.	1024
QCLAPDG	The default for the DEBUG parameter. You can set the variable to null to use the normal error recovery for VS APL, or to the sum of any combination of the values representing the following operands: • MSG -- use to receive debugging messages that identify work areas in main storage as well as all secondary messages. To use MSG, add 1 to the value in QCLAPDG. • ECHO -- use to display all data passed to the alternate input auxiliary processor. To use ECHO, add 2 to the value in QCLAPDG. • ABEND -- use to avoid VS APL's recovery routines. To use ABEND, add 64 to the value in QCLAPDG. • MICRO -- use to cancel the test for VS APL microcode on your central processor. To use MICRO, add 128 to the value in QCLAPDG. **Note:** ABEND and MICRO are for the use of system programmers only.	null
QCLAPB1	The default for the first LOADLIB. Each LOADLIB is a private load library from which auxiliary processors (supplied with APNAMES) can be loaded.	null
QCLAPB2	The default for the second LOADLIB.	null
QCLAPB3	The default for the third LOADLIB.	null
QCLAPB4	The default for the fourth LOADLIB.	null

Table 115. Variables for VS APL (continued)

Variable	Contents	Default value
QCLAPB5	The default for the fifth LOADLIB.	null
QCLAPA1	The default for the first APNAMES. Each APNAMES is an auxiliary processor to be loaded from the private load libraries (supplied with LOADLIB).	null
QCLAPA2	The default for the second APNAMES.	null
QCLAPA3	The default for the third APNAMES.	null
QCLAPA4	The default for the fourth APNAMES.	null
QCLAPA5	The default for the fifth APNAMES.	null
QCLAPA6	The default for the sixth APNAMES.	null
QCLAPA7	The default for the seventh APNAMES.	null
QCLAPA8	The default for the eighth APNAMES.	null

Changing the default values for APL2

For APL2, you can change the default values for the variables shown in Table 116.

Table 116. Variables for APL2

Variable	Contents	Default value
QCLA2V	The APL product variable.	APL2
QCLA2T	The product name that appears on interface panels.	APL2
QCLA2SM	The default for the SESSION MGR parameter. You can set the variable to ON or OFF. If you set the variable to ON, the APL2 Session Manager is used. If you set it to OFF, the Session Manager is not used.	OFF
QCLA2Q	The default for the QUIET parameter. You can set the variable to null or QUIET. If you set the variable to null, normal terminal output is displayed on invocation. If you set the variable to QUIET, terminal output is suppressed until the first time input is needed from the terminal.	null
QCLA2HL	The default for the HILIGHT parameter. You can set the variable to one of the following values: • ON to highlight all lines • OFF to turn off all highlighting • INPUT to highlight only input lines • OUTPUT to highlight only output lines	OFF
QCLA2DT	The default for the DATEFORM parameter. The value is the format in which dates are printed. You can set the variable to one of the following values: • ISO (International Standards Organization) -- uses the format yyyy-mm-dd hh.mm.ss. • US (United States convention) -- uses the format mm/dd/yyyy hh.mm.ss. • EU (European convention) -- uses the format dd.mm.yyyy hh.mm.ss.	US

Table 116. Variables for APL2 (continued)

Variable	Contents	Default value
QCLA2AS	The default for the AISIZE parameter. The value is the amount of storage, in kilobytes, to be reserved for input lines stacked by the alternate input auxiliary processor. You can specify a value from 1 to 9999.	1
QCLA2FS	The default for the FREESIZE parameter. The value is the amount of virtual storage, in kilobytes, to be reserved for MVS/TSO functions that the auxiliary processor uses. You can specify a value from 64 to 9999.	64
QCLA2SR	The default for the SHRSIZE parameter. The value is the amount of storage, in kilobytes, to be reserved for the shared variable processor. You can specify a value from 4 to 9999.	4
QCLA2WS	The default for the WSSIZE parameter. The value is the amount of contiguous virtual storage, in kilobytes, to be reserved for the user's active workspace. You can specify a value from 32 to 9999.	1024
QCLA2SV	The default for the SVMAX parameter. The value is the maximum number of shared variables that you can share at the same time. You can specify a value from 4 to 32,767.	4
QCLA2DG	The default for the DEBUG parameter. You can set the variable to null to use the normal error recovery for APL2, or to the sum of any combination of the values representing the following operands: • MSG -- use to receive debugging messages identifying work areas in main storage as well as all secondary messages. To use MSG, add 1 to the value in QCLA2DG. • ECHO -- use to display all data passed to the alternate input auxiliary processor. To use ECHO, add 2 to the value in QCLA2DG. • XDUMP -- use to receive a more extensive dump. The dump may be 200 to 500 pages long, depending on the size of your workspace. To use XDUMP, add 4 to the value in QCLA2DG. • MSGID -- use to display message IDs with any error messages you receive. To use MSGID, add 32 to the value in QCLA2DG.	null
QCLA2B1	The default for the first LOADLIB. Each LOADLIB is a private load library from which auxiliary processors (supplied with APNAMES) can be loaded.	null
QCLA2B2	The default for the second LOADLIB.	null
QCLA2B3	The default for the third LOADLIB.	null
QCLA2B4	The default for the fourth LOADLIB.	null
QCLA2B5	The default for the fifth LOADLIB.	null
QCLA2A1	The default for the first APNAMES. Each APNAMES is an auxiliary processor to be loaded from the private load libraries (supplied with LOADLIB).	null
QCLA2A2	The default for the second APNAMES.	null

Table 116. Variables for APL2 (continued)

Variable	Contents	Default value
QCLA2A3	The default for the third APNAMES.	null
QCLA2A4	The default for the fourth APNAMES.	null
QCLA2A5	The default for the fifth APNAMES.	null
QCLA2A6	The default for the sixth APNAMES.	null
QCLA2A7	The default for the seventh APNAMES.	null
QCLA2A8	The default for the eighth APNAMES.	null
QCLA2E1	The default for the first EXCLUDE. Each EXCLUDE is an auxiliary processor that is normally available, but that you do not want loaded at the start of your session.	null
QCLA2E2	The default for the second EXCLUDE.	null
QCLA2E3	The default for the third EXCLUDE.	null
QCLA2E4	The default for the fourth EXCLUDE.	null
QCLA2E5	The default for the fifth EXCLUDE.	null
QCLA2E6	The default for the sixth EXCLUDE.	null
QCLA2E7	The default for the seventh EXCLUDE.	null
QCLA2E8	The default for the eighth EXCLUDE.	null
QCI2PUBQ	The default high-level qualifier for a public workspace. Change the value of this variable if you installed APL2 with changes to APLYUOPT.	V
QCI2MIDQ	The default mid-level qualifier for workspaces. Change the value of this variable if you installed APL2 with changes to APLYUOPT.	APL2
QCI2LIBQ	The high-level qualifier for the catalog containing the library number and owner for project libraries. Change the value of this variable if you installed APL2 with changes to APLYUOPT.	APL2

Changing the default values for APLDI-II

For APLDI-II, you can change the default values for the following variables:

- Variables which define the OS data set names for each of the files for APLDI, and the names APL uses to load them using)LOAD. Table 117 shows these variables.

Table 117. APLDI-II variables for defining OS data set names

Variable	Default contents
QCIWS	The APLDI workspace name for)LOAD. The initial value is 5 APLDI.
QCIDSN	The OS data set containing the APLDI workspace. The initial value is @PL.@W000005.APLDI.
QCIUWS	The DIUPDATE workspace name for)LOAD. The initial value is 5 DIUPDATE.
QCIUDSN	The OS data set containing the DIUPDATE workspace. The initial value is @PL.@W000005.DIUPDATE.
QCILOAD	The load library containing DICREATE, DIUNLOAD, DISIZE, and DIUTIL. The initial value is @PL.DI2UTIL.LOADLIB.
QCITITLE	The product title. The initial value is APLDI-II.

Table 117. APLDI-II variables for defining OS data set names (continued)

Variable	Default contents
QCIQTYPE	The default file type. The initial value is DIFILE. **Note:** The default file type must be DIFILE unless you change DICREATE.

- Variables which contain the input string APLDI execute when the user selects the QUERY and UPDATE options on panel ICQCIM00. The input strings contain a copy command, the name of the function that passes control to the requested APLDI function, and the name of the query file APL is to use. Table 118 shows these variables.

Table 118. APLDI-II variables containing the input string

Variable	Default contents
QCI1IN	')COPY 51 ICQUPDTS ICFDI1' 'ICFINQ0' '&QDSN'
QCI2IN	')COPY 51 ICQUPDTS ICFDI2' 'ICFUPD0' '&QDSN'

  The workspace 51 ICQUPDTS contains several functions to simplify the invocation of APLDI. The functions include:
  – A termination function to support the PF keys
  – The function ICFINQ to pass control to the APLDI function ICQ
  – The function ICFUPDATE to simplify updating functions of the workspace 5 DIUPDATE

  To reduce the overhead of copying these functions each time APLDI is loaded, you can copy the functions into a copy of the distributed workspaces. If you copy the functions, remove the COPY command in each variable.
- The variable which identifies the workspace and the group that provides support for terminals that do not have APL keyboards. Table 119 shows the variable.

Table 119. APLDI-II variable identifying the workspace and group for non-APL keyboards

Variable	Default contents
QCINODAF	')COPY 5 DIAUX NODAFGRP'

- Variables which identify the APLDI character set. If you change the default alphabet, you must change the alphabet specified in the query workspace. The DIUNLOAD program requires the alphabet to create a sequential file. Table 120 shows the variables.

Table 120. APLDI-II variables identifying the character set

Variable	Default contents
QCIALPH	The APLDI character set. The initial value is ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789/.*-.
QCINALP	The length of QCIALPH. The initial value is 40.
QCIALPX	Y or N to indicate whether the alphabet has been changed. The initial value is N.

- Variables used for DICREATE in ICQCIC30. Table 121 on page 606 shows these variables.

Table 121. APLDI-II variables for DICREATE

Variable	Default contents
QCIBAD	The value used to replace invalid numbers. The initial value is -32,767.
QCIINDX	Y or N to indicate whether to build an index. The initial value is N.
QCIMAXE	The maximum number of errors allowed in input records. The initial value is 100.
QCIREJ	Y or N to indicate whether a reject routine is installed. The initial value is N.
QCINPRT	The number of lines after which to print a message. The initial value is 100.
QCIVOL	The volume for query files. The initial value is null.
QCIBLKSZ	The default block size. The initial value is 13030.
QCIBLKMN	The minimum block size. The initial value is 1600.
QCIBLKMX	The maximum block size. The initial value is 13030. If you change the value of QCIBLKMX, make sure the value you use is divisible by 4.
QCINREC	The number of input records to be read at a time. The initial value is 3250.

Changing the default values for ADRS-II

For ADRS-II, you can change the default values for the following variables:

- Variables which define the name and location of the ADRS load module. Table 122 shows these variables.

Table 122. ADRS-II variables defining the name and location of the load module

Variable	Default contents
QCRWS	The ADRS workspace name for)LOAD. The initial value is 51 ADRS2.
QCRDSN	The OS data set containing the workspace. The initial value is @PL.@W000051.ADRS2.
QCRTITLE	The product title. The initial value is ADRS-II.

- Variables which contain the input strings that ADRS executes when the user selects the NEW, OLD, and COPY options on panel ICQCRM00. The input strings contain a copy command, the name of the function that passes control to the requested ADRS function, and the name of the query file APL is to use. Table 123 shows these variables.

Table 123. ADRS-II variables containing input strings

Variable	Default contents
QCR1IN	')COPY 51 ICQUPDTS ICFADRS' 'ICFADRSMSG'
QCR2IN	')COPY 51 ICQUPDTS ICFADRS' 'ICFADRSMSG'
QCR3IN	')COPY 51 ICQUPDTS ICFADRS' ')COPY &TLNAME1 DIM DIN DIW' 'ICFADRSMSG'

The workspace 51 ICQUPDTS contains several functions used to simplify the invocation of ADRS. The functions include:

– A termination function to support the PF keys
– The function ICFADRSMSG

To reduce the overhead of copying these functions each time ADRS is loaded, you can copy the functions into a copy of the distributed workspaces. If you do so, remove the COPY command in each variable.

Changing the default values for general APL variables

You can change the default values for the following variables used by more than one APL program product:

- Variables which control whether users can access workspaces in other user's catalogs. If you do not change the default, the Information Center Facilityallows users to copy, create, and save workspaces in other users' catalogs, as well as in their own. VS APL, APL2, APLDI, and ADRS normally allow users to copy, create, and save workspaces only in their own catalogs. To prevent users from accessing catalogs other than their own while using the Information Center Facility, change the values of the variables shown in Table 124 to N. The default value for each variable is Y.

Table 124. Variables controlling user access to other user's catalogs

Variable	Default contents
QCICOPYD	Y or N to control whether people can copy data from other user's catalogs into their own.
QCICOPYW	Y or N to control whether people can copy a private workspace from another user's catalog into their own.
QCICRE8W	Y or N to control whether people can create or save workspaces in other users' catalogs.

- Variables which control the catalog or qualifier that is used in naming workspaces.

  Note: These variables are for VS APL, APLDI, and ADRS only.

  If you installed APL with changes to APLYUOPT, make the corresponding changes to the variables shown in Table 125.

Table 125. Variables controlling the catalog or qualifier for naming Workspaces

Variable	Default contents
QCIAPUBQ	The high-level qualifier for public workspaces. The initial value is @PL.
QCIAMIDQ	The middle-level qualifier for workspaces.
QCIALIBQ	The high-level qualifier for the catalog containing the library number and owner for project libraries.

- The variable which contains the command and parameters that the Information Center Facilityuses to invoke APL. Table 126 describes the variable.

Table 126. Variable containing the command and parameters used to invoke APL

Variable	Default contents
QCIAAPL	VSAPL AI(10) FR(400) SH(64) SM(OFF) QUIET

  If you installed APL with the APL Session Manager and you want the APL Session Manager to be active, you should:
  – Change the parameters in the variable QCIAAPL, and
  – Include an APL Session Manager profile data set.

  In the variable QCIAAPL, specify SM(ON) and the name of the profile data set, PR(ICQ). In the profile data set, specify DISPLAY OFF and set PF keys 3 and 15 to issue the command APL IMMEDIATE)CONTINUE HOLD.

Changing variables for Application Manager

The non-display panel ICQSIEAM contains the variables for customizing Application Manager functions. You can change the variables to:

- Customize the interaction between system, group, and private levels of administration
- Specify the high-level qualifier for group table data set names
- Specify the allocation characteristics for group and private table data sets
- Specify the low-level qualifier of the data set used for hierarchy displays
- Control the report width for hierarchy displays

Table 127 describes the variables that you can change and lists their default values.

Table 127. Variables for customizing Application Manager functions

Variable	Contents	Default value
QAMINVOK	Specifies whether the application invocation processing is to search any higher-level libraries for an available version of an application that is marked unavailable at the current level. Values may be 'YES' or 'NO'. If an application is unavailable or unverified, and the value of QAMINVOK is 'NO', processing stops and a message is returned indicating the application is unavailable. The QAMINVOK variable affects searches for environments and requests for hierarchy display in the same manner. If QAMINVOK is set to 'YES', unavailable applications appearing within the hierarchy listing are marked as unavailable. If QAMINVOK is set to 'NO', only the first available application is shown.	'NO'
QAMUNACE	Specifies if you can copy or export unavailable applications from higher levels of administration. Values may be 'YES' or 'NO'. If an attempt is made to copy or export an unavailable application from a higher level, the variable QAMUNACE is interrogated. If the value of QAMUNACE is 'NO', the action is not allowed and an error message is displayed.	'NO'
QAMUNVCE	Specifies if you can copy or export unverified applications from higher levels of administration. Values may be 'YES' or 'NO'. If an attempt is made to copy or export an unverified application from a higher level, the variable QAMUNVCE is interrogated. If the value of QAMUNVCE is 'NO', the action is not allowed and an error message is displayed.	'NO'
QAMUNVVW	Specifies if you can view unavailable or unverified applications from higher levels of administration. Values may be 'YES' or 'NO'. If an attempt is made to view an unavailable or unverified application from a higher level, the variable QAMUNVVW is interrogated. If the value of QAMUNVVW is 'NO', the action is not allowed and an error message is displayed.	'NO'

Table 127. Variables for customizing Application Manager functions (continued)

Variable	Contents	Default value
QAMRECOV	Specifies whether a particular library allocation is to be saved across several nested or split-screen sessions for a user. It applies only to data sets allocated via the Application Manager LIBRARIES option. Values may be 'YES' or 'NO'. Specifying 'YES' increments the library allocation count for a single user by one. The count is saved in table ICQAMUSE. If an error occurs in one nested or split-screen session the count is decremented by one. An attempt is made to save the allocation if the count is not down to zero. For example, if an error occurs and QAMRECOV is set to YES, ICQAMUSE is checked. If ICQAMUSE is greater than one (an allocation is being used by more than one application), an attempt is made to save the allocation. Specifying 'NO' does not maintain the library allocation count and does not save ICQAMUSE after each update. If an error occurs, no attempt is made to save the allocation, and the allocation may be lost.	'YES'
The following variables are used to customize the hierarchy display function.		
QAMHIER	Low-level qualifier of generated report data set for hierarchy list.	'HIERARCH.LST'
QAMHRECL	Width of the hierarchy display. The default value of 80 supports the ability to display 17 levels of indentation or hierarchy. Eighty is also the maximum length field that can be fully viewed on most terminals.	80
The following variable defines the first portion of the group tables data set names.		
QAMGPREF	High-level qualifier of data sets that contains group level tables for administrators and users.	'ICQGROUP'
The following customizable values are used in the allocation of new group and private level table data sets. The strings that are used must be valid specifications to the ALLOCATE command and must conform to ISPF requirements for its table data sets.		
QAMALPVT	Allocation characteristics of private application libraries.	'NEW DSORG(PO) RECFM(F B) LRECL(80) BLKSIZE(3120) DIR(10) SPACE(60,15)'
QAMALGRP	Allocation characteristics of group application libraries.	'NEW DSORG(PO) RECFM(F B) LRECL(80) BLKSIZE(3120) DIR(20) SPACE(30,10)'

Making installation changes available

When you are making changes, you must decide in what data set to put the changes. If you put the changes in installation data sets instead of IBM-supplied data sets or if you add installation services or products, you must decide how to concatenate those files with the IBM files.

Deciding the data set to use for changes

You customize the Information Center Facility by changing the elements described in "Information Center Facility structure" on page 585. You can make those changes available by putting them in either the IBM-supplied data sets or in installation-supplied data sets. Be aware that if you place your changes in the IBM-supplied data sets, customizing future releases becomes more difficult. Customization then entails extracting your changes from the original data sets and adding them to the new data sets.

If you want your changed elements to reside in any data sets already allocated in the LOGON procedure or start-up CLIST/REXX exec, you do not have to change the concatenation in any installation LOGON procedures or start-up CLISTs/REXX execs. However, name these elements using a different naming convention to distinguish the installation-supplied members from IBM-supplied members. For information about the IBM-naming conventions for Information Center Facility elements, see "Using the ICF naming conventions" on page 651.

If you want your changes to reside in data sets that are not already in the data set concatenation in the installation LOGON procedure or start-up CLIST/REXX exec, you have to change the concatenation so users can access the changes you make. Again, you might want to use different member names, but if you decide not to, you must concatenate installation-supplied data sets ahead of the IBM-supplied data sets. If you use different names for the installation-supplied members, you must concatenate the installation-supplied data sets to the IBM-supplied data sets, but the order is unimportant.

When concatenating data sets, check that they all have the same record format.

IBM distributes the CLIST and REXX library, ICQ.ICQCCLIB, with a RECFM of FB and an LRECL of 80. Only CLIST and REXX libraries with the same characteristics should be concatenated. If your production CLIST/REXX data set(s) has a RECFM of VB, run the CLIST ICQSMC00, a member of SYS1.SAMPLIB, against the CLIST/REXX libraries during installation to convert the libraries to a RECFM of VB.

Concatenating installation-developed products or services

There are two ways to control access to libraries required by applications. If you define the libraries using the library function of Application Manager, the libraries required by an application will be dynamically allocated when a user selects that application. Application Manager accesses the libraries using the ISPF LIBDEF service or the TSO/E ALTLIB command, or by dynamic allocation. See *z/OS TSO/E Administration* for more information about using the Application Manager library function.

If you want to permanently allocate the libraries required by an application, you can use either LOGON procedures, or start-up CLISTs/REXX execs. The following table lists the advantages and disadvantages of the different ways you can concatenate the libraries in LOGON procedures or start-up CLISTs/REXX execs.

Concatenation sequence	Advantages	Disadvantages
1. IBM-supplied products or services 2. Installation-developed products or services (system-wide, group, and user)	• SMP keeps track of all modifications to IBM libraries • Only one copy of each IBM panel exists • Control over the primary option menu is maintained.	• Must be familiar with SMP • Modifying IBM-supplied libraries makes the next installation a little more time-consuming.
1. Installation-developed products or services (system-wide) 2. IBM-supplied products or services 3. Installation-developed products or services (group and user)	• Need not be familiar with SMP • No modifications to IBM-supplied libraries • Control over the primary option menu is maintained.	• Difficult to implement more than one set of user models.
1. Installation-developed products or services (system-wide and group) 2. IBM-supplied products or services 3. Installation-developed products or services (user)	• Need not be familiar with SMP. • No modifications to IBM-supplied libraries.	• Control over the primary option menu is *not* maintained. • System-wide modules can be overridden inadvertently.
1. Installation-developed product or services (system-wide, group, and user) 2. IBM-supplied products or services	• Need not be familiar with SMP. • Total user freedom.	• Lack of consistency throughout the installation.

Defining printers to the Information Center Facility

Before Information Center Facility users can print news items, course abstracts, or use the hard copy selection on the Information Center Facility Utilities panel, printers must be defined to the Information Center Facility support function. See *z/OS TSO/E Administration* for information about defining printers to the Information Center Facility.

Making performance decisions for names service

You, the system programmer, can change the QCASTAT variable for the names service to indicate whether the names directory tables are to remain open after they are initially opened or are closed after each use and reopened when needed again. For information about how to change QCASTAT, see Table 109 on page 592. Information Center Facility users can make the same choice on a panel in the names services. By default, the names directories are open and kept in virtual storage only until users are finished accessing them.

Specifying that the directories remain open:

- Reduces the I/O overhead of opening and closing the directories each time a user accesses them.

- Simplifies directory processing by applications that use the names service because the applications do not have to open and close the tables each time they access them.

However leaving the directories open:

- Requires additional user storage above 16 MB in virtual storage. To estimate the amount of storage needed for the names service, see "Storage required by names service."
- Prevents users from always obtaining the directory they want:
  - Users may not obtain the latest changes the administrator makes to the master directory. To obtain the latest changes, users must close and open the directory.
  - When users request just the private directory, they receive both the private and the merged directory.
  - When users request the master directory, they may receive the merged directory, and vice versa.

Note: If your system is storage-constrained, do not specify that the names tables are to remain open because leaving them open could degrade performance.

Estimating space requirements

If your installation has a storage-constraint problem, you may want to estimate some of the storage requirements for the Information Center Facility. The following topics give information that will help you estimate your storage requirements for the names service and for Application Manager.

Note: Applications containing large numbers of panels may cause CLIST storage problems.

Storage required by names service

The Information Center Facility names service maintains a system names directory and users' private names directories. These names directories help users locate other users, and assist them in transmitting messages through the system.

The names service keeps directory information in ISPF tables. There are two types of table entries: single user entries, and group entries that apply to more than one user. The size and type of an entry can vary, along with the total number of directory entries. As a result, the total size of the personal and master directories can vary. The maximum size for a single user entry is 715 bytes, with the average size being 350-400 bytes. Group entries consist of 300 bytes per group, plus an additional 15 bytes per group member. A copy of the user's personal directory table and a copy of the concatenated personal and master directory tables reside in private area virtual storage. In addition to the storage for individual and group entries, each copy requires 1K bytes of additional virtual storage.

When users update the master directory, the names service requires an additional table to hold the updates. Each update requires a maximum of 800 bytes, with the average size being 400-450 bytes. A single table holds the updates for all users and empties as the administrator processes the updates. The total size of the additional table depends on the rate at which users update the master directory and the rate at which the administrator processes those updates. A copy of the table resides in private area virtual storage, requiring 1K bytes of virtual storage, in addition to the storage for individual and group entries.

The names table and all ISPF tables are loaded above 16 MB in virtual storage.

Storage required by Application Manager

The Information Center Facility Application Manager stores data in tables that it reads into the extended private area above 16 MB in virtual storage. The average amount of storage required to invoke an application and store the corresponding tables follows:

Application	Average Amount of Storage
Panel	21 K bytes
Function	60 K bytes
Environment	5 K bytes

If group and/or private levels of application definitions are active at invocation time, increased storage is required for those Application Manager tables.

Chapter 48. Customizing the Information Center Facility

This topic describes the following ways that you can customize the Information Center Facilityto suit the needs of your installation:
- Adding, deleting, and changing a product or service
- Creating or tailoring application definitions
- Invoking an application
- Modifying Information Center Facility start-up and termination processing
- Adding commands to the command table
- Resynchronizing the enrollment default profiles
- Writing exits for ADRS, names service, and Application Manager

To help you know where to make your changes, the following reference information is included at the end of this chapter:
- Naming conventions for Information Center Facility elements
- Application, panel, and CLIST/REXX hierarchy
- Functional and help panel association

If you are unfamiliar with the Information Center Facility, you may want to read the reference information first to get an overview of the organization of the Information Center Facility. If you are familiar with the Information Center Facility you may want to glance at the reference information from time to time as you read the other topics. The reference information starts with "Using the ICF naming conventions" on page 651.

Adding a product or service

You can add any product or service to the Information Center Facility provided it can run under the ISPF dialog manager and TSO/E. For information about writing programs that run under ISPF and TSO/E, see *z/OS ISPF Services Guide*. To use Application Manager to add products or services to the Information Center Facility, you must describe the product or service in terms of its applications. For a description of the applications that you can use, see "Information Center Facility structure" on page 585.

Through Application Manager panels, the Information Center Facility administrator, or more simply the administrator, can use one of the following methods to add a product or service:
1. Provide information on several Application Manager panels to define the environments, functions, and panels associated with the product or service
2. Load installation files that contain the information necessary to define the environments, functions, and panels associated with the product or service.

From the Information Center Facilityadministrator's point of view, the second method, specifying an installation file, is simpler than the first because it involves the use of fewer panels. However, if an installation file does not exist, the system programmer must create it using the required format. For information about creating installation files, see "General considerations for creating installation files" on page 616.

An administrator can optionally use the mass installation file processing function to add products and services. This function allows you to install multiple installation files at one time using a batch method.As each installation file is successfully processed, the installed application is automatically verified and marked as available in the Application Manager tables.

TSO/E also provides an upgrade function that lets administrators distribute updates to application definitions that are used across several locations and that may have local modifications. Instead of sending a complete installation file, administrators can send an upgrade file containing only the changes. Using an upgrade file minimizes the possibility of the changes conflicting with any local modifications.

When adding a product or service, the system programmer may have to assist the administrator as follows:

- If installation files do not exist, the system programmer may need to supply the information that the administrator must enter on the Application Manager panels. *z/OS TSO/E Administration*, contains sample blank information sheets that administrators can photocopy and give to system programmers for that purpose.

  The system programmer can use the on-line help and tutorial panels for Application Manager to learn about the Application Manager. In addition, the fields on the Application Manager panels correspond directly to the entries in the installation files. For a description of the entries in the installation files, refer to "Contents of a panel installation file" on page 617, "Contents of a function installation file" on page 621, and "Contents of an environment installation file" on page 627.
- If installation files do exist, the system programmer must give their names to the administrator.

General considerations for creating installation files

You can create installation files as sequential data sets or members of a partitioned data set. The records can be fixed or variable format. Application Manager loads up to 71 columns of data per record and ignores any columns that contain line numbers. Start each entry in an installation file with an asterisk (*), followed by the name of the entry and its value. Separate the name of the entry and its value with one or more blanks. Depending on the entry, the value appears in the same record or the next record. If you continue the value of an entry in the next record, column 1 must be blank.

The first three entries in an installation file are requiredand must appear in the following order:

```
*Application Manager INSTALLATION FILE
*SYSTEM  MVS TSO/E
*PANEL or *FUNCTION or *ENVIRONMENT
```

Subsequent entries can appear in any order in an installation file and are optional in the installation file. However, Application Manager requires some of the subsequent entries and prompts the administrator for those entries when the administrator loads and verifies the file.

Comments can appear anywhere in an installation file. They start with `/*` in columns 1 and 2.

Panel and function installation files allow an INVOKING_PANEL entry, which integratesthe application with existing panels. The INVOKING_PANEL entry allows you to specify the panel on which the application will be an option, the character(s) used to select the option, and the location on the panel where the option should appear.

When the installation file is loaded, each INVOKING_PANEL entry and its accompanying AFTER or POSITION entry is verified against existing table information as follows:

- The specified invoking panel must exist on the same administrative level as the administrator performing the installation. The panel does not have to be marked as available.
- The specified selection character(s) must be unique among the existing option selection characters on the invoking panel.
- If you use the AFTER entry to place the option on the invoking panel, the application name specified must represent an existing option on the invoking panel. See “Contents of a panel installation file” for a description of the installation file entry INVOKING_PANEL.

If the INVOKING_PANEL entry verification fails, the entry is not processed and the appropriate error message is displayed on the screen. Processing continues.

After the installation file has been loaded, the VERIFY option must be selected beforethe new application can be made available to users.

Contents of a panel installation file

The following figure shows the contents of a panel installation file.

```
*Application Manager INSTALLATION FILE
*SYSTEM             system name
*PANEL
*PANEL_NAME         panel name
*LANGUAGE           language
*KEYWORD            keyword
*PANEL_TITLE        panel title
*PRIMARY_PANEL      Y/N
*MODEL_ISPF_PANEL   panel ID
*INVOKING_PANEL     panel-name
 panel-appl-language
 selection-id
***************************************************************
***********  Only one of the following two entries can be used.
***************************************************************
*AFTER              application-name
 appl-language

   OR

*POSITION number
*----------
*ADMIN_DESC         administrator description
*USER_DESC_1        first line of user description
*USER_DESC_2        second line of user description
*STARTUP_FUNCTION   start-up function name
*TERM_FUNCTION      termination function name
*OPTION             option ID
 option-name
 language
*HELP_TEXT
 first line of help text
⋮
 last line of help text
```

Figure 103. Format of a panel installation file

A description of each panel-installation-file entry follows:

***Application Manager INSTALLATION FILE**
: identifies the file as an installation file. Application Manager does not load the installation file unless the first entry is *Application Manager INSTALLATION FILE.

***SYSTEM** *system name*
: identifies the system on which the installation file is to be loaded and run. *system name* contains the characters MVS TSO/E. Application Manager does not load the installation file unless the second entry is *SYSTEM MVS TSO/E.

***PANEL**
: identifies the type of installation file, in this case panel. Application Manager does not load the installation file unless the third entry is *PANEL, *FUNCTION, or *ENVIRONMENT.

***PANEL_NAME** *panel name*
: identifies the panel installation file. *panel name* consists of 1-12 characters, the first of which must be alphabetic (A-Z) or one of the special characters ($,#,@). The remaining characters can be any combination of alphabetic, numeric (0-9), or one of the special characters ($,#,@). Application Manager requires a *panel name* when defining a panel.

***LANGUAGE** *language*
: specifies the language used on the panel. *language* can be up to 8 characters in length. The QAMLANGL variable defined in the ICQENVIRON application contains alist of the languages that are valid in ISPF. The non-English versions

of the Information Center Facility are available after the English version and must be ordered separately. Therefore, non-English versions of the Information Center Facility may not be available at your installation.

Your installation can define a default value for LANGUAGE using the ISPF ZLANG variable.If the installation file does not contain an entry for LANGUAGE, Application Manager uses the value in ZLANG. For information about ZLANG, see *z/OS ISPF Planning and Customizing*.

`*KEYWORD` *`keyword`*
provides a short descriptive name for a selectable panel. The keyword is displayed on menus and users can type it on the Option line to select the associated panel. *keyword* consists of 1-11 characters, the first of which must be alphabetic (A-Z). The remaining characters can be any combination of alphabetic, numeric (0-9), or one of the special characters ($,#,@). Application Manager requires a *keyword* when defining a panel.

`*PANEL_TITLE` *`panel title`*
specifies the title that appears at the top of the panel. *panel title* can be up to 50 characters in length. Application Manager requires *panel title* when defining a panel.

`*PRIMARY_PANEL` *`Y/N`*
indicates whether the panel is a primary panel.A primary panel is the starting point for selecting options and the panel to which you return after issuing the ISPF RETURN command. ISPF uses the primary panel as the base for the jump function.The jump function enables you to go to any option on the most recently displayed primary panel by typing its option ID preceded by an equal sign in any field on a subsequent panel.

Set PRIMARY_PANEL to *Y* to indicate that the panel is a primary panel or *N* to indicate that it is not a primary panel. If the installation file does not contain *PRIMARY_PANEL, Application Manager assumes that the panel is not a primary panel.

`*MODEL_ISPF_PANEL` *`panel ID`*
specifies the *panel ID* of an existing ISPF panel to use as a model for a menu.The ISPF panel acts as a framework upon which to place the directions, options, keywords, and descriptions. If the installation file does not contain *MODEL_ISPF_PANEL, Application Manager assumes that the model panel is ICQAMED1.

`*INVOKING_PANEL` *`panel name`*
`panel-appl-language`
`selection-id`
specifies that the application is to automatically become an option on an existing panel.

panel name follows INVOKING_PANEL in the first record of an entry and specifies the name of an existing panel on which to install the application.

panel-appl-language appears in the next record, and specifies the language of the existing panel. The language of the panel must be specified because an installation can have two panels with the same name, but in different languages.

selection-id appears on the invoking panel for the installed application in the next record of the entry. *selection-id* specifies the selection character(s) that is to appear on the invoking panel for the installed application.

The INVOKING_PANEL entry can appear as many times as desired in an installation file, causing the installed application to become an option on several panels. Each INVOKING_PANEL entry must be accompanied by either an AFTER entry or a POSITION entry.

***AFTER** *application name*
appl-language
specifies that the panel is to be placed as an option after an existing option on the invoking panel.

application name follows AFTER in the first record of an entry and specifies the name of an existing application.

appl-language appears in the next record and specifies the language of the existing application. This entry is valid only if it is immediately preceded by an INVOKING_PANEL entry.

***POSITION** *number*
specifies the numeric position, among the existing options, in which to place the panel being installed.

A specified POSITION argument causes existing options equal to and greater than the number specified to be advanced (moved down) one position. If the POSITION argument is greater than the number of existing options, the panel being installed is placed last on the invoking panel. This entry is valid only if immediately preceded by an INVOKING_PANEL entry.

***ADMIN_DESC** *administrator description*
provides a short description of the application that administrators see when applications are listed. *administrator description* can be up to 38 characters in length. Application Manager requires the *administrator description* when defining a panel.

***USER_DESC_1** *first line of user description*
***USER_DESC_2** *second line of user description*
provides the first and, if needed, the second line of the description of an application that appears after the keyword on a menu that the users see. Each line can be up to 59 characters in length. Application Manager requires the *first line of user description* when defining a panel.

***STARTUP_FUNCTION** *startup function name*
provides the name of a function to be invoked before the panel is displayed. *startup function name* can be up to 12 characters in length.

***TERM_FUNCTION** *termination function name*
provides the name of a function to be invoked when the user presses the END PF key after the panel is displayed. *termination function name* can be up to 12 characters in length.

***OPTION** *option ID*
option name
language
identifies each option that users can select from the panel being added.

option ID, which follows OPTION in the first record of an entry, is displayed on the menu being defined. Users can type *option ID* on the Option line of the panel to select the associated function or panel. *option ID* contains 1-3 characters and can be any combination of alphabetic (A-Z), numeric (0-9), or one of the special characters ($,#,@).

option name, which appears in the next record, specifies the function or panel to be invoked when a user selects this option. *option name* can be up to 12 characters long.

language, which appears in the next record of the entry, specifies the language of the function or panel. *language* can be up to 8 characters long.

```
*HELP_TEXT
  first line of help text
:

  last line of help text
```

provides a short description of the panel being added. The text is part of the help information that users see when they choose the DESCRIBE option for a panel that contains this panel as an option. Each line of the help text can be up to 58 characters in length.

Contents of a function installation file

The following figure shows the contents of a function installation file.

```
*Application Manager INSTALLATION FILE
*SYSTEM             system-name
*FUNCTION
*FUNCTION_NAME      function-name
*LANGUAGE           language
*KEYWORD            keyword
*ENVIRONMENT_NAME   environment-name
*ISPF_APPL_ID       ISPF-application-id
*INVOKING_PANEL     panel-name
 panel-appl-language
 selection-id
**************************************************************
***********  Only one of the following two entries can be used.
**************************************************************
*AFTER              application-name
 appl-language

   OR

*POSITION number
*----------
*ADMIN_DESC         administrator description
*USER_DESC_1        first line of user description
*USER_DESC_2        second line of user description
*VARIABLE           variable-name
 description
 value line 1
:
 value line n
*LIBRARY            ddname        type
 data set 1
:
 data set n
*INIT_COMMAND
 initialization command line 1
:
 initialization command line n
*INVOCATION_COMMAND
 invocation command line 1
:
 invocation command line n

*TERM_COMMAND
 termination command line 1
:
 termination command line n
*ISPTUTOR_PANEL     tutorial-panel ID
*TUTORIAL_COMMAND
 tutorial command line 1
 tutorial command line 2
*HELP_TEXT
 first line of help text
 second line of help text
:
 last line of help text
```

Figure 104. Format of a function installation file

A description of each function-installation-file entry follows:

***Application Manager INSTALLATION FILE**
identifies the file as an installation file. Application Manager does not load the installation file unless the first entry is *Application Manager INSTALLATION FILE.

***SYSTEM** *system name*
identifies the system on which the installation file is to be loaded and run.

system name contains the characters MVS TSO/E. Application Manager does not load the installation file unless the second entry is *SYSTEM MVS TSO/E.

`*FUNCTION`
identifies the type of installation file, in this case function. Application Manager does not load the installation file unless the third entry is *PANEL, *FUNCTION, or *ENVIRONMENT.

`*FUNCTION_NAME` *`function name`*
identifies the function installation file. *function name* consists of 1-12 characters, the first of which must be alphabetic (A-Z) or one of the special characters ($,#,@). The remaining characters can be any combination of alphabetic, numeric (0-9), or special characters. Application Manager requires a *function name* when defining a function.

`*LANGUAGE` *`language`*
specifies the language used in the function. *language* can be up to 8 characters in length. The QAMLANGL variable defined in the ICQENVIRON application contains alist of the languages that are valid in ISPF. The non-English versions of the Information Center Facility are available after the English version and must be ordered separately. Therefore, non-English versions of the Information Center Facility may be unavailable at your installation.

Your installation can define a default value for LANGUAGE using the ISPF ZLANG variable.If the installation file does not contain an entry for LANGUAGE, Application Manager uses the value in ZLANG. For information about ZLANG, see *z/OS ISPF Planning and Customizing*.

`*KEYWORD` *`keyword`*
provides a short descriptive name for a selectable function. The keyword is displayed on menus and users can type it on the Option line to select the associated function. *keyword* consists of 1-11 characters, the first of which must be alphabetic (A-Z). The remaining characters can be any combination of alphabetic, numeric (0-9) or one of the special characters ($,#,@). Application Manager requires *keyword* when defining a function.

`*ENVIRONMENT_NAME` *`environment name`*
specifies the name of an existing environment that supports this function. *environment name* can be up to 12 characters in length.

`*ISPF_APPL_ID` *`ISPF application ID`*
specifies the prefix identifier that ISPF is to use for the user profile, edit profile, and commands associated with this function. *ISPF application ID* consists of 1-4 characters, the first of which must be alphabetic. The remaining characters can by any combination of alphabetic (A-Z), numeric (0-9), or one of the special characters ($,#,@).

`*INVOKING_PANEL` *`panel name`*
`panel-appl-language`
`selection-id`
specifies that the function is to automatically become an option on an existing panel.

panel name follows INVOKING_PANEL in the first record of an entry and specifies the name of an existing panel on which to install the function.

panel-appl-language appears in the next record, and specifies the language of the existing panel. The language of the panel must be specified because an installation can have two panels with the same name, but in different languages.

selection-id appears on the invoking panel for the installed application in the next record of the entry. *selection-id* specifies the selection character(s) that is to appear on the invoking panel for the installed application.

The INVOKING_PANEL entry can appear as many times as desired in an installation file, causing the installed function to become an option on several panels. Each INVOKING_PANEL entry must be accompanied by either an AFTER entry or a POSITION entry.

`*`**`AFTER`** *`application name`*
`appl-language`

specifies that the function is to be placed as an option after an existing option on the invoking panel.

application name follows AFTER in the first record of an entry and specifies the name of an existing application.

appl-language appears in the next record and specifies the language of the existing application. This entry is valid only if it is immediately preceded by an INVOKING_PANEL entry.

`*`**`POSITION`** *`number`*

specifies the numeric position, among the existing options, in which to place the function being installed. A specified POSITION argument causes existing options equal to and greater than the number specified to be advanced (moved down) one position. If the POSITION argument is greater than the number of existing options, the function being installed is placed last on the invoking panel. This entry is valid only if immediately preceded by an INVOKING_PANEL entry.

`*`**`ADMIN_DESC`** *`administrator description`*

provides a short description of the application that administrators see when applications are listed. *administrator description* can be up to 38 characters in length. Application Manager requires the *administrator description* when defining a function.

`*`**`USER_DESC_1`** *`first line of user description`*
`*`**`USER_DESC_2`** *`second line of user description`*

provides the first and, if needed, the second line of the description of an application that appears after the keyword on a menu that the user sees. Each line can be up to 59 characters in length. Application Manager requires the *first line of user description* when defining a function.

```
*VARIABLE variable name
  description
  value line 1
.
.
.
  value line n
```

identifies variables that the function requires. Any variable that the function requires must be defined in the function application or the environment application for the function.

The *variable name* identifies the variable and can be up to 8 characters in length.

The *variable description* provides a short description of the variable and can be up to 64 characters in length.

value line 1 through *value line n* specify the value of the variable. Application Manager appends the information from data columns 2 through 72 (fixed format) of each line to form the value, which can be up to 923 characters in length. If the records that contain the data have line numbers, they are ignored. You cannot define variables in terms of other variables.

Considerations for DBCS Data: Because double-byte character set (DBCS) data takes up from 2 to 4 bytes for every character, the installation may need to add place holder bytes at the end of the variable value line. Application Manager recognizes X'00' as the place holder value.

For example, to change a blank to X'00', enter on the COMMAND line:

```
c ' ' x'00'
```

Place the cursor on the character you want to change (usually between columns 2 and 72) and press the Enter key.

***LIBRARY** *ddname type*
  data set 1
⋮

  data set n

identifies the data set(s) that is to be allocated under the specified ddname. The order in which you specify the data set names in the subsequent records determines the concatenation order.

ddname specifies the ddname under which the data sets are to be concatenated. *ddname* can be up to 8 characters in length.

type specifies the library type. Valid values are:

ISPF ISPF files

CLIST CLIST files

EXEC REXX files

INPUT
: Any other type of input file

  During the process of allocation, if an INPUT type library has no current allocations, its ddname and list of data sets will be allocated. The allocation will specify SHR, which allows shared access to the library and data sets.If another application is invoked using the same INPUT library, the allocation is performed only when the list of data sets for both applications match.

  When the library type is INPUT, and the data set information does not match, the second application is not invoked. An error message appears stating that the requested function cannot be invoked because the required files are in use.

OUTPUT
: Any other type of output file Only one data set can be specified for the OUTPUT type library.The allocation will specify OLD, which allows exclusive access to the library and data set. If an OUTPUT type library is currently allocated, the function cannot be invoked because the required files are in use. If OUTPUT is entered and more than one data set is specified, an error message appears.

data set 1 through *data set n* specify the data sets in the library. Each can be up to 46 characters in length. If you specify the high-level qualifier as part of the data set name, enclose the data set name in single quotation marks. If you do not enclose the data set name in quotes, the user's TSO/E prefix is used as the high-level qualifier.

***INIT_COMMAND**

```
  initialization command line 1
.
.
.
  initialization command line n
```

identifies the panels, commands, or programs that execute before the function is invoked. The INIT_COMMAND uses the ISPF SELECT service to invoke one of the following:

- PANEL(panel name) OPT(option)
- CMD(command)
- PGM(program name) PARM(parameter(s))

initialization command line 1 through *initialization command line n* (where n is 1, 2, or 3) form the INIT_COMMAND. Application Manager appends data from columns 2 through 72 (fixed format) or from columns 10 through 80 (variable format) of each line to form the command, which can be up to 213 characters in length. For information about the SELECT service, see *z/OS ISPF Services Guide*.

Note: Do not use variables in command text because nulls will be substituted for these variables.

***INVOCATION_COMMAND**

```
  invocation command line 1
.
.
.
  invocation command line n
```

identifies the panels, commands, or programs that invoke the function. If a function does not have an INVOCATION_COMMAND, its environment must have one. The INVOCATION_COMMAND uses the ISPF SELECT service to invoke one of the following:

- PANEL(panel name) OPT(option)
- CMD(command)
- PGM(program name) PARM(parameter(s))

invocation command line 1 through *invocation command line n* (where n is 1, 2, or 3) form the INVOCATION_COMMAND. Application Manager appends data from columns 2 through 72 (fixed format) or from columns 10 through 80 (variable format) to form the command, which can be up to 213 characters in length. For information about the SELECT service, see *z/OS ISPF Services Guide*.

Note: Do not use variables in command text because nulls will be substituted for these variables.

***TERM_COMMAND**

```
  termination command line 1
.
.
.
  termination command line n
```

identifies the panels, commands, or programs that execute after the function terminates. The TERM_COMMAND uses the ISPF SELECT service to invoke one of the following:

- PANEL(panel name) OPT(option)
- CMD(command)

- PGM(program name) PARM(parameter(s))

termination command line 1 through *termination command line n* (where n is 1, 2, or 3) form the TERM_COMMAND. Application Manager appends data from columns 2 through 72 (fixed format) or from columns 10 through 80 (variable format) to form the command, which can be up to 213 characters in length. For information about the SELECT service, see *z/OS ISPF Planning and Customizing*.

Note: Do not use variables in command text because nulls will be substituted for these variables.

```
*ISPTUTOR_PANEL tutorial panel ID
*TUTORIAL_COMMAND
  tutorial command line 1
  tutorial command line 2
```

identify the way that the tutorial is to be invoked. ISPTUTOR_PANEL and TUTORIAL_COMMAND are mutually exclusive. If you specify both, Application Manager uses the first.

tutorial panel ID, which identifies the first tutorial panel, can be up to 8 characters in length.

tutorial command line 1 and *tutorial command line 2* form the tutorial command used to invoke the tutorial. Application Manager appends the data from columns 2 through 72 (fixed format) or from columns 10 through 80 (variable format) to form the command, which can be up to 142 characters in length.

```
*HELP_TEXT
  first line of help text
.
.
.
  last line of help text
```

provides a short description of this function that is being added. The text is part of the help information that users see when they choose the DESCRIBE option for a panel that contains this function as an option. Each line of the help text can be up to 58 characters in length.

Contents of an environment installation file

The following figure shows the contents of an environment installation file.

```
*Application Manager INSTALLATION FILE
*SYSTEM             system-name
*ENVIRONMENT
*ENVIRONMENT_NAME   environment-name
*LANGUAGE           language
*ENVIRONMENT_DESC   environment-description
*ISPF_APPL_ID       ISPF-application-id
*VARIABLE           variable-name
 description
 value line 1
:
 value line n
*LIBRARY            ddname     type
 data set 1
:
 data set n
*INIT_COMMAND
 initialization command line 1
:
 initialization command line n
*INVOCATION_COMMAND
 invocation command line 1
:
 invocation command line n
*TERM_COMMAND
 termination command line 1
:
 termination command line n
```

Figure 105. Format of an environment installation file

A description of each environment-installation-file entry follows:

***Application Manager INSTALLATION FILE**
identifies the file as an installation file. Application Manager does not load the installation file unless the first entry is *Application Manager INSTALLATION FILE.

***SYSTEM** *system name*
identifies the system on which the installation file is to be loaded and run. *system name* contains the characters MVS TSO/E. Application Manager does not load the installation file unless the second entry is *SYSTEM MVS TSO/E.

***ENVIRONMENT**
identifies the type of installation file, in this case environment. Application Manager does not load the installation file unless the third entry is *PANEL, *FUNCTION, or *ENVIRONMENT.

***ENVIRONMENT_NAME** *environment name*
identifies the environment installation file. *environment name* consists of 1-12 characters, the first of which must be alphabetic (A-Z) or one of the special characters ($,#,@). The remaining characters can be any combination of alphabetic, numeric (0-9), or one of the special characters ($,#,@). Application Manager requires an *environment name* when defining an environment.

***LANGUAGE** *language*
specifies the language used in the environment. *language* can be up to 8 characters in length. The QAMLANGL variable defined in the ICQENVIRON application contains alist of the languages that are valid in ISPF. The non-English versions of the Information Center Facility are available after the English version and must be ordered separately. Therefore, non-English versions of the Information Center Facility may not be available at your installation.

Your installation can define a default value for LANGUAGE using the ISPF ZLANG variable.If the installation file does not contain an entry for LANGUAGE, Application Manager uses the value in ZLANG. For information about ZLANG, see *z/OS ISPF Planning and Customizing*.

***ENVIRONMENT_DESC** *environment description*

provides a short description of the application that administrators see when applications are listed. *environment description* can be up to 38 characters in length. Application Manager requires *environment description* when defining an environment.

ISPF_APPL_ID *ISPF application ID*

specifies the prefix identifier that ISPF is to use for this environment. *ISPF application ID* consists of 1-4 characters, the first of which must be alphabetic. The remaining characters can by any combination of alphabetic (A-Z), numeric (0-9), or one of the special characters ($,#,@).

***VARIABLE** *variable name*
description
value line 1
⋮
value line n

identifies variables that one or more functions require.

The *variable name* identifies the variable and can be up to 8 characters in length.

The *variable description* provides a short description of the variable and can be up to 64 characters in length.

value line 1 through *value line n* specify the value of the variable. Application Manager appends the information from data columns 2 through 72 (fixed format) of each line to form the value, which can be up to 923 characters in length. If the records that contain the data have line numbers, they are ignored. You cannot define variables in terms of other variables.

Considerations for DBCS Data: Because double-byte character set (DBCS) data takes up from 2 to 4 bytes for every character, the installation may need to add place holder bytes at the end of the variable value line. Application Manager recognizes X'00' as the place holder value.

For example, to change a blank to X'00', enter on the COMMAND line:

```
c ' ' x'00'
```

Place the cursor on the character you want to change (usually between columns 2 and 72) and press the Enter key.

***LIBRARY** *ddname type*
data set 1
⋮
data set n

identifies the data set(s) that is to be allocated under the specified ddname. The order in which you specify the data set names in the subsequent records determines the concatenation order.

ddname specifies the ddname under which the data sets of the library are concatenated. *ddname* can be up to 8 characters in length.

type specifies the library type. Valid values are:

ISPF ISPF files

CLIST CLIST files

EXEC REXX files

INPUT
Any other type of input file During the process of allocation, if an INPUT type library has no current allocations, its ddname and list of data sets will be allocated. The allocation will specify SHR, which allows shared access to the library and data sets.If another application is invoked using the same INPUT library, the allocation is performed only when the list of data sets for both applications match.

When the library type is INPUT, and the data set information does not match, the second application is not invoked. An error message appears stating that the requested function cannot be invoked because the required files are in use.

OUTPUT
Any other type of output file Only one data set can be specified for the OUTPUT type library. The allocation will specify OLD, which allowsexclusive access to the library and data set. If an OUTPUT type library is currently allocated, the function cannot be invoked because the required files are in use. If OUTPUT is entered and more than one data set is specified, an error message appears.

data set 1 through *data set n* specify the data sets in the library. Each can be up to 46 characters in length. If you specify the high-level qualifier as part of the data set name, enclose the data set name in single quotation marks. If you do not enclose the data set name in quotes, the user's TSO/E prefix is used as the high-level qualifier.

```
*INIT_COMMAND
  initialization command line 1
⋮

  initialization command line n
```

identifies the panels, commands, or programs that execute before the associated function is invoked. The INIT_COMMAND uses the ISPF SELECT service to invoke one of the following:
- PANEL(panel name) OPT(option)
- CMD(command)
- PGM(program name) PARM(parameter(s))

initialization command line 1 through *initialization command line n* (where n=1, 2, or 3) form the INIT_COMMAND. Application Manager appends data from columns 2 through 72 (fixed format) or from columns 10 through 80 (variable format) to form the command, which can be up to 213 characters in length. For information about the SELECT service, see *z/OS ISPF Planning and Customizing*.

```
*INVOCATION_COMMAND
  invocation command line 1
⋮

  invocation command line n
```

identifies the panels, commands, or programs that invoke the associated function. If a function does not have an INVOCATION_COMMAND, its environment must have one. The INVOCATION_COMMAND uses the ISPF SELECT service to invoke one of the following:

- PANEL(panel name) OPT(option)
- CMD(command)
- PGM(program name) PARM(parameter(s))

invocation command line 1 through *invocation command line n* (where n=1, 2, or 3) form the INVOCATION_COMMAND. Application Manager appends the data from columns 2 through 72 (fixed format) or from columns 10 through 80 (variable format) of each line to form the command, which can be up to 213 characters in length. For information about the SELECT service, see *z/OS ISPF Services Guide*.

```
*TERM_COMMAND
  termination command line 1
:

  termination command line n
```

identifies the panels, commands, or programs that execute after the associated function terminates. The TERM_COMMAND uses the ISPF SELECT service to invoke one of the following:

- PANEL(panel name) OPT(option)
- CMD(command)
- PGM(program name) PARM(parameter(s))

termination command line 1 through *termination command line n* (where n=1, 2, or 3) form the TERM_COMMAND. Application Manager appends the data from columns 2 through 72 (fixed format) or from columns 10 through 80 (variable format) to form the command, which can be up to 213 characters in length. For information about the SELECT service, see *z/OS ISPF Services Guide*.

Example using installation files when adding a service

Assume that you want to create installation files to add a hypothetical office service to the Information Center Facility. The service prompts you for the name of a person in your company and then finds the room and floor number for the person. You want the service to have the following characteristics:

- Users can select it as option 2, on the panel, Information Center Facility - Office Services.
- All applications for the service are in English.
- Users see the following description on the menu where they select the service:
  `Find a person's room and floor number.`
- Administrators see the following description on the list of applications:
  `Locate service`
- The help text that users see when they select the DESCRIBE option on panel, Information Center Facility - Office Services, is:

```
Select this option to find a person's room and floor number.
The service prompts you for the person's last name followed
by the first name or initial.
```

- The service is defined by the function application, ABCLOC, and invoked by the command %LOCATE.
- The tutorial for the service starts with panel ABCLOC00.
- The service has only one shared variable, FRMTOFF, associated with it. The description of the variable is:
  `Room and floor number format used by office services`

The value of the variable is *room/fl*. This variable will be used by other office services that you plan to write in the future.

Required installation files

You decide to create the following installation files to install this service:

- Panel installation file to define the Information Center Facility - Office Services panel and include an option for your service.
- Function installation file to define your service and provide the data associated with it, except for the shared variable.
- Environment installation file to define the shared variable for your service.

Create the panel installation file

To create the panel installation file, you need the following information about the Information Center Facility - Office Services panel:

- Application name
- Application type
- Application language
- Application keyword
- Panel title
- Indication of whether it is a primary panel
- Model ISPF panel that it uses
- Description that the administrator sees
- Description that user sees
- Option ID, name, and language for each option
- Help text

To obtain that information, access Application Manager and do the following:

- On the first Application Manager panel, Application Manager - List of Applications, locate the TSO/E - Office Services application to find the application name (ICQOFICE), type (panel), and language (English).
- Type an `M` to the left of the application and you see the panel, Application Manager - Define a Panel. Select the following three options on the panel to obtain the specified information:
  - GENERAL INFORMATION to find the keyword (OFFICE), panel title (Information Center Facility - Office Services), indication of whether it is a primary panel (no), model ISPF panel that it uses (ICQAMED1), and the descriptions for the administrator (TSO/E - Office Services) and user (Use mail/document/other office services).
  - OPTIONS to find the option ID, keyword, and option name for each option on the panel. They are:

Option ID	Option name	Language
0	ICQDESCRIBE	English
1	ICQPSTSO	English
T	ICQTUTOR	English

  - HELPTEXT to find the help text for the panel. It is:

```
Select this option to use office services through Personal
Services/TSO, if provided.
With this service you can exchange messages and data with other
users on the same computer system or any other system in a network.
```

For a complete description of how to use Application Manager, see *z/OS TSO/E Administration*.

Combining the information about your service with the information you found about the Information Center Facility - Office Services panel, you create the following panel installation file. The information related to your service is highlighted.

```
*Application Manager INSTALLATION FILE
*SYSTEM            MVS TSO/E
*PANEL
*PANEL_NAME        ICQOFFICE
*LANGUAGE          ENGLISH
*KEYWORD           OFFICE
*PANEL_TITLE       Information Center Facility - Office
Services
*PRIMARY_PANEL     N
*MODEL_ISPF_PANEL  ICQAMED1
*ADMIN_DESC        TSO/E - Office Services
*USER_DESC_1       Use mail/document/other office services
*OPTION            0
 ICQDESCRIBE
 ENGLISH
*OPTION            1
 ICQPSTSO
 ENGLISH
*OPTION            2
 ABCLOC
 ENGLISH
*OPTION            T
 ICQTUTOR
 ENGLISH
*HELP_TEXT
 Select this option to use office services through
 Personal Services/TSO, if provided.
 With this service
 you can exchange messages and data with other users on
 the same computer system or any other system in a
 network.
```

Figure 106. Sample panel installation file

Create the function installation file

Using information about your service, you create the following function installation file.

```
*Application Manager INSTALLATION FILE
*SYSTEM MVS TSO/E
*FUNCTION
*FUNCTION_NAME     ABCLOC
*LANGUAGE          ENGLISH
*KEYWORD           LOCATE
*ENVIRONMENT_NAME  LOCENV
*ADMIN_DESC        Locate service
*USER_DESC_1       Find a person's  room and floor number
*INVOCATION_COMMAND
 CMD(%LOCATE)
*ISPTUTOR_PANEL    ABCLOC00
*HELP_TEXT
 Select this option to find a person's room and floor number.
 The service prompts you for the person's last name followed
 by the first name or initial.
```

Figure 107. Sample function installation file

Create the environment installation file

Using information about your service, you create the following environment installation file.

```
*Application Manager INSTALLATION FILE
*SYSTEM  MVS TSO/E
*ENVIRONMENT
*ENVIRONMENT_NAME   LOCENV
*LANGUAGE           ENGLISH
*ENVIRONMENT_DESC   Office environment
*VARIABLE           FRMTOFF
 Room and floor number format used by office services
 room/fl
```

Figure 108. Sample environment installation file

For information about loading and using application files, see *z/OS TSO/E Administration*.

Using invoking panel entries

You can use the INVOKING_PANEL entry to add an option to the Offices Services panel ICQOFFICE. The following sample shows an installation file for a function. This installation file automatically adds the new function PHONE as an option on the Office Services panel after the option for ABCLOC.

```
*Application Manager INSTALLATION FILE
*SYSTEM MVS TSO/E
*FUNCTION
*FUNCTION_NAME      PHONE
*LANGUAGE           ENGLISH
*KEYWORD            PHONE
*ENVIRONMENT_NAME   LOCENV
*INVOKING_PANEL     ICQOFFICE
 ENGLISH
 PHO
*AFTER              ABCLOC
 ENGLISH
*ADMIN_DESC         Phone service
*USER_DESC_1        Find a person's phone number
*INVOCATION_COMMAND
 CMD(%PHONE)
*ISPTUTOR_PANEL     PHONE00
*HELP_TEXT
 Select this option to find a person's phone number.
 The service prompts you for the person's last name followed
 by the first name or initial.
```

Figure 109. Sample function installation file using INVOKING_PANEL entry

Upgrading installation files

You can use the Application Manager upgrade function to install new versions of existing applications while preserving any customization that may have been done. The upgrade function allows you to upgrade an existing installation file with the contents of an upgrade file.

The installation file to be upgraded is usually an export of an application that may have been customized.The export function of the Application Manager copies the application definition information from an application table, formats the information into entries corresponding to an installation file, and saves these entries in a user-specified data set.

An upgrade file may be delivered with a new version of an existing application and contain entries for adding, deleting, and replacing portions of the existing installation file. The Application Manager upgrade edit macro reads this file, interprets the entries, and applies them to the installation file to be upgraded. When upgrade files are used to distribute changes to application definitions, there is less chance of having conflicts with local modifications because only the changes have to be processed against the application definitions.

If you use upgrade files, the time required to install subsequent new versions of a service will be reduced. See *z/OS TSO/E Administration* for more information.

Mass installation file processing

Mass installation file processing allows Information Center Facility administrators to process multiple installation files at one time using a batch method. This service provides two separate functions:
- Mass upgrade and install
- Mass export

Using the mass upgrade and install function, applications can be upgraded, installed, verified, and made available.

The ability to process multiple installation files simplifies the installation of ISPF/PDF dialog applications in the Information Center FacilityApplication Manager environment. For detailed information about multiple installation and export file processing, see *z/OS TSO/E Administration*.

Deleting a product or service

Except for Application Manager, you can delete any of the products or services in the Information Center Facility but do so with caution. You cannot delete Application Manager because all products and services in the Information Center Facility depend on it. Several services and one product have dependencies on other services. For example:
- COURSES depends on NEWS.
- COURSES, ENROLL, and PS/TSO depend on NAMES. Before users can use the mail facility in PS/TSO, the Information Center Facility administrator must create the master names directory.
- ENROLL depends on USERTYPES and ISPF defaults.

Therefore, removing NEWS, NAMES, or USERTYPES could cause unpredictable results.

Possible ways to delete a product or service are:
- Use Application Manager panels to delete all applications associated with the product or service and update any menu panels that contain an option for the product or service.

  One advantage of using this method is that Application Manager dynamically updates the tutorial menus associated with the functional menus that you change.
- Replace the first panel of the product or service by a dummy panel that you create. You can put the panel in the data set supplied by IBM or in an installation-supplied data set. Use any of the dummy panels supplied by IBM (for example, the PROBLEM option panel, ICQGCM07) as a model.

  A disadvantage of this method is that the tutorial remains unchanged.

Instead of deleting a product or service, you can limit its availability to selected users or groups of users. For example, if an accounting department requires certain products that general users do not need, you can define the applications for these products at the group level for that group. You can also copy down system level applications to the group or private level and then make the applications unavailable at the system level.

Note: You can make group and private Application Manager unavailable at the system level. To do this, copy down the applications for group and private Application Manager to the group level. Do not make the system level Application Manager unavailable. If you do, you will have difficulty maintaining your applications.

For information about how to use Application Manager to delete applications, change menu panels, and make applications unavailable, see *z/OS TSO/E Administration* or the on-line tutorial and help panels for Application Manager.

Changing a product or service

You can use Application Manager to make changes to a product or service by changing the applications that define the product or service. For a description of how to change applications, see *z/OS TSO/E Administration* or the on-line tutorial and help panels for Application Manager.

You can also change a product or service by changing its associated panels. Use Application Manager to change the dynamic menu panels in the Information Center Facility. Follow the ISPF rules for modifying panels when changing any other panels. For example, when you change a help or tutorial panel or a panel within a product or service, and the panel is not generated using Application Manager, ensure that it has no more than 24 lines in the)BODY section. If you try to display more than the maximum number of lines that the terminal can display, ISPF issues an error message. For information about defining the various types of panels, see *z/OS ISPF Services Guide*.

You can also change a product or service by replacing it. One way to replace a product or service is to delete the current product or service and then add the one with which you are replacing it. However, do not delete a product or service if another product or service depends on it. For information about dependencies, see "Deleting a product or service" on page 635.

When you change or replace a product or service, add applicable help and tutorial information to support it. Table 129 on page 685 shows a list of all menu and data entry panels and their associated help panels.

Creating or tailoring application definitions

Application Manager supports three levels of application definitions:
- System
- Group
- Private

This support allows departments or user groups, and individual end-users to create or tailor panels, functions, and environments for their own use. See *z/OS TSO/E Administration* for more information.

The table libraries ICQAMTAB and ICQCMTAB containthe system (highest) -level application definitions. You can create group level application definitions for different groups of users. Also, users can create a private (lowest) level of application definitions. Unique applications can be defined at the lower levels or existing applications can be copied down from higher levels and then customized.

You can limit access to selected applicationsto specific group(s) or private users. To control access, copy down the selected applications to specific group or private levels and then delete the options from the system-level User Services panel.

Invoking an application

If you define an application using Application Manager, you should also use Application Manager to invoke the application. Then at invocation, Application Manager allocates the libraries that the application requires.

Using Application Manager to define and invoke an application eliminates the requirement that all libraries must be included in the LOGON procedure. Decreasing the number of libraries allocated in the LOGON procedure decreases the time required for LOGON. Allocating libraries as an application requires them, enables an installation to make different versions of an application available.

If you use Application Manager to define an application, but do not use Application Manager to invoke the application, the libraries the application requires may not be available. Specifically, if the invocation you use establishes a new ISPF environment, the libraries that were allocated before the invocation will not be available. Examples of how a new ISPF environment can be established are:
- You use ISPF SELECT with the NEWAPPL parameter.
- You split the screen and cause an application to run on a different logical screen.

If you define an application using Application Manager, but cannot invoke it through Application Manager, and if the application establishes a new ISPF environment, then you must allocate the required libraries in the LOGON procedure or before starting ISPF.

Multiple level applications

Up to three levels of application definition libraries can be active during application invocation, as follows:
- The system level library is always active.
- The group level library can be active.
- The private level library can be active.

The order from private to system is termed lower to higher, respectively. If the application definitions have the same name and language, lower-level applications supersede higher level applications. See “Modifying ICF start-up and termination processing” on page 639.

System application definitions are allocated to the files ICQAMTAB and ICQCMTAB.

A single group's application definitions are stored in two data sets as follows:

```
<group prefix>.<group name>.ICQAGTAB  (administrator)
<group prefix>.<group name>.ICQRGTAB  (user)
```

<group prefix>
Is a customizable constant set in the non-display panel ICQSIEAM.

<group name>
Is the name that you entered on the Group Specification panel.

Group library allocation depends on whether you used the Group Specification panel to select a particular group to be used during application invocation. If you specified a group name, it is retrieved from the user's ISPF.PROFILE member ICQGROUPand combined to form the group data set name:

```
<group identifier>.ICQRGTAB
```

<group identifier>
Contains the system group prefix (which you can customize in the non-display panel ICQSIEAM) that has been concatenated with the group name specified on the Group Specification panel.

The string ICQRGTAB identifies the group user data set. If the above data set does not exist or cannot be allocated, an informational message appears on the screen indicating that no allocation can take place. However, application invocation will continue. If the group data set exists, it is allocated to the file ICQRGTAB and used during application invocation. Any previous allocation for the file ICQRGTAB is freed. See *z/OS TSO/E Administration* for additional information.

A single user's private application definitions are stored in a single data set as follows:

```
<TSO/E prefix>.ICQRPTAB
```

<TSO/E prefix>
Defaults to the USERID or it can be set by the user.

The string ICQRPTAB identifies the private user data set. If the data set does not exist or cannot be allocated, an informational message appears on the screen indicating that no allocation can take place. However, application invocation will continue. If the private data set exists, it is allocated to the file ICQRPTAB and used during application invocation. Any previous allocation for the file ICQRPTAB is freed. See *z/OS TSO/E Administration* for additional information.

Application search order

When an application is invoked using the application name (keyword APPLNAME), the system first searches each active library for matching names or keywords using the application name and session language. The order of searching is always from private to system. If the application is not found, the system then searches each active library using the application name and default language. If the application is still not found, the system searches each active library using only the application name. When the application is found, it is invoked. Finding the application with the desired language takes precedence over finding the application at the lowest level.

When an application is invoked by table name (keyword TABLE), the application table is invoked regardless of level. The fourth character of the table name indicates the library level.

When an option is selected on an Application Manager panel, the search order for finding the application is the same as when you use the APPLNAME keyword. In some circumstances the TABLE keyword parameter is used internally to increase performance, but it appears to the user as if the lowest-level application name (respective to language) has been invoked. When an application is invoked by keyword (keyword KEYWORD), the processing is similar to using APPLNAME. The system searches for a match to the keyword from the lowest to the highest level. If more than one application matches the keyword, the Applications Matching Keyword Panel (ICQAME71) is displayed. This panel displays one row for each unique application name and language, with lower-level applications superseding higher-level applications with the same name, language, and keyword.

A switch to alter the search process is available in ICQSIEAM. See "Changing variables for Application Manager" on page 608.

Performance impact

The best performance occurs when you use only system level libraries. Application selection time increases if you use group and private libraries. However, application selection time is impacted the most when invocation crosses from one application level to another.

To minimize the performance impact, lower-level administrators should copy clusters of applications down to their level. For example, if you copy two or more applications, which are attached to one higher panel, down to a lower level, you should also copy that panel down to a lower level. If you copy several functions that use the same higher-level environment down to a lower level, you should also copy down the environment.

Hierarchy display

You can display the hierarchy of applications that can be reached from a selected panel application. The Hierarchy Display option (H) is available on the List of Applications panel and the Where Used panel. The hierarchy is presented in list form, and represents the applications as they are invoked at run-time. To display a hierarchy, the panel selected must have options and must be verified. For a higher-level panel to be listed in a hierarchy, it must be available.

Modifying ICF start-up and termination processing

You can modify the CLISTs and applications that perform start-up and termination processing to include the initial or final processing that your installation requires. The CLISTs supplied by IBM, ICQICF and ICQGCC10,perform start-up processing, and the CLIST supplied by IBM, ICQGCC11,performs termination processing.

Figure 110 shows the invocation syntax of the ICQICF CLIST.

```
%ICQICF OLDAPPL(old_application_ID)  +
        NEWAPPL(new_application_ID)  +
        APPLNAME(application_name)  +
        PROFILE(profile_name)  +
        PRIVATE(Y/N)  +
        EXITPROC(Y/N)  +
        DEBUG(Y/N)  +
        INIT("command_string")
```

Figure 110. ICQICF Invocation Syntax

A description of each of the parameters follows.

OLDAPPL(*old_application_ID***)**
: specifies the application ID previously used for ISPF. The default is ISR.

NEWAPPL(*new_application_ID***)**
: specifies the application ID to be used for the Information Center Facility. *new_application_ID* consists of 1-4 characters, the first of which must be

alphabetic (A-Z) and the remaining characters are any combination of alphabetic (A-Z), numeric (0-9), or one of the special characters ($,#,@). The default is ICQ. If you do not use ICQ, you must change the variable QAEAPPL in the ENROLL CLIST, ICQAEC00.

If you change the application ID to a value that has not been used before, you must define new Information Center Facility system defaults or the ISPF base profile will be used during enrollment.

APPLNAME(*application name*)
specifies the name of the first application that is to be invoked. The default is ICQUSER, (TSO/E - User's Services). Use ICQADMIN to invoke the first application for the administrator.

PROFILE(*profile_name*)
specifies the name of the ISPF profile data set. The default is ISPF.PROFILE. If you do not use ISPF.PROFILE, you must change variable QAEPROF in the ENROLL CLIST, ICQAEC00.

PRIVATE(Y/N)
specifies whether the private Application Manager library is to be used. N indicates that the private Application Manager library (file ICQRPTAB) will not be allocated. Y indicates ICQICF will attempt to find and allocate the private library of the form:

```
<TSO/E prefix>.ICQRPTAB
```

The default is Y. The suffix ICQRPTAB indicates the private table library.

EXITPROC(Y/N)
indicates whether ICQGCC11, the termination CLIST, is to be invoked when ISPF terminates. Y indicates that ICQGCC11 is invoked when ISPF terminates; N indicates that it is not. The default is N.

DEBUG(Y/N)
indicates whether tracing is to be done. Y indicates that tracing for ICQICF equivalent to TRACE3 is to be done. For information about TRACE3, see “TRACE3 command — level 3” on page 695. N indicates that tracing is not to be done. The default is N.

INIT("*command_string*")
specifies the command string used to initialize the first application invoked. If the first application specifies a start-up function, the command specified by INIT is processed before that start-up function. INIT is useful for specifying initialization that you require only once per ISPF session. *command_string* must start and end with two single quotation marks and contain a maximum of 256 characters. It uses the format of ISPEXEC SELECT. For information about ISPEXEC SELECT, see *z/OS ISPF Services Guide*. If the command string uses the command interface (as is the case for a CLIST), it requires an additional set of quotes. Therefore, a CLIST requires three sets of single quotation marks. The default is CLIST ICQGCC10.

Modifying start-up processing

CLIST ICQICF starts ISPF and invokes the first application for the user. As shipped by IBM, ICQICF invokes ICQUSER, the TSO/E - Use Services application. Before displaying the primary user menu, ICQICF invokes ICQGCC10, which theninvokes the news service, CLIST ICQCNC10, to display any "new" news.

You can modify start-up processing by specifying a command string using the ICQICF INIT parameter.

Another way to modify start-up processing is by providing a start-up function in the first application that ICQICF invokes. That start-up function is performed before the processing is done for the application. You can use Application Manager panels to add start-up functions to applications. For information about how to use Application Manager, see *z/OS TSO/E Administration*.

You can also add calls to other CLISTs, programs, or ISPF dialogs to ICQGCC10, to include other processing that your installation needs for start-up. If you add calls to other routines, return all message IDs to ICQGCC10, and process the messages in ICQGCC10 to display a combined message or show all the messages on one panel.

Modifying termination processing

As shipped by IBM, the termination CLIST, ICQGCC11, contains no trace processing. If your installation requires termination processing, add the necessary code to ICQGCC11. Then you can either:

- Change the parameter EXITPROC(N) on the PROC statement of CLIST ICQICF to EXITPROC(Y).
- Add the parameter EXITPROC(Y) on the EXEC statement in the LOGON procedures for the users and administrators, as shown in the following examples:

  For administrators:

  ```
  //ICQAPROC EXEC PGM=IKJEFT01,REGION=4096K,DYNAMNBR=40,
  //         PARM='ICQICF APPLNAME(ICQADMIN)
  EXITPROC(Y)'
  ```

  For users:

  ```
  //ICQPROC  EXEC PGM=IKJEFT01,REGION=4096K,DYNAMNBR=40,
  //         PARM='ICQICF APPLNAME(ICQUSER) EXITPROC(Y)'
  ```

ICQGCC11 is invoked after both the Information Center Facility session and the ISPF session have ended.

Adding commands to the command table

To extend the function of the Information Center Facilityyou can add your own commands to the command table, ICQCMDS. For example, you can add commands to enter dialogs, to act as aliases of other commands, or to execute CLISTs. After you have added the new commands, you can enter them on the COMMAND or OPTION line of any panel in the Information Center Facility.

Each entry in ICQCMDS consists of four fields: the command, the truncation permitted, the action the command performs, and a description of the command.IBM ships the following commands in ICQCMDS:

Command	Purpose
ADMIN	To display the primary administrator menu panel.
IC	To display the primary user menu panel.
GO	To go to a specific option, type GO followed by the keyword for the option.

If ICQCMDS is not in use, you can add commands to it using option 3.9 in ISPF/PDF. If ICQCMDS is in use, you cannot use option 3.9, but must copy

ICQCMDS to a data set with a different name, modify the copy, and replace the original. For more information about adding commands to a command table, see *z/OS ISPF Services Guide*.

Note: Before modifying the command table, consider that commands execute before keywords. If, for example, you defined a command NEWS to display reminders, you would not be able to enter the news function by typing NEWS.

Resynchronizing the enrollment default profiles

To automatically pick up the latest level of the ISPF/PDF profile tables, delete the members ISPFSPROF and ICQZPROF from the data set "ICQ.ICQTLIB". This data set is defined in the Application Manager function ICQISPFDEF. In addition, if you have customized the Information Center Facility system defaults, you should use the SYSDEF option to re-customize the default profile for enrolled users.

Writing an exit for ADRS

Functional description

If you install ADRS, users can access it from the Information Center Facility panel, DATA ANALYSIS/REPORT CREATION SERVICES. Whenever a user selects the ADRS option, the system invokes VS APL, which loads ADRS.

You can provide an exit for ADRS to perform various functions. The exit receives control before the system invokes VS APL.The exit can:
- Display your own panels
- Allocate data sets. For example, the exit can pre-allocate the report data set.

TSO/E-supplied exit

TSO/E does not provide a default exit routine for ADRS.

Return specifications

The exit returns one of the following return codes:

Return code (decimal)	Description
0	The Information Center Facility invokes VS APL, which loads the ADRS workspace. The system then displays the ADRS-II panel.
4	The Information Center Facility does not invoke the ADRS option. It redisplays the DATA ANALYSIS/REPORT CREATION SERVICES panel.

Programming considerations

The exit can return a return code of either 0 or 4. For information about the return codes, see "Return specifications."

Installing the exit

There are no required naming conventions for the exit. To supply the ADRS exit, edit the non-display panel ICQSIECR and locate the variable QCREXIT. By default, the variable is set to null. Set variable QCREXIT to the name of your exit.

Writing an exit for the ICF names service

Functional description

You can write an exit for the names service to keep track of changes that users or Information Center Facility administrators make in the private or master directories. The exit receives control whenever a user or Information Center Facility administrator adds, modifies, or deletes a name or group in a private directory or in the master directory. The exit receives control after the names service has successfully updated the directory.

You can use the names service exit to:

- Maintain a parallel data base that contains names such as a SCRIPT/VS names macro library. When changes are made to the Information Center Facility names directories, the exit can update your parallel names data base.
- Keep records of the changes that are made to the master names directory. The exit can record these changes in a separate data set. You can then periodically print the data set.

TSO/E-supplied exit

TSO/E does not provide a default exit routine for the Information Center Facility names service.

Programming considerations

The exit can access information that the names service places in a temporary table. To retrieve the name of the table and the information in it, include the following statements in the exit:

```
ISPEXEC VGET (QCAEXITT) SHARED
ISPEXEC TBGET &QCAEXITT SAVENAME(varname)
```

After the two statements execute, *varname* contains one of the following:

- Nulls if the information in the table is for an individual name
- Contains a list of extension variables if the information in the table is for a group. The extension variables are the IDs for the group entries.

After the exit retrieves the name of the table and the information in the table, it can use different variables in its processing. For a list of the variables and their contents, see “Variable descriptions.”

Installing the exit

There are no required naming conventions for the exit. To supply the names service exit, edit the non-display panel ICQSIECA and locate the variable QCAUEXIT. By default, the variable is blank. Set variable QCAUEXIT to the invocation string for the exit.

Variable descriptions

After the exit retrieves the name of the table and the information in the table, it can use the following variables. The variables that are marked with an asterisk (*) can be used to determine what type of processing to perform. They contain information about the update just performed, rather than information about the entry itself that was changed.

Variable	Contains:
QAAADDR	The first line of the internal address.
QAAADDR2	The second line of the internal address.
QAAADMIN*	• "YES" if the exit is invoked from the administrator dialog • "NO" if the exit is invoked from the user dialog.
QAADISP	The display form of the name, which is in the form *lastname, firstname m.*.
QAADNAME	The department name.
QAADNUM	The department number.
QAAFRST	The first name.
QAAID	A string identifier that the user supplies. It can be up to seven characters long. The identifier must be unique in the directory that contains the entry.
QAAIND	• "#" if the entry is either a master directory entry or a private directory entry that is a modified version of a master directory entry • "@" if the entry is a private directory entry that is not a modified version of a master directory entry.
QAALAST	The last name or group name.
QAALIB*	The name of the library (the ddname) that contains the table that was just updated.
QAAMIDLE	The middle name.
QAANICK	The nickname.
QAANODE	The system node.
QAANODE2	A second system node.
QAANSEL	• The selection character the user entered to select the entry, if the user selected this entry from a list • A blank, if the user did not use a list to select this entry.
QAANTITL	The name title, for example, Mr., Mrs., or Ms.
QAAPHONE	The phone number.
QAAPRIV	• ">" if the entry is for or from a private directory • A blank if the entry is for or from the master directory.
QAARTYPE*	The type of update that was performed (add, delete, or modify).
QAASDNAM	The search form of QAADNAME (department name). All of the letters are capitalized.
QAASDNUM	The search form of QAADNUM (department number). All of the letters are capitalized.
QAASFLAG	A search flag. The flag is on if the user entered partial information to see a subset of the directory. This entry was a member of the subset that was displayed.
QAASFRST	The search form of QAAFRST (first name). All of the letters are capitalized.
QAASLAST	The search form of QAALAST (last name or group name). All of the letters are capitalized.
QAASNICK	The search form of QAANICK (nickname). All of the letters are capitalized.

Variable	Contains:
QAASTITL	The search form of QAATITLE (job title). All of the letters are capitalized.
QAASUFIX	The name suffix, for example, Jr., Sr., or IV.
QAASUTYP	The search form of QAAUTYPE (user type). All of the letters are capitalized.
QAATITLE	The job title.
QAATABLE*	The name of the table that was just updated.
QAATYPE*	The type of entry (name or group).
QAAUSER	The system user ID.
QAAUSER2	A second system user ID.
QAAUTYPE	The user type.
QAAXADR1	The first line of the external address.
QAAXADR2	The second line of the external address.
QAAXADR3	The third line of the external address.
QAAXADR4	The fourth line of the external address.

Writing exits for Application Manager

Functional description

TSO/E provides several exits you can use to customize functions and panels that are invoked by the Application Manager.The exits for Application Manager are:

- **Function pre-initialization**

  The function pre-initialization exit receives control *before* the initialization command for the function, if one is specified, is invoked. This exit receives control regardless of whether an initialization command is specified.
- **Function post-termination**

  The function post-termination exit receives control *after* the termination command for the function, if one is specified, is invoked. This exit receives control regardless of whether a termination command is specified.
- **Panel pre-display**

  The panel pre-display exit receives control *before* a panel is displayed to the terminal user.
- **Panel post-display**

  The panel post-display exit receives control *after* a panel is displayed and the terminal user presses the Enter key.

Some ways you can use the function pre-initialization and post-termination exits include:

- Checking that the user is authorized to use an application in the function pre-initialization exit
- Allocating data sets needed by an application in the function pre-initialization exit, and freeing those data sets in the function post-termination exit
- Accumulating accounting information associated with the user in the pre-initialization exit, and then summarizing this information in the post-termination exit.

You can use the panel pre-display exit to provide default values for the panel that is to be displayed. You can use the panel post-display exit to verify that data the user entered on the panel is correct.

TSO/E-supplied exits

TSO/E does not provide default exit routines for any of the Application Manager exits.

Entry specifications

The contents of the registers on entry for all of the Application Manager exits are:

Register 0
Unpredictable

Register 1
Address of the parameter list

Registers 2–12
Unpredictable

Register 13
Address of a register save area

Register 14
Return address

Register 15
Exit entry point address

Parameter descriptions

Information about the parameters that each Application Manager exit receives is described in the following topics. For information about the standard exit parameter list and the parameter entry keys, see “TSO/E standard exit parameter list” on page 32.

The same exit-to-exit communication word is passed to both the function pre-initialization exit and the function post-termination exit. Similarly, the same exit-to-exit communication word is passed to both the panel pre-display exit and the panel post-display exit.

Function pre-initialization exit

The function pre-initialization exit receives the standard exit parameter list. For a description of this parameter list, see “TSO/E standard exit parameter list” on page 32. Figure 111 on page 647 shows the exit-dependent data that the exit receives beginning at offset +36 in the parameter list. Each parameter entry is described following the figure.

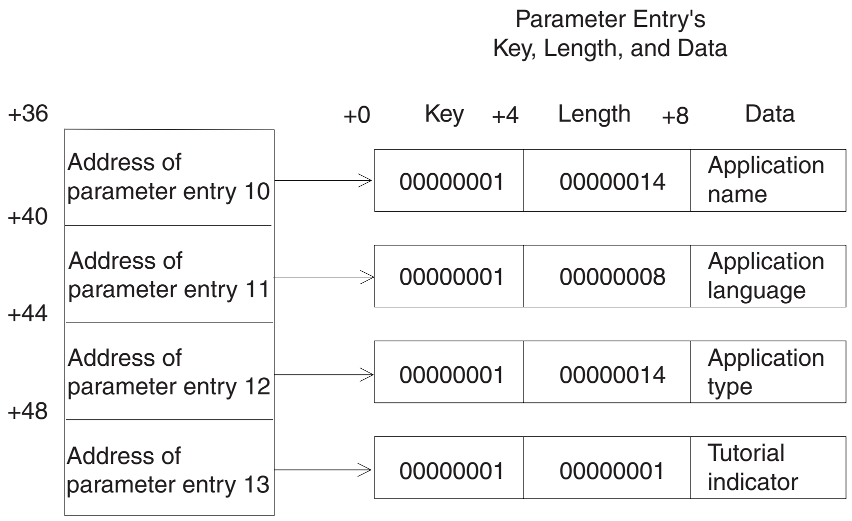


Figure 111. Exit-rependent data for the Application Manager function pre-Initialization exit

Application Name (Parameter Entry 10)
indicates the name of the application that is to be displayed.

Application Language (Parameter Entry 11)
indicates the language used in the application.

Application Type (Parameter Entry 12)
indicates the type of application. This parameter contains the value "FUNCTION".

Tutorial Indicator (Parameter Entry 13)
indicates whether the tutorial for the application is to be invoked. This parameter contains either the value "Y" or "N".

Function post-termination exit

The function post-termination exit receives the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. Figure 112 on page 648 shows the exit-dependent data that the exit receives beginning at offset +36 in the parameter list. Each parameter entry is described following the figure.

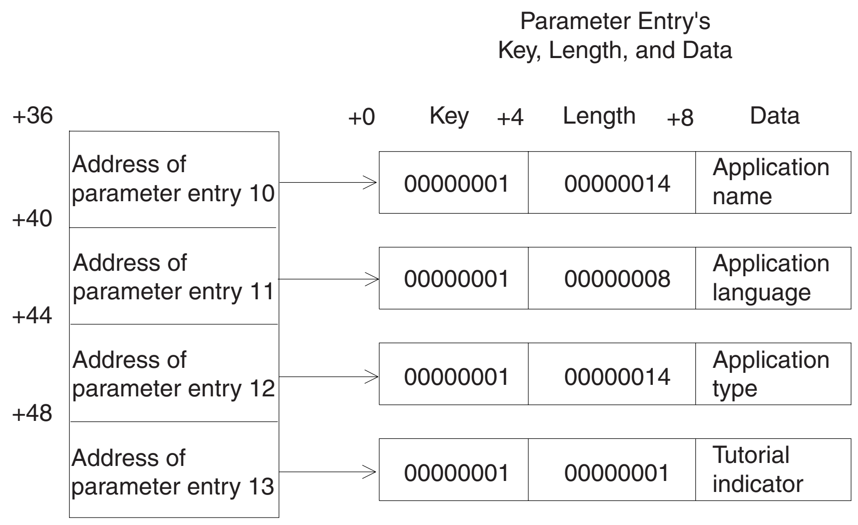


Figure 112. Exit-dependent data for the Application Manager function post-termination exit

`Application Name (Parameter Entry 10)`
: indicates the name of the application that has been displayed.

`Application Language (Parameter Entry 11)`
: indicates the language used in the application.

`Application Type (Parameter Entry 12)`
: indicates the type of application. This parameter contains the value "FUNCTION".

`Tutorial Indicator (Parameter Entry 13)`
: indicates whether the tutorial for the application has been invoked. This parameter contains either the value "Y" or "N".

Panel pre-display exit

The panel pre-display exit receives the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. Figure 113 shows the exit-dependent data that the exit receives beginning at offset +36 in the parameter list. Each parameter entry is described following the figure.

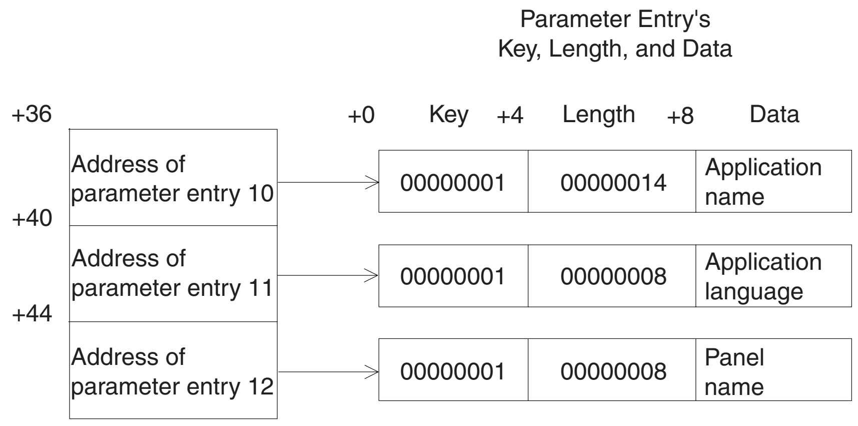


Figure 113. Exit-dependent data for the Application Manager panel pre-display exit

Application Name (Parameter Entry 10)
indicates the name of the application currently being displayed.

Application Language (Parameter Entry 11)
indicates the language used in the application.

Panel Name (Parameter Entry 12)
indicates the name of the panel that is to be displayed.

Panel post-display exit

The panel post-display exit receives the standard exit parameter list. For a description of this parameter list, see "TSO/E standard exit parameter list" on page 32. Figure 114 shows the exit-dependent data that the exit receives beginning at offset +36 in the parameter list. Each parameter entry is described following the figure.

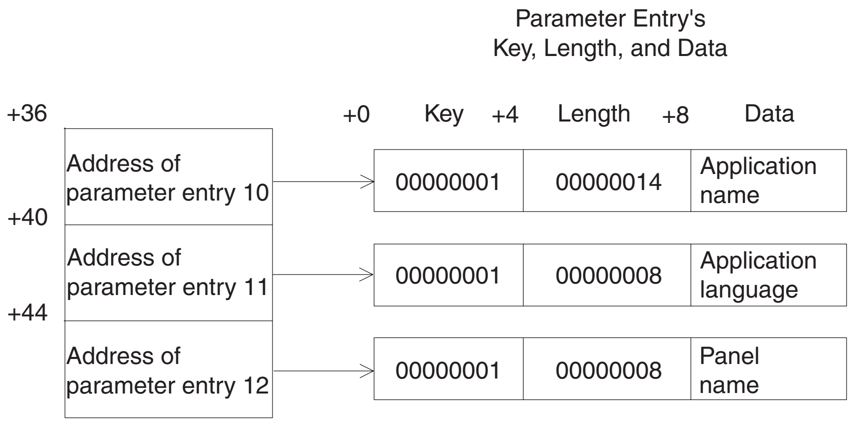


Figure 114. Exit-dependent data for the Application Manager post-display exit

Application Name (Parameter Entry 10)
indicates the name of the application currently being displayed.

Application Language (Parameter Entry 11)
indicates the language used in the application.

Panel Name (Parameter Entry 12)
indicates the name of the panel that has been displayed.

Return specifications

The contents of the registers on return for all of the Application Manager exits must be:

Registers 0–14
Same as on entry

Register 15
Return code

Return codes

Table 128 on page 650 shows the return codes that all of the Application Manager exits support.

Table 128. Return codes that all Application Manager exits support

Return code (decimal)	Description
0	Successful completion. The Application Manager continues processing the current application.
12	Error occurred in the exit. Application Manager terminates the current application. Application Manager issues an error message if when the application was selected from a panel. Otherwise, the caller of the Application Manager is responsible for analyzing the return code and taking the appropriate action. If a failing exit sets a reason code with a key value of X'03', this reason code is used as the return code from the Application Manager. (See the note.) Otherwise, • For failing pre-initialization and pre-display exits, Application Manager sets a return code of 6 • For failing post-termination and post-display exits, the Application Manager sets a return code of zero, or whatever is appropriate.
16	Error occurred in the exit. Application Manager terminates the current application without issuing an error message. If a failing exit sets a reason code with a key value of X'03', this reason code is used as the return code from the Application Manager. (See the note.) If your exit sets a return code of 16, you should consider displaying an informational message to the user. You can use the PUTLINE service routine to issue an informational message. See *z/OS TSO/E Programming Services* for more information.

Note: For return codes 12 and 16, the Application Manager uses the exit's reason code as the return code from Application Manager only if the code is *not* the same as one of Application Manager's return codes. Therefore, Application Manager exits should set unique reason codes. Return codes from the Application Manager are described in the chapter about invoking an Information Center Facility application in *z/OS TSO/E Programming Services*.

Programming considerations

The exits must follow standard linkage conventions. They must save the registers on entry and restore the registers when they return.

The exits must obtain any storage that they need and should free it when it is no longer needed. If an exit returns an undefined return code, Application Manager terminates the current application without issuing an error message to the user.

Do not use the Application Manager exits to set up a new command buffer, because Application Manager ignores the new command buffer.

Environment

The attributes for each of the Application Manager exits are:
- State: Problem program
- Key: 8
- AMODE(31), RMODE(ANY)

Installing the exits

You must name each of the exits as follows:

Function pre-initialization
ICQAMFX1

Function post-termination
ICQAMFX2

Panel pre-display
ICQAMPX1

Panel post-display
ICQAMPX2

Link-edit each exit as a separate load module. You can link-edit the exits in a separate load library that is exclusively for TSO/E exits or in an existing library containing other routines. The exits can reside in:
- The link pack area (LPA)
- LNKLST
- A private STEPLIB

For more information about using the LPA, LNKLST, or STEPLIB, see "Installing the standard-format exits" on page 39.

Using the ICF naming conventions

The Information Center Facility CLIST, REXX exec, panel, message, table, and skeleton names follow specific conventions. A name following these conventions not only identifies one particular element, but also the type of element (for example, a panel, message, CLIST, or REXX exec) and the part of the Information Center Facility to which the element belongs.

You can use these naming conventions to:
- Help identify elements of the Information Center Facilityto simplify customization.
- Name installation-written panels to make naming more consistent at your installation.

To use the naming conventions to identify the elements in the Information Center Facility that belong to a particular function, or to use the naming conventions to name elements you add to the Information Center Facility consistently, refer to the sections describing the naming conventions.

To use the naming conventions to identify a particular element, first see Figure 115 on page 652 to find what type of element it is. Then see the section describing the naming convention for that type of element.

Figure 115 on page 652 shows a simple way to identify which naming convention to use to identify an element. Check the characters in the name of the element as shown, from the top of the figure to the bottom. The first match indicates the type of element to which the name belongs. For example, an element with the name ICQ001AB is a help panel for a message, not a message.

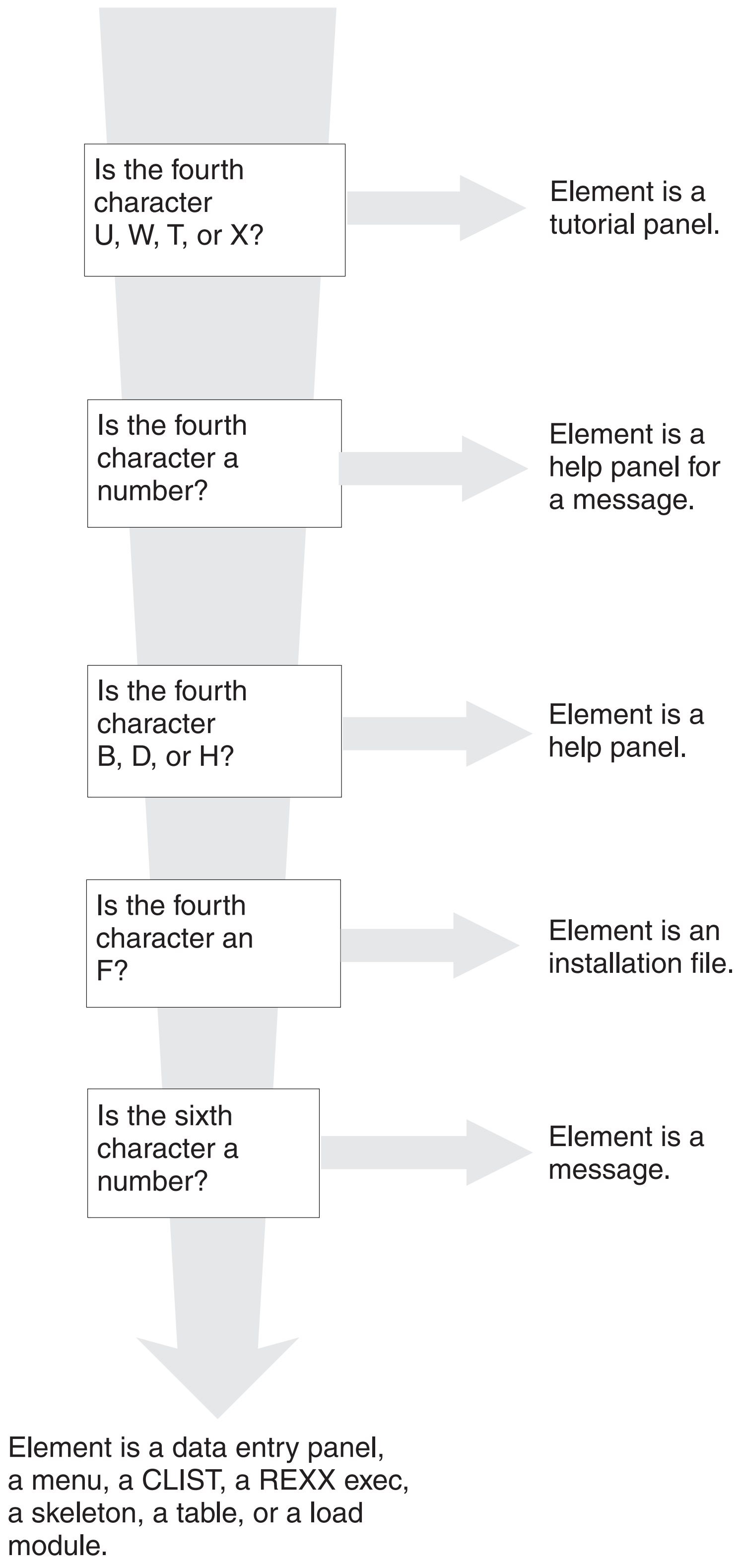

Is the fourth character U, W, T, or X?
Element is a tutorial panel.
Is the fourth character a number?
Element is a help panel for a message.
Is the fourth character B, D, or H?
Element is a help panel.
Is the fourth character an F?
Element is an installation file.
Is the sixth character a number?
Element is a message.
Element is a data entry panel, a menu, a CLIST, a REXX exec, a skeleton, a table, or a load module.

Figure 115. Identifying types of elements in the Information Center Facility

Naming conventions: applications, panels, CLISTs, REXX execs, skeletons, tables, and load modules

In the Information Center Facility, data entry panels, menus, CLISTs, REXX execs, skeletons, tables, and load modules are all named as follows:

- The first three characters of the name are the letters ICQ.
- The fourth and fifth characters identify the component to which the element belongs:

Characters	**Component**
AA	Administrator names
AB	Administrator courses
AD	User types
AE	Enrollment
AI	ISPF defaults
AM	Application Manager
AN	Administrator news
AP	Administrator printer support
AS	APPC/MVS administration dialog
CA	User names
CB	User courses
CG	Chart creation
CI	An interface for ALPDI II
CL	Interfaces for VS APL and APL2
CN	User news
CP	User printer support
CR	An interface for ADRS
GA	General administrator panels
GC	General user services, including: – Placeholder panels for products and services – Panels, CLISTs, and REXX execs that are common to two or more user functions (for example, CLISTs common to both ADRS and APLDI).
SI	Non-display panels, used to set variables common to two or more CLISTs
SP	Space management

- The sixth character distinguishes between CLISTs, REXX execs, data entry panels, menus, skeletons, and tables. The characters are:

Character	**Type**
C	CLIST
E	Data entry panel
J	Skeleton

L	Load module
M	Menu
R	REXX exec
T	Table

- The seventh, and eighth characters uniquely identify the CLIST, REXX exec, panel, skeleton, or table.

Naming conventions: application definition tables and installation files

In the Information Center Facility, application definition tables and installation files are all named as follows:

- The first three characters of the name are the letters ICQ.
- For application definition tables:
  - The fourth character defines the level of the table and the fifth character identifies the type of application:

Character	**Component**
ME	A table containing data for an environment defined using system level Application Manager
MF	A table containing data for a function defined using system level Application Manager
MP	A table containing data for a menu panel defined using system level Application Manager
GE	A table containing data for an environment defined using group level Application Manager
GF	A table containing data for a function defined using group level Application Manager
GP	A table containing data for a menu panel defined using group level Application Manager
PE	A table containing data for an environment defined using private level Application Manager
PF	A table containing data for a function defined using private level Application Manager
PP	A table containing data for a menu panel defined using private level Application Manager

  - The sixth, seventh, and eighth characters uniquely identify the application definition table.
- For installation files:
  - The fourth character is F.
  - The fifth character identifies the type of installation file:

Character	**Type**
E	Environment
F	Function
P	Panel

– The sixth, seventh, and eighth characters uniquely identify the installation file.

Naming conventions: tutorial panels

The Information Center Facility tutorial panels are named as follows:

- The first three characters in the name are the letters ICQ.
- The fourth character can be one of four letters: T, X, U, or W. This letter indicates whether the panel is a member of a tutorial for administrators or users, and whether the tutorial describes a function of the Information Center Facility. The letters are used as follows:

Letter	**Used for:**
T	Administrator tutorial panel in a tutorial for a function (for example, ICQTN for administrator News)
X	Administrator tutorial panel in a tutorial not describing a function (for example, ICQX90 for the problem option)
U	User tutorial panel in a tutorial for a function (for example, ICQUB for user Courses)
W	User tutorial panel in a tutorial not describing a function (for example, ICQW0 for the user introduction to the tutorial)

All Information Center Facility tutorial panel names are at least four characters long. Additional characters are added to indicate in what tutorial the panel is used, and where in the tutorial it is displayed.

Figure 116 demonstrates the naming conventions for administrator tutorial panels in tutorials that do not describe a function.

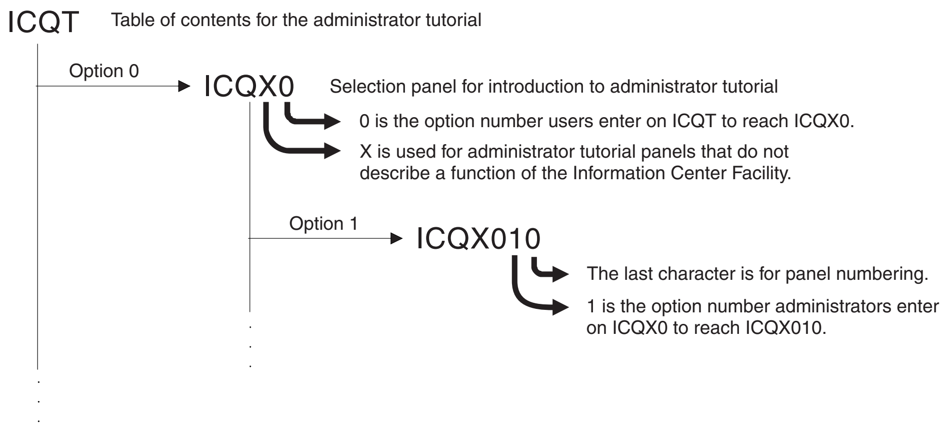

ICQT
Table of contents for the administrator tutorial
Option 0
ICQX0
Selection panel for introduction to administrator tutorial
0 is the option number users enter on ICQT to reach ICQX0.
X is used for administrator tutorial panels that do not describe a function of the Information Center Facility.
Option 1
ICQX010
The last character is for panel numbering.
1 is the option number administrators enter on ICQX0 to reach ICQX010.

Figure 116. Naming conventions: administrator tutorial panels not describing a Function

Figure 117 on page 656 demonstrates the naming conventions for user tutorial panels in tutorials that do not describe a function.

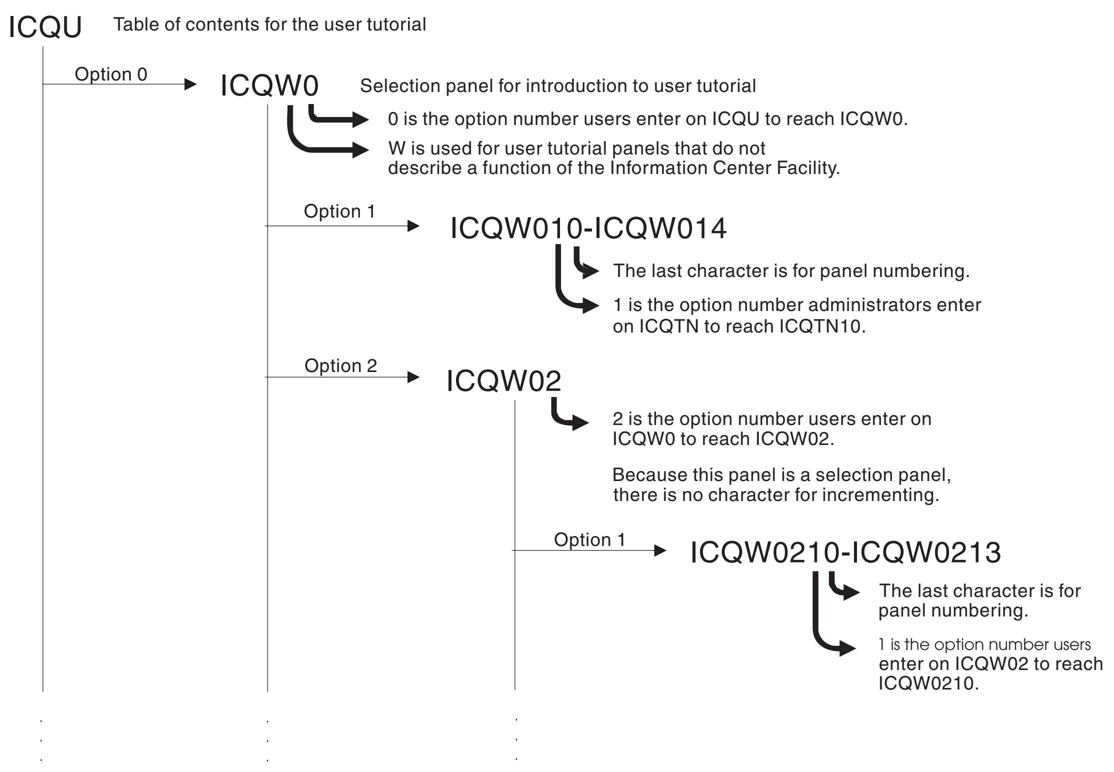

ICQU
Table of contents for the user tutorial
Option 0
ICQW0
Selection panel for introduction to user tutorial
0 is the option number users enter on ICQU to reach ICQW0.
W is used for user tutorial panels that do not describe a function of the Information Center Facility.
Option 1
ICQW010-ICQW014
The last character is for panel numbering.
1 is the option number administrators enter on ICQTN to reach ICQTN10.
Option 2
ICQW02
2 is the option number users enter on ICQW0 to reach ICQW02.
Because this panel is a selection panel, there is no character for incrementing.
Option 1
ICQW0210-ICQW0213
The last character is for panel numbering.
1 is the option number users enter on ICQW02 to reach ICQW0210.

Figure 117. Naming conventions: user tutorial panels not describing a function

If a tutorial is for a function of the Information Center Facility, the fourth and fifth, and in some cases the sixth, characters of the names of the panels in the tutorial indicate the function the tutorial describes:

Characters	Function
TA	Administrator names
TB	Administrator courses
TD	User types
TE	Enrollment
TI	ISPF defaults
TM	Application Manager
TN	Administrator news
TP	Administrator printer support, user hard copy support
UA	User names
UB	User courses
UG	Chart creation
UI	An interface for ALPDI II
ULV	An interface for VS APL
ULA	An interface for APL2
UN	User news

UR An interface for ADRS

Figure 118 demonstrates the naming conventions for administrator tutorial panels in tutorials that describe functions.

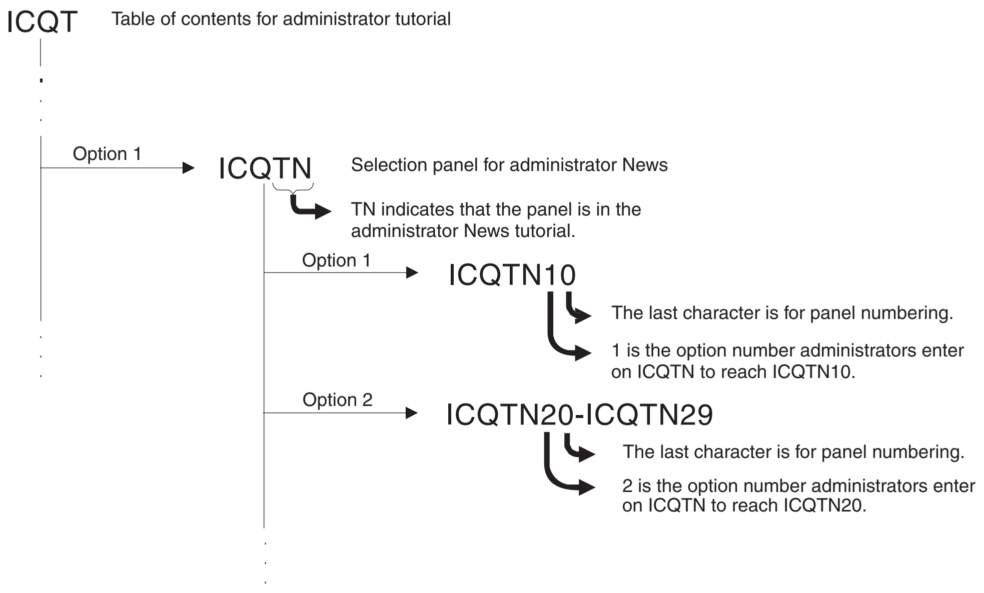

ICQT
Table of contents for administrator tutorial
Option 1
ICQTN
Selection panel for administrator News
TN indicates that the panel is in the administrator News tutorial.
Option 1
ICQTN10
The last character is for panel numbering.
1 is the option number administrators enter on ICQTN to reach ICQTN10.
Option 2
ICQTN20-ICQTN29
The last character is for panel numbering.
2 is the option number administrators enter on ICQTN to reach ICQTN20.

Figure 118. Naming conventions: administrator tutorial panels describing a function

Figure 119 demonstrates the naming conventions for user tutorial panels in tutorials that describe functions.

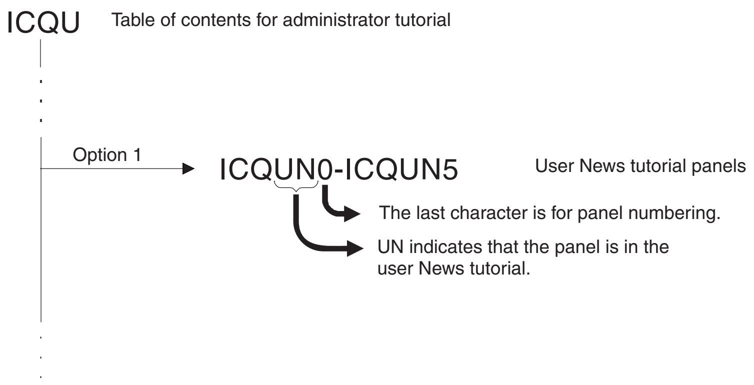

ICQU
Table of contents for administrator tutorial
Option 1
ICQUN0-ICQUN5
User News tutorial panels
The last character is for panel numbering.
UN indicates that the panel is in the user News tutorial.

Figure 119. Naming Conventions: User Tutorial Panels Describing a Function

Naming conventions: help for data entry panels and menus

The name of a panel providing help for a functional panel is based on the name of the functional panel it explains. Figure 120 on page 658 shows how help panels for menus and data entry panels are named.

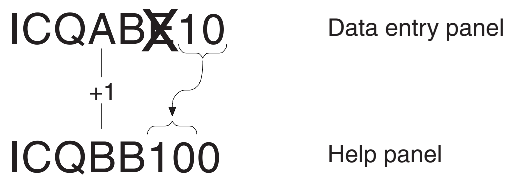


Figure 120. Naming conventions: help panels for data entry panels and menus

The fourth character of the help panel name is one greater than the fourth character of the data entry panel or menu name, so that, for example, G becomes H and C becomes D. The sixth character in the name of the menu or data entry panel is not used. An eighth character is added to the help panel name for incrementing.

Naming conventions: messages

Messages are named by function:

- The first three characters are ICQ
- The fourth and fifth characters indicate the function to which the message belongs:

Characters	**Component**
AB	Administrator courses
AD	User types
AE	Enrollment
AI	ISPF defaults
AM	Application Manager
AN	Administrator news
AP	Administrator printer support
AS	APPC/MVS administration dialog
CA	Names (All names messages use CA, not only user names messages.)
CB	User courses
CG	Chart creation
CI	An interface for ALPDI-II
CL	Interfaces for VS APL and APL2
CN	User news
CP	User printer support
CR	An interface for ADRS
GA	General administrator messages
GC	General user messages
SP	Space management

- The last three characters uniquely identify the message

Naming conventions: help for messages

The name of a panel providing help for a message is based on the name of the message it explains. Figure 121 shows how help panels for messages are named.

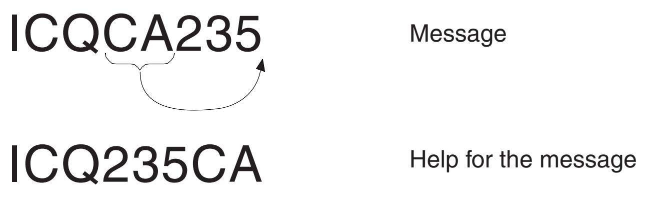


Figure 121. Naming conventions: help panels for messages

The two characters in the name of the message that indicate the function to which the message belongs become the last two characters in the name of the message help panel. The last three characters in the message name become the fourth, fifth, and sixth characters in the name of the message help panel.

On occasion, the help for a message can require two or three panels. The name of the second help panel is based on the name of the first help panel and the name of the third help panel is based on the name of the second help panel, as shown in Figure 122.

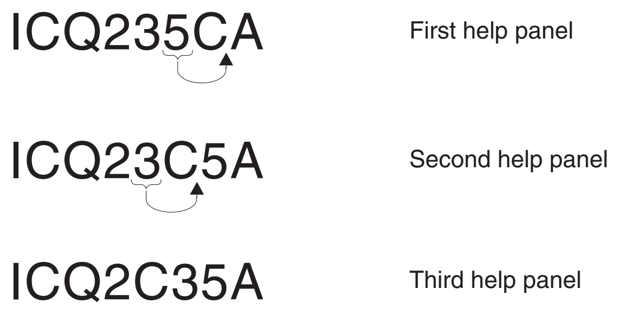


Figure 122. Naming conventions: help panels for messages, second and third panels

The sixth and seventh characters in the name of the first help panel are reversed in the name of the second help panel.

The fifth and sixth characters in the name of the second help panel are reversed in the name of the third help panel.

Application,panel, CLIST, and REXX exec hierarchy

Figures Figure 123 on page 661 through Figure 141 on page 683 show the Information Center Facility hierarchy of applications, panels, CLISTs, and REXX execs. You can use the figures as a reference when you customize the Information Center Facility.

For example, if you change CLIST ICQCAC30, check each CLIST and panel that ICQCAC30 invokes. ICQCAC30 invokes:
- ICQAAE30
- ICQCAE14
- ICQAAE15

- ICQCAE25

Figures Figure 123 on page 661 through Figure 141 on page 683 show only panel and function applications. In addition to panel and function applications, the Information Center Facility uses one environment application, ICQENVIRON. The function applications that use ICQENVIRONare:
- ICQADRS
- ICQAPLDI
- ICQAPL2
- ICQAPPLMGR
- ICQCOURSEA
- ICQCOURSES
- ICQENROLL
- ICQGRPSPEC
- ICQGRPAMGR
- ICQICU
- ICQIMAGE
- ICQNAMES
- ICQNAMESA
- ICQNEWS
- ICQNEWSA
- ICQPRINTER
- ICQPVTAMGR
- ICQUSERTYPE
- ICQVECTOR
- ICQVSAPL

To recognize applications, data entry panels, menu panels, help panels, tutorial panels, messages, CLISTs, and REXX execs, see “Using the ICF naming conventions” on page 651.

```
ICQADMIN                     TSO/E - Administration application
  │
  ├─ICQDESCRIBE          Describe application
  │
  ├─ICQNEWSA             TSO/E - News Maintenance application
  │   │
  │   ├─ICQANC00         Administrator NEWS CLIST (See Figure 122)
  │
  ├─ICQNAMESA            TSO/E - Names Administration application
  │   │
  │   ├─ICQCAC00         Administrator NAMES CLIST (See Figure 123)
  │
  ├─ICQENROLL            TSO/E - Enroll application
  │   │
  │   ├─ICQCAC00         ENROLL CLIST (See Figure 124)
  │
  ├─ICQUSERTYPE          TSO/E - User Types
  │   │
  │   ├─ICQADC00         USER TYPES CLIST (See Figure 125)
  │
  ├─ICQSYSDEF            TSO/E - System Defaults application
  │   │
  │   ├─ICQDESCRIBE      Describe application
  │   │
  │   ├─ICQISPFDEF       Set ISPF Defaults application
  │   │   │
  │   │   ├─ICQAIR00     REXX exec to process the system defaults profile
  │   │
  │   ├─ICQPRINTER       Set Printer Defaults for System application
  │   │   │
  │   │   ├─ICQAPC00     Printer CLIST (See Figure 126)
  │   │
  │   ├─ICQAPPLMGR       TSO/E System Application Manager
  │   │   │
  │   │   ├─ICQAMCM0     Application Manager CLIST (See Figure 127)
  │   │
  │   ├─ICQTUTOR         Tutorial
  │
  ├─ICQUSER              TSO/E - User Services (See Figure 129)
  │
  ├─ICQCOURSEA           TSO/E - Maintain Courses application
  │   │
  │   ├─ICQABM00         Administrator COURSES menu (See Figure 128)
  │
  ├─ICQPDF               ISPF/PDF Services application
  │
  ├─ICQPROBLEMA          TSO/E - Problem Reporting Services application
  │   │
  │   ├─ICQGAM01         Administration PROBLEM (place holder panel)
  │
  ├─ICQINTROA            Learn to Use application
  │
  ├─ICQTUTOR             Tutorial application
  │
  └─ICQEXIT              Exit application
```

Figure 123. Administration services application, panel, CLIST, and REXX exec hierarchy

```
ICQANC00                     The main administrator NEWS CLIST.
  │
  ├─ICQANE00          A data entry panel on which you specify an action you want to take and
  │                   the type of news item you want to display.  You can add a new item,
  │                   access the tutorial, or request to see a list of news items.
  │
  ├─ICQANE10          A list panel that displays available news items.
  │   │
  │   ├─ICQCNR20      A REXX exec for printing news/course abstracts.
  │   │
  │   ├─ICQCNE30      A data entry panel on which you can specify the printer type, location,
  │   │               and format.
  │   │
  │   ├─ICQCPC00      A CLIST that prints news items/course abstracts.
  │       │
  │       ├─ICQCPE00  A list panel for printer selection.
  │
  ├─ICQANE60          A data entry panel on which you describe the news item you are adding or copying.
  │
  ├─ICQANC01          A CLIST that adds or copies a news item.
  │   │
  │   ├─ICQANC05      A CLIST that adds or copies a news item you requested.
  │       │
  │       ├─ICQANE90  A data entry panel on which you type the text of a news item you are copying
  │                   or adding.
  │
  ├─ICQANE40          A data entry panel on which you modify the description of a news item.
  │
  ├─ICQANC02          A CLIST that modifies a news item.
  │   │
  │   ├─ICQANC05      A CLIST that modifies a news item you requested.
  │       │
  │       ├─ICQANE90  A data entry panel on which you type the text of a news item you are modifying.
  │
  ├─ICQANC03          A CLIST that deletes a news item.
  │   │
  │   ├─ICQANE20      A confirmation panel on which you confirm the deletion of a news item.
  │                   You can also view the news item before it is deleted.
  │
  ├─ICQCNC01          A CLIST that accesses and displays the text of a news item you requested to view.
  │   │
  │   ├─ICQCNE20      A view panel that displays the text of a news item you requested to view.
  │   │
  │   ├─ICQCNR20      A REXX exec for printing news/course abstracts.
  │       │
  │       ├─ICQCNE30  A data entry panel on which you can specify the printer type, location,
  │       │           and format.
  │       │
  │       ├─ICQCPC00  A CLIST for printer support.
  │           │
  │           ├─ICQCPE00 A list panel for printer selection.
  │
  └─ICQTN             Tutorial for administrator NEWS.
```

Figure 124. Administrator NEWS application, panel, CLIST, and REXX exec hierarchy

```
ICQCAC00                     The main CLIST for administrator and user NAMES and for ENROLL.
 │
 ├─ICQAAM00            A menu panel that lists the options you can choose to use and maintain
 │                     the names directory.
 │
 ├─ICQCAC10            A CLIST that processes names in the directory.
 │   │
 │   ├─ICQAAE10        A data entry panel on which you identify the name(s) in the directory you want
 │   │                 to see.  You can also add a new name to the directory.
 │   │
 │   ├─ICQAAE11        A list panel that displays the names you requested to see.  You can view, modify,
 │   │                 or delete a particular name.  You can also add a new name to the directory.
 │   │
 │   ├─ICQCAE12        A data entry panel on which you specify information about a name you are
 │   │                 adding or modifying.
 │   │
 │   ├─ICQAAE13        A data entry panel on which you specify information about a name you are
 │   │                 adding or modifying.
 │   │
 │   ├─ICQAEE50        A list panel that displays the user types you requested to see.
 │   │                 You select the user type for the name you are adding or modifying.
 │   │
 │   ├─ICQCAE14        A view panel that displays information about a particular name you requested
 │   │                 to view.
 │   │
 │   ├─ICQAAE15        A view panel that displays information about a particular name you requested
 │   │                 to view.
 │   │
 │   └─ICQCAE16        A confirmation panel on which you confirm the deletion of a name.
 │
 ├─ICQCAC20            A CLIST that processes groups in the directory.
 │   │
 │   ├─ICQAAE20        A data entry panel on which you identify the group(s) in the directory you want
 │   │                 to see.  You can also add a new group to the directory.
 │   │
 │   ├─ICQAAE21        A list panel that displays the groups you requested to see.
 │   │                 You can view, modify, or delete a particular group.
 │   │                 You can also add a new group to the directory.
 │   │
 │   ├─ICQAAE22        A list panel that displays the group name, description, and entries.
 │   │                 You can modify the name and description, view or delete a particular entry,
 │   │                 or add a new group entry.
 │   │
 │   ├─ICQCAE23        A list panel on which you add entries to a group you are modifying.  You can view
 │   │                 or delete a particular entry.  You can request a list of entries in the directory
 │   │                 and select names you want to add to the group.  You can also view or delete an
 │   │                 entry in the group.
 │   │
 │   ├─ICQAAE27        A list panel on which you describe a group you are adding and the group entries.
 │   │                 You can also request a list of names and select the group entries from the list.
 │   │
 │   ├─ICQCAE24        A list panel that displays the names you requested to see to include in a group.
 │   │                 You can view a particular name or select the names you want in the group.
 │   │
 │   ├─ICQCAE41        A list panel that displays the name, description, and entries of a group
 │   │                 you requested to view.
 │   │

 │   │
 │   ├─ICQCAE14        A view panel that displays information about a particular name you requested
 │   │                 to view.
 │   │
 │   ├─ICQCAE15        A view panel that displays information about a particular name you requested
 │   │                 to view.
 │   │
 │   ├─ICQAAE15        A view panel that displays information about a particular name
 │   │                 you requested to view.
 │   │
```

```
 │   ├─ICQCAE25     A list panel that displays the name, description, and entries of a group you
 │   │              requested to view.  You can view a particular entry within the group.
 │   │
 │   ├─ICQCAE26     A confirmation panel on which you confirm the deletion of a group
```

```
ICQCAC00                      The main CLIST for administrator and user NAMES and for ENROLL.
   │
   ├─ICQCAC10           A CLIST that processes names and enrollment.
   │
       ├─ICQAEE10       A data entry panel on which you specify an action you want to take (find, modify,
       │                or enroll).  You can "request a list of user(s)", as defined in the names
       │                directory, or enroll a person who is not defined in the names directory.  You can
       │                also access the tutorial.
       │
       ├─ICQAEE11       A list panel that displays the entries in the names directory you requested to see.
       │                You can enroll, view, or modify a name or enroll a user who is not defined
       │                in the directory.
       │
       ├─ICQCAE12       A data entry panel on which you specify information about a name you are adding
       │                or modifying.
       │
       ├─ICQAAE13       A data entry panel on which you specify information about a name you are adding
       │                or modifying.
       │
       ├─ICQAEE50       A list panel that displays the user types you requested to see.
       │                You select the user type for the name you are adding or modifying.
       │
       ├─ICQCAE14       A view panel that displays information about a name you requested to view.
       │
       ├─ICQAAE15       A view panel that displays information about a name you requested to view.
       │
       ├─ICQAEC00       A CLIST that enrolls the person in the Information Center Facility.
       │   │
       │   │  ┌─ICQAEE40 A data entry panel on which you identify the person you are enrolling to TSO/E,
       │   │  │          RACF, or the master catalog.
       │   ├──┤
       │   │  └─ICQAEE41 A data entry panel on which you identify the person you are enrolling to TSO/E
       │   │             and/or RACF.
       │   │
       │   ├─ICQAIR00   A CLIST that updates the ISPF defaults for a user.
       │
       └─ICQTE          Tutorial for ENROLL.
```

Figure 126. Administrator enrollment application, panel, CLIST, and REXX exec hierarchy

```
ICQADC00                    The main CLIST for administration USER TYPES.
  │
  ├─ICQADE00          A data entry panel on which you specify an action you want to take
  │                   (add, delete, modify or view) and information about the user type.
  │                   You can also access the tutorial.
  │
  ├─ICQADE01          A data entry panel that is displayed if you add a user type which already exists.
  │                   You can delete, view, modify, or replace the user type or add another one with a
  │                   different name or description.
  │
  ├─ICQADE02          A list panel that displays the user types you requested to see.  You can delete,
  │                   view, or modify a particular user type.
  │
  ├─ICQADE03          A data entry panel that is displayed if you and another user are adding a user type
  │                   with the same name.  You can specify either a different user type or user
  │                   description.
  │
  ├─ICQADE14          A confirmation panel on which you confirm the deletion of a user type.
  │
  ├─ICQADM16          A menu panel that lists options that define a user type.
  │   │
  │   ├─ICQADC01      A CLIST that accesses and displays the TSO/E user ID information.
  │   │  │
  │   │  ├─ICQADE04   A view panel that displays TSO/E user ID information for the user type you
  │   │  │            requested.
  │   │  │
  │   │  ├─ICQADE05   A view panel that displays the authorization information for the user type you
  │   │  │            requested.
  │   │  │
  │   │  ├─ICQADE06   A data entry panel on which you specify user ID information for the user type
  │   │  │            you are adding or replacing.
  │   │  │
  │   │  └─ICQADE07   A data entry panel on which you specify authorization information for the
  │   │               user type you are adding or replacing.
  │   │
  │   ├─ICQADC02      A CLIST that accesses and displays RACF security information.
  │   │  │
  │   │  ├─ICQADE08   A view panel that displays RACF security information for the user type you
  │   │  │            requested.
  │   │  │
  │   │  ├─ICQADE09   A view panel that displays optional RACF information for the user type you
  │   │  │            requested.
  │   │  │
  │   │  ├─ICQADE10   A data entry panel on which you specify RACF security information for the user
  │   │  │            tape you are adding or replacing.
  │   │  │
  │   │  └─ICQADE11   A data entry panel on which you specify optional RACF information for the user type
  │   │               you are adding or replacing.
  │   │
  │   └─ICQADC03      A CLIST that accesses and displays the DEFINE ALIAS parameters.
  │      │
  │      ├─ICQADE12   A view panel that displays DEFINE ALIAS parameters for the user type you requested.
  │      │
  │      └─ICQADE13   A data entry panel on which you specify DEFINE ALIAS parameters for the user type
  │                   you are adding or replacing.
  │

  │
  ├─ICQADM17          A menu panel on which you select which part of the user type you want to modify.
  │   │
  │   ├─ICQADC01      A CLIST that accesses and displays user ID information.
  │   │  │
  │   │  ├─ICQADE06   A data entry panel on which you specify user ID information for the user type
  │   │  │            you are modifying.
  │   │  │
  │   │  └─ICQADE07   A data entry panel on which you specify authorization information for the user
  │   │               type you are modifying.
  │   │
  │   ├─ICQADC02      A CLIST that accesses and displays RACF security information.
  │   │  │
  │   │  ├─ICQADE10   A data entry panel on which you specify RACF security information for the user
  │   │  │            type you are modifying.
  │   │  │
```

```
│
├─ICQAPC00           A CLIST that updates printer defaults for the system.
│    │
│    ├─ICQAPE00      A list panel that displays the printers.  You can modify, delete, or copy
│    │               a particular printer.  You can also add a new printer or access the tutorial.
│    │
│    ├─ICQAPE10      A confirmation panel on which you confirm the deletion of a selected printer.
│
├─ICQAPM20           A menu panel on which you can add or modify a printer's characteristics.
│    │               You can view or modify the name and location, specify functional or installation
│    │               characteristics, define characters sets or output functions, and test the
│    │               printer function.
│    │
│    ├─ICQAPE30      A data entry panel on which you specify printer characteristics such as location,
│    │    │          format, and type for a printer you are adding or modifying.
│    │    │
│    │    ├─ICQAPE40 A list panel that displays the printer types.  You select a type from the list.
│    │    │
│    │    ├─ICQAPE41 A list panel that displays the printer formats.  You select a format
│    │    │          from the list.
│    │    │
│    │    ├─ICQAPE42 A list panel that displays the printer locations.  You select a location
│    │               from the list.
│    │
│    ├─ICQAPE80      A data entry panel on which you define a print function and parameters.
│    │
│    ├─ICQAPE50      A data entry panel on which you specify general printer characteristics
│    │               such as sysout class, program, form, and UCS and FCB names.
│    │
│    ├─ICQAPE51      A data entry panel on which you specify general printer characteristics
│    │               such as module name, translate code, flash name, and count.
│    │
│    ├─ICQAPE52      A data entry panel on which you specify general printer characteristics
│    │               for the number of copies parameter.
│    │
│    ├─ICQAPE53      A data entry panel on which you can specify options of the PRINTDS command such as
│    │               output descriptors, sysout forms, whether to print PDS members or directories,
│    │               and the data set to which the output goes.
│    │
│    ├─ICQAPE54      A data entry panel on which you can specify options of the PRINTDS command such as
│    │               sysout class, page length, title, margins, and line spacing.
│    │
│    ├─ICQAPE55      A data entry panel on which you can specify a range of lines to be printed.
│    │
│    ├─ICQAPE56      A data entry panel on which you can specify columns of data to be printed.
│    │
│    ├─ICQAPE57      A data entry panel on which you can specify options of the PRINTDS command such as
│    │               sysout writer, UCS and FCB names, flash, burst, and number of copies to print.

│    │
│    ├─ICQAPE60      A data entry panel on which you specify characteristics which are specific
│    │               to the printer.
│    │
│    ├─ICQAPE90      A data entry panel on which you specify the data set you want to print to test the
│    │               print function.
│    │
│    ├─ICQCPC20      A CLIST that accesses the fonts which are available on the system.
│    │    │
│    │    ├─ICQAPE70 A data entry panel on which you specify and maintain a list of the fonts
│    │               for the printer.
│    │
│    └─ICQTP         Tutorial for maintaining the printer list.
│
└─ICQX5              Tutorial for maintaining the system defaults.
```

Figure 128. Administrator printer defaults application, panel, CLIST, and REXX exec hierarchy

```
│
├─ICQAMCM0            The main CLIST for system, group, and private Application Manager.  It establishes
│   │                 control for the administrator, opens tables, and if necessary, calls the update
│   │                 recovery routine.
│   │
│   ├─ICQSIEAM        A non-display panel that contains the variables for customizing Application
│   │                 Manager.
│   │
│   ├─ICQAMEA2        A list panel that you use to identify the group you want to administer.
│   │
│   ├─ICQAMRG1        A REXX exec which displays a selection list of Application Manager groups
│   │   │             and returns the selected group's information.
│   │   │
│   │   ├─ICQGCL00    A program that creates the list of groups to be displayed.
│   │   │
│   │   └─ICQAMEA3    A panel that displays a list of groups.
│   │
│   ├─ICQAME10        A list panel that displays a list of applications.  You can add, upgrade, view,
│   │                 copy, delete, modify, export applications, and find out where the applications are
│   │                 used. You can also display a hierarchy of applications and create an installation
│   │                 file from an existing application.
│   │
│   ├─ICQAMCS0        A CLIST that generates and controls the hierarchy display.
│   │   │
│   │   ├─ICQAMCS2    A CLIST that extracts the information and creates the hierarchy display.
│   │   │
│   │   └─ICQAME19    A list panel that displays the hierarchy listing in browse mode.
│   │
│   ├─ICQAMRY0        A REXX exec that controls upgrade processing.
│   │   │
│   │   ├─ICQAMRY2    A REXX exec that processes the upgrade file against the installation file.
│   │   │
│   │   ├─ICQAMRY3    A REXX exec that identifies errors in the upgrade file.
│   │   │
│   │   ├─ICQGCC00    A CLIST that performs a general search for files specified with an * (asterisk)
│   │   │             suffix.
│   │   │
│   │   └─ICQAME88    A data entry panel on which you can specify an installation file to upgrade
│   │       │         and the upgrade file.
│   │       │
│   │       ├─ICQAME8A A list panel that displays a list of installation members that can be selected or
│   │       │          edited.
│   │       │
│   │       ├─ICQAME8B A list panel that displays a list of upgrade members that can be selected or
│   │       │          edited.
│   │       │
│   │       └─ICQGCE17 A list panel that displays a list of data sets that can be selected or edited.
│   │
│   ├─ICQAME13        A menu panel on which you can select the type of application you want to add.
│   │   │             You can also choose to add an application using an installation file.
│   │   │
│   │   ├─ICQAMCP0    A CLIST that creates, copies, or modifies a panel definition.
│   │   │                   (See "Define a Panel" in Part 3 of this figure.)
│   │   │
│   │   ├─ICQAMCF0    A CLIST that creates, copies, or modifies a function or an environment definition.
│   │   │                   (See "Define a Function or Environment" in Part 4 of this figure.)
│   │   │
│   │   └─ICQAMCI0    A CLIST that obtains the name of the sequential data set or partitioned
│   │                 data set and the member that contains the installation file that is to be loaded or
│   │                 edited.  (See "Load an Installation File" in Part 5 of this figure.)
│   │
│   ├─ICQAME16        A confirmation panel that confirms your request to delete an application.
│   │
│   ├─ICQAME17        A list panel that displays a list of applications that use a specified application.
│   │   │             You can delete, modify, view, and find out where applications on the list are used.
│   │   │             You can also display a hierarchy listing of applications.
│   │   │
│   │   └─ICQAMCS0    A CLIST that generates and controls the hierarchy display.
│   │       │
│   │       ├─ICQAMCS2 A CLIST that extracts the information and creates the hierarchy display.
│   │       │
│   │       └─ICQAME19 A list panel that displays the hierarchy listing in browse mode.
```

```
ICQABM00  The menu panel for administrator courses (Information Center Facility)
│
├─ICQABC00          A CLIST that adds, deletes, views, modifies, and prints course abstracts.
│   │
│   ├─ICQABE10(20)  A list panel that displays the course abstracts.  You can view, modify, print, and
│   │               delete a particular abstract, or add a new course abstract.
│   │
│   ├─ICQCNR20      A REXX exec for printing news/course abstracts.
│   │   │
│   │   ├─ICQCNE30  A data entry panel on which you can specify the printer type, location,
│   │   │           and format.
│   │   │
│   │   ├─ICQCPC00  A CLIST for printer support.
│   │       │
│   │       ├─ICQCPE00 A list panel for printer selection.
│   │
│   ├─ICQABM30      A menu panel on which you select the type of course you want to add (IIPS,
│   │               classroom, or computer).
│   │
│   ├─ICQABEQ0      A data entry panel on which you specify information about an IIPS course you
│   │               are adding.
│   │
│   ├─ICQABER0      A data entry panel on which you specify information about a classroom course you
│   │               are adding.
│   │
│   ├─ICQABES0      A data entry panel on which you specify information about a computer course you
│   │               are adding.
│   │
│   ├─ICQANC01      A CLIST that processes course abstracts when you add an abstract.
│   │   │
│   │   ├─ICQANC05  A CLIST that processes course abstracts when you add an abstract.
│   │       │
│   │       ├─ICQABE40 A data entry panel on which you specify the text of the course abstract you are
│   │                  adding.
│   │
│   ├─ICQABE50      A data entry panel on which you modify the characteristics of an IIPS course.
│   │
│   ├─ICQABET0      A data entry panel on which you modify the characteristics of a classroom course.
│   │
│   ├─ICQABEU0      A data entry panel on which you modify the characteristics of a computer course.
│   │
│   ├─ICQANC02      A CLIST that modifies a course description.
│   │   │
│   │   ├─ICQANC05  A CLIST that processes course abstracts when you modify an abstract.
│   │       │
│   │       ├─ICQABE40 A data entry panel on which you specify the text of the course abstract you are
│   │                  modifying.
│   │
│   ├─ICQANC03      A CLIST that deletes a course.
│   │   │
│   │   ├─ICQABE70  A confirmation panel on which you confirm the deletion of an IIPS or classroom
│   │   │           course.  You can view the course abstract before the course is deleted.
│   │   │
│   │   ├─ICQABEV0  A confirmation panel on which you confirm the deletion of a computer course.  You
│   │               view the course abstract before the course is deleted.
│   │
│   ├─ICQCNC01      A CLIST that accesses a course abstract and displays information about the course.
│       │
│       ├─ICQCBE20  A view panel that displays the course abstract you requested to view.
│           │
│           │
│           ├─ICQCNR20 A REXX exec for printing news/course abstracts.
│               │
│               ├─ICQCNE30 A data entry panel on which you can specify the printer type, location,
│               │          and format.
│               │
│               ├─ICQCPC00 A CLIST for printer support.
│                   │
│                   ├─ICQCPE00 A list panel for printer selection.
│
├─ICQABC20          A CLIST that processes registration requests.
│   │
```

```
ICQUSER                     TSO/E - User Services application
 │
 ├─ICQDESCRIBE       Describe application
 │
 ├─ICQNEWS           Information Center Facility - News application
 │   │
 │   ├─ICQCNC00      User NEWS  (See Figure 131)
 │
 ├─ICQNAMES          Information Center Facility - Name application
 │   │
 │   ├─ICQCAC00      User NAMES  (See Figure 132)
 │
 ├─ICQOFFICE         TSO/E - Office Services application
 │   │
 │   ├─ICQDESCRIBE   Describe application
 │   │
 │   ├─ICQPSTSO      PS/TSO application
 │   │
 │   ├─ICQTUTOR      Tutorial application
 │
 ├─ICQPROGRAM        TSO/E - Programmer Services
 │   │
 │   ├─ICQDESCRIBE   Describe application
 │   │
 │   ├─ICQBASIC      IBM BASIC/MVS application
 │   │
 │   ├─ICQAPL2       APL-II application
 │   │  │
 │   │  ├─ICQCLC00   APL-II (See Figure 138)
 │   │
 │   ├─ICQVSAPL      VS/APL application
 │   │  │
 │   │  ├─ICQCLC00   VS/APL (See Figure 138)
 │   │
 │   ├─ICQTIF        The Information Facility application
 │   │
 │   ├─ICQAS         Application System application
 │   │
 │   ├─ICQGRPSPEC    Group specification application
 │   │  │
 │   │  ├─ICQAMRGO   A REXX exec that processes group specification.
 │   │     │
 │   │     ├─ICQAMEA1 Group specification panel
 │   │     │
 │   │     ├─ICQAMRG1 A REXX exec which displays a selection list of Application Manager
 │   │     │   │      groups and returns the selected group's information.
 │   │     │   │
 │   │     │   ├─ICQGCC00 A CLIST that performs a general search for files
 │   │     │                  specified with an * (asterisk) suffix.
 │   │     │
 │   │     ├─ICQAMEA3 A list panel that displays groups.
 │   │
 │   │
 │   ├─ICQGROUP      Group Application Manager
 │   │  │
 │   │  ├─ICQAMCM0   Application Manager CLIST (See Figure 127)
 │   │
 │   ├─ICQPRIVATE    Private Application Manager
 │   │  │
 │   │  ├─ICQAMCM0   Application Manager CLIST (See Figure 127)
 │   │
 │   ├─ICQTUTOR      Tutorial application
 │
 ├─ICQANALYSIS       Analysis/Report Creation Services application
 │   │
 │   ├─ICQDESCRIBE   Describe application
 │   │
 │   ├─ICQAPLDI      APLDI-II application
 │   │  │
 │   │  ├─ICQCIC00   APLDI II  (See Figure 133)
 │   │
 │   ├─ICQADRS       A Departmental Reporting System application
 │
 ├─ICQAMED1          A menu panel that lists the utilities you can select.
 │   │
     ├─ICQGCR70      A REXX exec that processes print requests.
         │
         ├─ICQGCE35  A data entry panel on which you can specify a print request.
         │
         ├─ICQCPC00  A CLIST for printer support.
         │   │
         │   ├─ICQCPE00 A list panel for printer selection.
         │
         ├─ICQGCC00  A CLIST that performs a general search for files specified with
         │           an * (asterisk) suffix.
         │
         ├─ICQGCE50  A list panel for member selection.
```

Figure 132. User services—print utility application, panel, CLIST, and REXX exec hierarchy

```
ICQCNC00                     The main CLIST for user NEWS.
 │
 ├─ICQCNE00          A data entry panel on which you specify the type of news item you want to display.
 │                   You can also access the tutorial.
 │
 ├─ICQCNE10          A list panel that displays the news items you requested.  You can select a
 │   │               particular item to view or print.
 │   │
 │   ├─ICQCNR20      A REXX exec for printing news/course abstracts.
 │       │
 │       ├─ICQCNE30  A data entry panel on which you can specify the printer type, location,
 │       │           and format.
 │       │
 │       ├─ICQCPC00  A CLIST for printer support.
 │           │
 │           ├─ICQCPE00 A list panel for printer selection.
 │
 ├─ICQCNC01          A CLIST that accesses the text of a news item when you request to view the item.
 │   │
 │   ├─ICQCNE20      A view panel that displays the text of a news item and allows you to
 │   │               print a news item.
 │   │
 │   ├─ICQCNR20      A REXX exec for printing news/course abstracts.
 │       │
 │       ├─ICQCNE30  A data entry panel on which you can specify the printer type, location,
 │       │           and format.
 │       │
 │       ├─ICQCPC00  A CLIST for printer support.
 │           │
 │           ├─ICQCPE00 A list panel for printer selection.
 │
 └─ICQUN0            Tutorial for user NEWS.
```

Figure 133. User NEWS application, panel, CLIST, and REXX exec hierarchy

```
ICQCAC00                    The main CLIST for administrator and user NAMES and for ENROLL.
 │
 ├─ICQCAM00           The menu panel for user NAMES.  You select either names, groups, close (close the
 │                    directories), change (change the private directory prefix), or the tutorial.
 │
 ├─ICQCAC10           A CLIST that creates and maintains names in the names directory.
 │   │
 │   ├─ICQCAE10       A data entry panel on which you identify the name(s) you want to display in the
 │   │                names directory.  You can also add a new name to the directory.
 │   │
 │   ├─ICQCAE11       A list panel that displays the names you requested to see.  You can view,
 │   │                modify, or delete a particular name.  You can also add a new name to the directory.
 │   │
 │   ├─ICQCAE12       A data entry panel on which you specify information about a name you are adding or
 │   │                modifying.
 │   │
 │   ├─ICQCAE13       A data entry panel on which you specify information about a name you are
 │   │                adding or modifying.
 │   │
 │   ├─ICQCAE14       A view panel that displays information about a particular name you requested
 │   │                to view.
 │   │
 │   ├─ICQCAE15       A view panel that displays information about a particular name you requested
 │   │                to view.
 │   │
 │   ├─ICQCAE16       A confirmation panel on which you confirm the deletion of a name.
 │
 ├─ICQCAC20           A CLIST that creates and maintains groups in the names directory.
 │   │
 │   ├─ICQCAE20       A data entry panel on which you identify the group(s) you want to display in the
 │   │                names directory.  You can also add a new group to the directory.
 │   │
 │   ├─ICQCAE21       A list panel that displays the groups you requested to see.  You can view, modify,
 │   │                or delete a particular group.  You can also add a new group to the directory.
 │   │
 │   ├─ICQCAE22       A list panel that displays the group name, description, and entries.  You can
 │   │                modify the name and description, view or delete a particular entry, or
 │   │                add a new group entry.
 │   │
 │   ├─ICQCAE23       A list panel on which you add entries to a group you are modifying.  You can view
 │   │                or delete a particular entry.  You can request a list of entries in the directory
 │   │                and select the names you want to add to the group.  You can also view or delete
 │   │                an entry in the group.
 │   │
 │   ├─ICQCAE27       A list panel on which you describe a group you are adding and the group entries.
 │   │                You can also request a list of names and select the group entries from the list.
 │   │
 │   ├─ICQCAE24       A list panel that displays the names you requested to see to include in a group.
 │   │                You select the names you want in the group.
 │   │
 │   ├─ICQCAE14       A view panel that displays information about a particular name you requested
 │   │                to view.
 │   │
 │   ├─ICQCAE15       A view panel that displays information about a particular name you requested
 │   │                to view.
 │   │
 │   ├─ICQCAE25       A view panel that displays the name, description, and entries in the group you
 │   │                requested to view.
 │   │
 │   ├─ICQCAE26       A confirmation panel on which you confirm the deletion of a group.
 │
 └─ICQUA              Tutorial for user NAMES.
```

Figure 134. User NAMES application, panel, and CLIST hierarchy

```
ICQCIC00                The main CLIST for using the APLDI-II program product to extract data stored in
  │                     files.
  │
  ├─ICQCIM00            A menu panel that lists the options you can select to use APLDI-II.
      │
      ├─ICQCIC10        A CLIST that accesses the online query files when you query or update a file.
      │  │
      │  ├─ICQCIE10     A data entry panel on which you specify information about the file and
      │  │              workspace when you query a file.
      │  │
      │  ├─ICQCIE20     A data entry panel on which you specify information about the query
      │  │              file and workspace when you update a file.
      │  │
      │  ├─ICQGCC30     See Figure 134 for the continuation of the panel and
      │  │              CLIST hierarchy for ICQGCC30.
      │  │
      │  ├─ICQGCC40     A CLIST that saves the current workspace.
      │  │  │
      │  │  ├─ICQGCE40  A data entry panel on which you specify whether you want to save the current work-
      │  │  │           space.  If you save the workspace, you must specify a project and workspace name.
      │  │  │
      │  │  ├─ICQGCE41  A data entry panel that is displayed if you are saving a workspace and
      │  │              you specified a workspace name that already exists.  You can specify
      │  │              that you do not want to save the workspace, type in a different name,
      │  │              or replace the old workspace.
      │  │
      │  ├─ICQGCE01     A CLIST that handles the display of messages when APLDI II is being loaded,
      │  │              if you have an APL keyboard.
      │  │
      │  ├─ICQGCE02     A CLIST that handles the display of messages when APLDI II is being loaded,
      │                 if you have an non-APL keyboard.
      │
      ├─ICQCIC30        A CLIST that creates query files.
      │  │
      │  ├─ICQCIE30     A data entry panel on which you specify information about the data and
      │  │              definition files.
      │  │
      │  ├─ICQCIE32     A data entry panel on which you specify information about the query file.
      │  │
      │  ├─ICQCIE33     A data entry panel that is displayed if you specified a query file which
      │  │              already exists.  You can specify a new file or replace the existing file.
      │  │
      │  ├─ICQCIE34     A data entry panel on which you can modify the options used in creating
      │  │              the query file.
      │  │
      │  ├─ICQCIC40     A CLIST that calculates the amount of space you need to create a query file.
      │  │
      │  ├─ICQCIC35     A CLIST that accesses the definition file for editing.
      │  │  │
      │  │  ├─ICQCIE35  A data entry panel on which you specify information about the definition file.
      │  │
      │  ├─ICQGCC30     See Figure 134 for the continuation of the panel and
      │  │              CLIST hierarchy for ICQGCC30.
      │  │
      │  │
      │  ├─ICQGCC40     A CLIST that saves the current workspace.
      │     │
      │     ├─ICQGCE40  A data entry panel on which you specify whether or not you want to
      │     │           save the current workspace.  If you save the workspace, you must
      │     │           specify a project and workspace name.
      │     │
      │     ├─ICQGCE41  A data entry panel that is displayed if you are saving a workspace
      │                 and you specified a workspace name that already exists.  You can
      │                 specify that you do not want to save the workspace, type in a
      │                 different name, or replace the old workspace.
      │
      ├─ICQCIC40        A CLIST that calculates the amount of space you need to create a query file.
      │  │
      │  ├─ICQCIE40     A data entry panel on which you specify information about the definition,
      │  │              query, and data files.
      │  │
      │  ├─ICQGCC30     See Figure 134 for the continuation of the panel and
      │  │              CLIST hierarchy for ICQGCC30.
```

```
│
├─ICQGCC30          This CLIST is a continuation of the hierarchy for APLDI-II from Figure 133.
│     │             The CLIST accesses the workspaces within a particular project.
│     │
│     ├─ICQGCC00    A CLIST that accesses the data sets within a particular project.
│         │
│         ├─ICQGCE13    A list panel that displays the files you requested for workspace.  You can
│         │             select a file from the list or delete a particular file.
│         │
│         ├─ICQGCE15    A list panel that displays the files you requested for workspace.  You can
│         │             select a file from the list or delete a particular file.
│         │
│         ├─ICQGCE17    A list panel that displays the files you requested for a query file.  You can
│         │             select a file from the list or delete a particular file.
│         │
│         ├─ICQGCE19    A confirmation panel on which you confirm the deletion of a data set.
│         │
│         ├─ICQGCL00    A command to search the catalog to obtain a list of data sets.
```

Figure 136. APLDI II hierarchy for the CLIST that accesses requested files (ICQGCC30)

```
ICQCRC00                        The main CLIST for using the ADRS-II program product to design and print
 │                              reports, collect data, and calculate results.
 │
 ├─ICQCRMOO               A menu panel on which you select one of three options to use ADRS-II.  You can
 │   │                    also select the tutorial.
 │   │
 │   ├─ICQCRC10           A CLIST that creates a new ADRS workspace.
 │   │  │
 │   │  ├─ICQCRE10        A data entry panel on which you specify the name of a new workspace.
 │   │  │
 │   │  ├─ICQCRE11        A data entry panel that is displayed if the name of the workspace you specified
 │   │  │                 already exists.  You can specify another name or replace the existing workspace.
 │   │  │
 │   │  ├─ICQGCE01        A CLIST that handles the display of messages when ADRS is being loaded, if
 │   │  │                 you have an APL keyboard.
 │   │  │
 │   │  ├─ICQGCE02        A CLIST that handles the display of messages when ADRS is being loaded, if
 │   │  │                 you have a non-APL keyboard.
 │   │  │
 │   │  ├─ICQGCC30        A CLIST that accesses the workspaces within a particular project.
 │   │  │  │
 │   │  │  ├─ICQGCC00     A CLIST that accesses the data sets within a particular project.
 │   │  │     │
 │   │  │     ├─ICQGCE13  A list panel that displays the files you requested for workspace.  You can
 │   │  │     │           select a file from the list or delete a particular file.
 │   │  │     │
 │   │  │     ├─ICQGCE15  A list panel that displays the files you requested for workspace.  You can
 │   │  │     │           select a file from the list or delete a particular file.
 │   │  │     │
 │   │  │     └─ICQGCE19  A confirmation panel on which you confirm the deletion of a data set.
 │   │  │
 │   │  └─ICQGCC40        A CLIST that saves the current workspace.
 │   │     │
 │   │     ├─ICQGCE40     A data entry panel on which you specify whether or not you want to save the
 │   │     │              current workspace.  If you save the workspace, you must specify a project and
 │   │     │              workspace name.
 │   │     │
 │   │     └─ICQGCE41     A data entry panel that is displayed if you are saving a workspace and you
 │   │                    specified a workspace name that already exists.  You can specify that you do not
 │   │                    want to save the workspace, type in a different name, or replace the old
 │   │                    workspace.
 │   │
 │   ├─ICQCRC20           A CLIST that accesses an existing ADRS workspace.
 │   │  │
 │   │  ├─ICQCRE20        A data entry panel on which you specify the name of an existing ADRS workspace.
 │   │  │
 │   │  ├─ICQGCE01        A CLIST that handles the display of messages when ADRS is being loaded, if
 │   │  │                 you have an APL keyboard.
 │   │  │
 │   │  ├─ICQGCE02        A CLIST that handles the display of messages when ADRS is being loaded, if
 │   │  │                 you have a non-APL keyboard.
 │   │  │
 │   │  ├─ICQGCC30        A CLIST that accesses the workspaces within a particular project.
 │   │  │  │
 │   │  │  ├─ICQGCC00     A CLIST that accesses the data sets within a particular project.
 │   │  │     │
 │   │  │     ├─ICQGCE13  A list panel that displays the files you requested for workspace.  You can
 │   │  │     │           select a file from the list or delete a particular file.
 │   │  │     │
 │   │  │     ├─ICQGCE15  A list panel that displays the files you requested for workspace.  You can
 │   │  │     │           select a file from the list or delete a particular file.
 │   │  │     │
 │   │  │     └─ICQGCE19  A confirmation panel on which you confirm the deletion of a data set.
 │   │  │
 │   │  └─ICQGCC40        A CLIST that saves the current workspace.
 │   │     │
 │   │     ├─ICQGCE40     A data entry panel on which you specify whether or not you want to save the
 │   │     │              current workspace.  If you save the workspace, you must specify a project and
 │   │     │              workspace name.
 │   │     │
 │   │     └─ICQGCE41     A data entry panel that is displayed if you are saving a workspace and you
 │   │                    specified a workspace name that already exists.  You can specify that you do not
```

```
ICQGCM02                     A menu panel on which you can select three services to create charts, graphs,
  │                          or symbols.  You can also access the tutorial.
  │
  ├─ICQCGC00           A CLIST for using the Interactive Chart Utility.
  │   │
  │   ├─ICQCGE00       A data entry panel on which you identify files in which to store your charts.
  │   │
  │   ├─ICQCGE03       A data entry panel on which you specify up to three alternate
  │   │                symbol libraries to use in the Interactive Chart Utility.
  │   │
  │   ├─ICQGCC00       A CLIST that accesses and displays a list of the files you can select.
  │   │  │
  │   │  ├─ICQGCE17    A list panel that displays the files you requested to see.  You select the file
  │   │  │             you want to use from the list.  You can also delete a particular file.
  │   │  │
  │   │  ├─ICQGC319    A confirmation panel on which you confirm the deletion of a file.
  │   │
  │   ├─Interactive Chart Utility
  │
  ├─ICQCGC01           A CLIST for using the image symbol set editor.
  │   │
  │   ├─ICQCGE01       A data entry panel on which you identify the image and object files for your
  │   │                image symbol sets.
  │   │
  │   ├─ICQGCC00       A CLIST that accesses and displays a list of the files  you can select.
  │   │  │
  │   │  ├─ICQGCE17    A list panel that displays the files you requested to see.  You select the file
  │   │  │             you want to use from the list.  You can also delete a particular file.
  │   │  │
  │   │  ├─ICQGCE19    A confirmation panel on which you confirm the deletion of a file.
  │   │
  │   ├─Image Symbol Editor
  │
  ├─ICQCGC02           A CLIST for using the vector symbol set editor.
  │   │
  │   ├─ICQCGE02       A data entry panel on which you identify the vector file.
  │   │
  │   ├─ICQGCC00       A CLIST that accesses and displays a list of the files you can select.
  │   │  │
  │   │  ├─ICQGCE17    A list panel that displays the files you requested to see.  You select the file
  │   │  │             you want to use from the list.  You can also delete a particular file.
  │   │  │
  │   │  ├─ICQGCE19    A confirmation panel on which you confirm the deletion of a file.
  │   │
  │   ├─Vector Symbol Editor
  │
  └─ICQUG              Tutorial for Chart Creation Services.
```

Figure 138. Chart creation services application, panel, and CLIST hierarchy

```
ICQCBC00                     The main CLIST for user COURSES.
  │
  ├─ICQCBE00(10)       A list panel that displays the courses.  You can audit, register for, take, view,
  │   │                or produce (write) a particular course and you can access the tutorial.
  │   │                You can also print a course abstract.
  │   │
  │   ├─ICQCNR20       A REXX exec for printing news/course abstracts.
  │      │
  │      ├─ICQCNE30    A data entry panel on which you can specify the printer type, location,
  │      │                and format.
  │      │
  │      ├─ICQCPC00    A CLIST for printer support.
  │         │
  │         ├─ICQCPE00 A list panel for printer selection.
  │
  ├─ICQCBE30           A confirmation panel on which you confirm that you want to audit the IIPS course
  │                    you selected.
  │
  ├─ICQCBE40           A data entry panel on which you specify your IIPS student number in order to take
  │                    the IIPS course you selected.
  │
  ├─ICQCBE50           A data entry panel on which you specify your IIPS author number, and optionally,
  │                    the course segment number, in order to write an IIPS course.
  │
  ├─ICQCBE60           A confirmation panel on which you confirm that you want to audit the computer
  │                    course you selected.
  │
  ├─ICQCBE70           A confirmation panel on which you confirm that you want to take the computer course
  │                    you selected.
  │
  ├─ICQCNC01           A CLIST that accesses the course abstracts.
  │   │
  │   ├─ICQCBE20       A view panel that displays the course abstract you requested and gives you the
  │   │                option to print.
  │   │
  │   ├─ICQCNR20       A REXX exec for printing news/course abstracts.
  │      │
  │      ├─ICQCNE30    A data entry panel on which you can specify the printer type, location,
  │      │             and format.
  │      │
  │      ├─ICQCPC00    A CLIST for printer support.
  │         │
  │         ├─ICQCPE00 A list panel for printer selection.
  │
  └─ICQUB              Tutorial for User Courses.
```

Figure 139. User courses application, panel, CLIST, and REXX exec hierarchy

```
ICQCLC00                    The main CLIST for programming in either the APL2 or VS APL environment.
  │
  ├─ICQCLM00          A menu panel that lists the programmer services you can use for APL2 or VS APL.
  │
  ├─APL2 program product
  │
  ├─ICQCLC10          A CLIST that modifies APL2 or VS APL invocation parameters.
  │   │
  │   ├─ICQCLE10      A data entry panel on which you specify the VS APL interactive control
  │   │               parameters.
  │   │
  │   ├─ICQCLE11      A data entry panel on which you specify the VS APL library parameters.
  │   │
  │   ├─ICQCLE20      A data entry panel on which you specify the APL2 interactive control parameters.
  │   │
  │   ├─ICQCLE21      A data entry panel on which you specify the storage parameters for APL2.
  │   │
  │   ├─ICQCLE22      A data entry panel on which you specify the library parameters for APL2.
  │   │
  │   ├─ICQCLE23      A data entry panel on which you specify the options for the debug parameter
  │                   for APL2.
  │
  ├─ICQCLC20          The CLIST for reusing an existing workspace for either APL2 or VS APL programming.
      │
      ├─ICQCRE20      A data entry panel on which you specify the project and name of an existing
      │               workspace.
      │
      ├─ICQGCC30      A CLIST that accesses the existing workspaces within a  particular project.
         │
         ├─ICQGCC00   A CLIST that accesses the existing data sets within a particular project.
            │
            ├─ICQGCE15 A list panel that displays the files (workspaces) you requested to see.
            │          You can either select or delete a workspace from the list.
            │
            ├─ICQGCE19 A confirmation panel on which you confirm the deletion of a data set (workspace).
```

Figure 140. APL2 and VS APL program environment application, panel, and CLIST hierarchy

```
ICQAMRMI                     A REXX exec that controls the mass installation file process.
  │
  ├─ICQAMRME           A REXX exec that controls the mass export of applications.
  │   │
  │   ├─ICQAMRM1       A REXX exec that controls a single export of an application.
  │      │
  │      ├─ICQAMCX2    A CLIST that exports the application table into an installation file.
  │
  ├─ICQAMRML           A REXX exec that controls the mass upgrade and load of applications.
  │   │
  │   ├─ICQAMRM1       A REXX exec that controls a single export of an application.
  │   │  │
  │   │  ├─ICQAMCX2    A CLIST that exports the application table into an installation file.
  │   │
  │   ├─ICQAMRY2       A REXX exec that processes the upgrade file against the installation
  │   │                file.
  │   │
  │   ├─ICQAMCI2       A CLIST that loads an installation file into a table and calls the
  │      │             appropriate verification routine.
  │      │
  │      ├─ICQAMCF0    A CLIST that defines function and environment applications.
  │      │
  │      ├─ICQAMCP0    A CLIST that defines panel applications.
```

Figure 141. Mass installation file processing application, panel, CLIST, and REXX exec hierarchy

```
ICQASRM0                     The REXX exec that processes the main menu and associated requests, presents
 │                           and processes the data base token panel, and performs file allocation.
 │
 └ICQASE02                    A menu panel that lets you select the system file you wish to administer.
    │                         You can also name the current transaction program profile, side
    │                         information, or data base token system files.
    │
    ├ICQASE22                 A menu that lets you select the model data set using the criteria entered
    │                         in panel ICQASE02.
    │
    ├ICQASE98                 A data entry panel on which you can name a new data base token.
    │  │
    │  └ICQASRR0              The REXX exec that processes your requests for changing data base tokens.
    │                         (See Figure 141)
    │
    ├ICQASRS0                 The REXX exec that processes your request for side information and
    │  │                      processes file allocation.
    │  │
    │  └ICQASRP1              The REXX exec that displays the side information list panel.
    │     │
    │     └ICQASE72           A data entry panel that lets you specify actions you wish to take on side
    │        │                information in the current system file.  You may change the current system
    │        │                file name.  You may select to edit, browse, copy, delete, and add side
    │        │                information.
    │        │
    │        ├ICQASE22        A menu that lets you select the model data set using the criteria entered
    │        │                in panel ICQASE72.
    │        │
    │        └ICQASRP2        The REXX exec that displays side information.
    │           │
    │           ├ICQASE74     A data entry panel that allows you to add side information.  You can
    │           │  │          specify the symbolic destination name, transaction program name, mode name,
    │           │  │          and partner LU name.
    │           │  │
    │           │  └ICQASRR0  The REXX exec that processes your requests for adding side information.
    │           │             (See Figure 141)
    │           │
    │           ├ICQASE76     A data entry panel that allows you to copy side information from one system
    │           │  │          file to another.  Symbolic destination name, transaction program name, the
    │           │  │          mode name, and the partner LU name can be changed.  You can change the "to"
    │           │  │          system file.
    │           │  │
    │           │  └ICQASRR0  The REXX exec that processes your requests for copying side information.
    │           │             (See Figure 141)
    │           │
    │           ├ICQASE80     A data entry panel that allows you to edit side information.  You can edit
    │           │  │          the transaction program name, mode name, and partner LU name.
    │           │  │
    │           │  └ICQASRR0  The REXX exec that processes your requests for editing side information.
    │           │             (See Figure 141)
    │           │
    │           ├ICQASE84     A panel that displays the side information that you selected to browse.
    │           │
    │           └ICQASE84     A panel that displays the side information that you selected to delete.
    │              │
    │              └ICQASRR0  The REXX exec that processes your requests for deleting side information.
    │                         (See Figure 141)
    │
    └ICQASRT0           The REXX exec that processes your request for transaction program profile
       │                and processes file allocation.
       │
       └ICQASRP1        The REXX exec that displays the transaction program profile list panel.
          │
          └ICQASE04     A menu panel that allows you to select the type of administration to be
             │          performed on a transaction program profile for the current system file.
             │          You can also change the current system file.
             │
             ├ICQASE22  A menu that lets you select the model data set using the criteria
             │          entered in panel ICQASE04.
             │
             └ICQASRP2  The data entry panel that displays transaction program profile information.
                │
```

```
ICQASRR0                The REXX exec that processes an APPC/MVS administration utility request.
 │
 │
 ├── ICQASRF0       The REXX exec that calls ICQASLIO in an authorized state.
 │
 ├── ICQASLC0       The module that calls the APPC/MVS administration utility in an authorized state.
 │     │
 │     └── ICQASLI0 The module that displays and processes the APPC/MVS administration utility
 │                  request through an authorized invocation of the APPC/MVS administration utility.
 │
 └── ICQASRE0       The REXX exec that displays and processes results of APPC/MVS administration utility
                    extracts.
```

Figure 143. APPC/MVS administration utility request application and REXX exec hierarchy

Menu, data entry panel, and help panel associations

Table 129 shows the relationship between each data entry panel or menu and its associated help panel(s). If you customize a particular panel or menu, refer to the figure to find which help panels are affected. Read the help panels to see whether the changes you made to the functional panels require corresponding changes to the help panels.

Table 129. Menu, data entry panel, and help panel cross reference

Function	Data entry panel or menu	Help panels
Administrator NEWS	ICQANE00	ICQBN000 – ICQBN002
	ICQANE10	ICQBN100 – ICQBN101
	ICQANE20	ICQBN200
	ICQANE40	ICQBN400 – ICQBN402
	ICQANE60	ICQBN600 – ICQBN601
	ICQANE90	ICQBN900 – ICQBN905
	ICQCNE20	ICQDN200
	ICQCNE30	ICQDN300
Administrator NAMES	ICQAAM00	ICQBA000 – ICQBA001
	ICQAAE10	ICQBA100 – ICQBA102
	ICQAAE11	ICQBA110 – ICQBA111
	ICQAAE13	ICQBA130 – ICQBA131
	ICQAAE15	ICQBA150
	ICQAAE20	ICQDA200 – ICQDA201
	ICQAAE21	ICQBA210 – ICQBA211
	ICQAAE22	ICQBA220 – ICQBA221
	ICQAAE27	ICQBA270 – ICQBA272
	ICQAAE30	ICQBA300 – ICQBA302
	ICQAEE50	ICQBE500
	ICQCAE12	ICQDA120
	ICQCAE14	ICQDA140
	ICQCAE15	ICQDA150
	ICQCAE16	ICQDA160
	ICQCAE23	ICQDA230
	ICQCAE24	ICQDA240
	ICQCAE25	ICQDA250
	ICQCAE26	ICQDA260
	ICQCAE41	ICQDA410 – ICQDA411

Table 129. Menu, data entry panel, and help panel cross reference (continued)

Function	Data entry panel or menu	Help panels
ENROLL	ICQAAE13	ICQBA130
	ICQAAE15	ICQBA150
	ICQAEE10	ICQBE100 – ICQBE101
	ICQAEE11	ICQBE110
	ICQAEE40	ICQBE400 – ICQBE407, ICQBE409
	ICQAEE41	ICQBE401 – ICQBE410
	ICQAEE50	ICQBE500
	ICQAEMC0	ICQBEC00 – ICQBEC01
	ICQCAE12	ICQDA120
	ICQCAE14	ICQDA140
User Types	ICQADE00	ICQBD000 – ICQBD002
	ICQADE01	ICQBD010
	ICQADE02	ICQBD020
	ICQADE03	ICQBD030
	ICQADE04	ICQBD040 – ICQBD049, ICQBD04A, ICQBD04B
	ICQADE05	ICQBD050 – ICQBD052
	ICQADE06	ICQBD060 – ICQBD069, ICQBD06A, ICQBD06B
	ICQADE07	ICQBD070 – ICQBD072
	ICQADE08	ICQBD080 – ICQBD089, ICQBD08A, ICQBD08B, ICQBD08C, ICQBD08D
	ICQADE09	ICQBD090 – ICQBD091
	ICQADE10	ICQBD100 – ICQBD109, ICQBD10A, ICQBD10B, ICQBD10C, ICQBD10D
	ICQADE11	ICQBD110 – ICQBD111
	ICQADE12	ICQBD120
	ICQADE13	ICQBD130
	ICQADE14	ICQBD140
	ICQADE15	ICQBD150
	ICQADM16	ICQBD160 – ICQBD161
	ICQADM17	ICQBD170 – ICQBD171

Table 129. Menu, data entry panel, and help panel cross reference (continued)

Function	Data entry panel or menu	Help panels
System Defaults Application Manager	ICQAMED1	ICQBMD10
	ICQAMED4	ICQBMD40
	ICQAMED5	ICQBMD50
	ICQAME10	ICQBM100 – ICQBM103
	ICQAME13	ICQBM130 – ICQBM131
	ICQAME16	ICQBM160
	ICQAME17	ICQBM170 – ICQBM171
	ICQAME19	ICQBM190 – ICQBM191
	ICQAME20	ICQBM200 – ICQBM202
	ICQAME21	ICQBM210 – ICQBM213
	ICQAME24	ICQBM240 – ICQBM241
	ICQAME26	ICQBM260 – ICQBM263
	ICQAME30	ICQBM300 – ICQBM302
	ICQAME31	ICQBM310 – ICQBM312
	ICQAME32	ICQBM320 – ICQBM323
	ICQAME33	ICQBM330 – ICQBM334
	ICQAME34	ICQBM340 – ICQBM341
	ICQAME35	ICQBM350 – ICQBM352
	ICQAME36	ICQBM360 – ICQBM361
	ICQAME37	ICQBM370
	ICQAME38	ICQBM380 – ICQBM381
	ICQAME39	ICQBM390 – ICQBM391
	ICQAME40	ICQBM400 – ICQBM401
	ICQAME41	ICQBM410 – ICQBM411
	ICQAME42	ICQBM420 – ICQBM421
	ICQAME50	ICQBM500 – ICQBM502
	ICQAME51	ICQBM510 – ICQBM513
	ICQAME60	ICQBM600
	ICQAME61	ICQBM610
	ICQAME70	ICQBM700
	ICQAME71	ICQBM710 – ICQBM711
	ICQAME80	ICQBM800 – ICQBM803
	ICQAME82	ICQBM820 – ICQBM821
	ICQAME85	ICQBM850 – ICQBM853
	ICQAME87	ICQBM870
	ICQAME88	ICQBM880 – ICQBM833
	ICQAME89	ICQBM890
	ICQAME8A	ICQBM8A0 – ICQBM8A1
	ICQAME8B	ICQBM8B0 – ICQBM8B1
	ICQAME9B	ICQBM9B0
	ICQAME9C	ICQBM9C0
	ICQAME90	ICQBM900 – ICQBM901
	ICQAME91	ICQBM910
	ICQAME92	ICQBM920
	ICQAME93	ICQBM930
	ICQAME94	ICQBM940
	ICQAME95	ICQBM950 – ICQBM951
	ICQAME96	ICQBM960
	ICQAME97	ICQBM970 – ICQBM971
	ICQAME98	ICQBM980
	ICQAME99	ICQBM990
	ICQAMEA1	ICQBMA10
	ICQAMEA2	ICQBMA20
	ICQAMEA3	ICQBMA30

Table 129. Menu, data entry panel, and help panel cross reference (continued)

Function	Data entry panel or menu	Help panels
System Defaults Printer Defaults	ICQAPE00	ICQBP000
	ICQAPE10	ICQBP100
	ICQAPE30	ICQBP300 – ICQBP305
	ICQAPE40	ICQBP400
	ICQAPE41	ICQBP410
	ICQAPE42	ICQBP420
	ICQAPE50	ICQBP500 – ICQBP50A
	ICQAPE51	ICQBP510 – ICQBP516
	ICQAPE52	ICQBP520 – ICQBP527
	ICQAPE53	ICQBP530 – ICQBP536
	ICQAPE54	ICQBP540 – ICQBP549
	ICQAPE55	ICQBP550 – ICQBP551
	ICQAPE56	ICQBP560 – ICQBP561
	ICQAPE57	ICQBP570 – ICQBP57B
	ICQAPE60	ICQBP600
	ICQAPE70	ICQBP700 – ICQBP705
	ICQAPE80	ICQBP800 – ICQBP830
	ICQAPE90	ICQBP900 – ICQBP902
	ICQAPM20	ICQBP200 – ICQBP201
Administrator Courses (Computer-based training)	ICQABE10	ICQBB100 – ICQBB107
	ICQABE20	ICQBB200 – ICQBB206
	ICQABE40	ICQBB400 – ICQBB405
	ICQABE50	ICQBB500 – ICQBB506
	ICQABE70	ICQBB700 – ICQBB701
	ICQABE90	ICQBB900 – ICQBB901
	ICQABEA0	ICQBBA00 – ICQBBA01
	ICQABEB0	ICQBBB00
	ICQABEC0	ICQBBC00 – ICQBBC04
	ICQABED0	ICQBBD00
	ICQABEE0	ICQBBE00 – ICQBBE06
	ICQABEF0	ICQBBF00 – ICQBBF06
	ICQABEG0	ICQBBG00 – ICQBBG01
	ICQABEH0	ICQBBH00 – ICQBBH02
	ICQABEI0	ICQBBI00 – ICQBBI01
	ICQABEJ0	ICQBBJ00 – ICQBBJ04
	ICQABEK0	ICQBBK00
	ICQABEM0	ICQBBM00
	ICQABEQ0	ICQBBQ00 – ICQBBQ03
	ICQABER0	ICQBBR00 – ICQBBR01
	ICQABES0	ICQBBS00 – ICQBBS02
	ICQABET0	ICQBBT00 – ICQBBT04
	ICQABEU0	ICQBBU00 – ICQBBU05
	ICQABEV0	ICQBBV00 – ICQBBV01
	ICQABEW0	ICQBBW00
	ICQABM00	ICQBB000 – ICQBB003
	ICQABM30	ICQBB300 – ICQBB301
	ICQCBE20	ICQDB200
	ICQCNE30	ICQDN300
Administrator problem reporting services	ICQGAM01	ICQHA010
User NEWS	ICQCNE00	ICQDN000 – ICQDN001
	ICQCNE10	ICQDN100 – ICQDN101
	ICQCNE20	ICQDN200
	ICQCNE30	ICQDN300

Table 129. Menu, data entry panel, and help panel cross reference (continued)

Function	Data entry panel or menu	Help panels
User NAMES	ICQCAE10	ICQDA100 – ICQDA101
	ICQCAE11	ICQDA110 – ICQDA112
	ICQCAE12	ICQDA120
	ICQCAE13	ICQDA130
	ICQCAE14	ICQDA140
	ICQCAE15	ICQDA150
	ICQCAE16	ICQDA160
	ICQCAE17	ICQDA170 – ICQDA171
	ICQCAE20	ICQDA200 – ICQDA201
	ICQCAE21	ICQDA210 – ICQDA213
	ICQCAE22	ICQDA220 – ICQDA221
	ICQCAE23	ICQDA230
	ICQCAE24	ICQDA240
	ICQCAE25	ICQDA250
	ICQCAE26	ICQDA260
	ICQCAE27	ICQDA270 – ICQDA272
	ICQCAM00	ICQDA000
Printing Services	ICQGCM04	ICQHC040
APLDI-II	ICQCIE10	ICQDI100 – ICQDI104
	ICQCIE20	ICQDI200 – ICQDI203
	ICQCIE30	ICQDI300 – ICQDI304
	ICQCIE32	ICQDI320 – ICQDI322
	ICQCIE33	ICQDI330 – ICQDI332
	ICQCIE34	ICQDI340 – ICQDI343
	ICQCIE35	ICQDI350 – ICQDI353
	ICQCIE40	ICQDI400 – ICQDI404
	ICQCIE50	ICQDI500 – ICQDI504
	ICQCIE51	ICQDI510 – ICQDI514
	ICQCIE52	ICQDI520 – ICQDI524
	ICQCIE53	ICQDI530 – ICQDI531
	ICQCIE60	ICQDI600 – ICQDI606
	ICQCIE61	ICQDI610 – ICQDI612
	ICQCIE62	ICQDI620 – ICQDI622
	ICQCIM00	ICQDI000 – ICQDI004
	ICQGCE13	ICQHC130
	ICQGCE15	ICQHC130
	ICQGCE17	ICQHC130
	ICQGCE19	ICQHC190
	ICQGCE40	ICQHC400 – ICQHC401
	ICQGCE41	ICQHC410 – ICQHC411
ADRS	ICQCRE10	ICQDR100 – ICQDR101
	ICQCRE11	ICQDR110 – ICQDR111
	ICQCRE20	ICQDR200 – ICQDR201
	ICQCRE30	ICQDR300 – ICQDR303
	ICQCRM00	ICQDR000 – ICQDR001
	ICQGCE13	ICQHC130
	ICQGCE15	ICQHC130
	ICQGCE19	ICQHC190
	ICQGCE40	ICQHC400 – ICQHC401
	ICQGCE41	ICQHC410 – ICQHC411
Chart creation services	ICQCGE00	ICQDG000 – ICQDG005
	ICQCGE01	ICQDG010 – ICQDG013
	ICQCGE02	ICQDG020 – ICQDG023
	ICQCGE03	ICQDG030 – ICQDG034
	ICQGCE17	ICQHC130
	ICQGCE19	ICQHC190

Table 129. Menu, data entry panel, and help panel cross reference (continued)

Function	Data entry panel or menu	Help panels
User COURSES(Computer-based training)	ICQCBE00 ICQCBE10 ICQCBE20 ICQCBE30 ICQCBE40 ICQCBE50 ICQCBE60 ICQCBE70 ICQCNE30	ICQDB000 – ICQDB005 ICQDB100 – ICQDB107 ICQDB200 ICQDB300 – ICQDB301 ICQDB400 – ICQDB401 ICQDB500 – ICQDB501 ICQDB600 ICQDB700 ICQDN300
Programmer services	ICQCLM00 ICQCLE10 ICQCLE11 ICQCLE20 ICQCLE21 ICQCLE22 ICQCLE23 ICQCRE20 ICQGCE15 ICQGCE19	ICQDL000 – ICQDL001 (VS APL) and ICQDL00A – ICQDL00B (APL2) ICQDL100 – ICQDL103, ICQDL105, ICQDL107, ICQDL109, ICQDL10B, ICQDL10D, ICQDL10F, ICQDL10H, ICQDL10J, ICQDL10L ICQDL110 – ICQDL113 ICQDL200 – ICQDL203, ICQDL205, ICQDL207, ICQDL209, ICQDL20B, ICQDL20D, ICQDL20F, ICQDL20H, ICQDL20I, ICQDL20J ICQDL210 – ICQDL212 ICQDL220 – ICQDL221 ICQDL230 – ICQDL234 ICQDR200 – ICQDR201 ICQHC130 ICQHC190
User report problems	ICQGCM07	ICQHC070
Utility services	ICQGCM06 ICQGCE35 ICQGCE50	ICQHC060 ICQHC350 – ICQHC353 ICQHC500

Table 129. Menu, data entry panel, and help panel cross reference (continued)

Function	Data entry panel or menu	Help panels
APPC/MVS administration	ICQASE02	ICQBS020
	ICQASE04	ICQBS040 – ICQBS041 – ICQBS012
	ICQASE08	ICQBS080 – ICQBS081 – ICQBS010
	ICQASE10	ICQBS100 – ICQBS101
	ICQASE12	ICQBS120 – ICQBS121
	ICQASE16	ICQBS160
	ICQASE20	ICQBS200 – ICQBS201
	ICQASE22	ICQBS220
	ICQASE24	ICQBS240
	ICQASE28	ICQBS280 – ICQBS281 – ICQBS010
	ICQASE30	ICQBS300 – ICQBS301
	ICQASE32	ICQBS320 – ICQBS321 – ICQBS322
	ICQASE36	ICQBS360
	ICQASE38	ICQBS380
	ICQASE40	ICQBS400 – ICQBS401
	ICQASE44	ICQBS440
	ICQASE46	ICQBS460
	ICQASE48	ICQBS480
	ICQASE52	ICQBS520
	ICQASE54	ICQBS540
	ICQASE56	ICQBS560
	ICQASE60	ICQBS600
	ICQASE64	ICQBS640
	ICQASE66	ICQBS660
	ICQASE68	ICQBS680 – ICQBS010
	ICQASE70	ICQBS700
	ICQASE72	ICQBS720 – ICQBS721
	ICQASE74	ICQBS740 – ICQBS741 – ICQBS010
	ICQASE76	ICQBS760 – ICQBS761 – ICQBS010
	ICQASE80	ICQBS800 – ICQBS010
	ICQASE84	ICQBS840
	ICQASE90	ICQBS900
	ICQASE94	ICQBS940
	ICQASE96	ICQBS960
	ICQASE98	ICQBS980 – ICQBS981

Chapter 49. Diagnosing problems with the Information Center Facility

This section describes techniques you can use to diagnose problems in the Information Center Facility.

You can find charts of the Information Center Facility CLISTs, REXX execs, and panels in “Application,panel, CLIST, and REXX exec hierarchy” on page 659. For additional information about error messages, see the on-line message help or *z/OS TSO/E Messages*. For information about CLISTs, see *z/OS TSO/E CLISTs*. For information about REXX execs, see *z/OS TSO/E REXX Reference*. For information about ISPF dialog and table services, see *z/OS ISPF Services Guide*.

Displaying the panel ID

If you have a problem with a panel in the Information Center Facility, you can display the ID of the panel by entering the command “PANELID” or “PANELID ON” on the Command or Option line of the panel. After you enter this command, all the panels displayed show the panel identifier in the upper left corner of the screen. You can suppress the display of the panel ID by entering “PANELID OFF”.

Activating and using trace

The Information Center Facility is written in the TSO/E CLIST and REXX languages. You can use the trace commands provided with the Information Center Facility to diagnose problems in the facility's CLISTs and REXX execs. The commands provide three levels of tracing:

- Level 1 - activated by the TRACE1 command
- Level 2 - activated by the TRACE2 command
- Level 3 - activated by the TRACE3 command

You can turn Information Center Facility tracing off using the TRACEOFF command.

When a CLIST is invoked for execution and Information Center Facilitytracing is active, the CLIST writes its name to the screen using a CLIST WRITE statement. The information appears as follows:

```
************************
**  CLIST=clistname  ***
************************
```

When a REXX exec is invoked for execution and Information Center Facility tracing is active, the exec writes its name to the screen using a REXX SAY instruction. The information appears as follows:

```
***************************
**  REXX exec=execname  ***
***************************
```

The ISPF dialog service LOG is issued at the beginning and termination of a CLIST/REXX exec. The following message is written to the ISPF log when a CLIST/REXX exec begins execution:

```
CLIST or REXX exec membername beginning execution.  ICQGA000
```

At the termination of a CLIST/REXX exec, the following message is written:

```
CLIST/REXX exec membername exiting; final condition code n.  ICQGA001
```

The executing CLIST/REXX exec determines the final condition code.

Note: You can enter the TRACE commands on the Option line of any Information Center Facility menu panel except the following:
- ICQABM30
- ICQCLM00

TRACE1 command — level 1

Level 1 traces the control flow between (or among) nested CLISTs/REXX execs and shows the order of CLIST/REXX exec invocation and execution. As each CLIST/REXX exec is invoked, it writes its name to the screen as described previously.

To activate level 1 tracing, enter TRACE1 on the Option line of an Information Center Facility menu panel. The menu panel is redisplayed with the command still on the Option line, and the following message is displayed:

```
CLIST or REXX exec tracing will be at level 1.  ICQGA006
```

TRACE2 command — level 2

Level 2 provides the same information as level 1. In addition, it displays, for all the nested CLISTs, each CLIST statement and TSO/E command and subcommand before execution. For all the nested REXX execs, it displays each clause before execution.

Each CLIST executes with the following options in effect:

```
CONTROL SYMLIST CONLIST LIST MSG
```

SYMLIST
Display each executable statement before it is scanned for symbolic substitution. Executable statements include CLIST statements, and TSO/E commands and subcommands.

CONLIST
Display CLIST statements after symbolic substitution, but before execution.

LIST Display TSO/E commands and subcommands after symbolic substitution, but before execution.

MSG Display informational messages from the CLIST.

Each REXX exec executes with the following option in effect:

```
TRACE INTERMEDIATES
```

INTERMEDIATES
Display all clauses before execution and display intermediate results during evaluation of expressions and substituted names. Clauses include null clauses, labels, assignments, keyword instructions, and commands.

To activate level 2 tracing, enter TRACE2 on the Option line of an Information Center Facility menu panel. The menu panel is redisplayed with the command still on the Option line, and the following message is displayed:

```
CLIST or REXX exec tracing will be at level 2.  ICQGC006
```

TRACE3 command — level 3

For all CLISTs/REXX execs, level 3 tracing provides the same trace as level 1. In addition, for a single explicitly-named CLIST/REXX exec, level 3 tracing provides the same trace as level 2. The explicitly-named CLIST/REXX exec executes with the options described for the level 2 trace.

To activate level 3 tracing, enter TRACE3.*membername* on the Option line of an Information Center Facility menu panel, where *membername* is the name of a nested CLIST/REXX exec. The menu panel is redisplayed with the command still on the Option line, and the following message is displayed:

```
CLIST/REXX exec member membername will be traced on the screen.  ICQGA005
```

Entering the TRACE3 command without specifying *membername* turns off all tracing. The following message is displayed:

```
CLIST or REXX exec member name required.  TRACEOFF is set.  ICQGA004
```

Deactivating trace

To deactivate trace, enter TRACEOFF on the Option line of an Information Center Facility menu panel. The menu panel is redisplayed with the command still on the Option line, and the following message is displayed:

```
CLIST or REXX exec trace is turned off.  ICQGA003
```

If you enter TRACEOFF when trace is not active, the following message is displayed:

```
CLIST or REXX exec trace is not active.  ICQGA002
```

Part 10. Reference

This part contains the following reference information:

- Chapter 50, "Overview of facilities for customizing TSO/E," on page 699 contains a summary of the facilities you can use to customize TSO/E. It includes TSO/E macros, exits, initialization statements, data sets, and SYS1.PARMLIB members for: ISPF/PDF, JES2, JES3, SMF, MVS, and VTAM. The TSO/E exits are not included in this section. For an overview of the TSO/E exits, see Chapter 2, "Writing exit routines," on page 23.
- Chapter 51, "Macro syntax," on page 707 provides the syntax of the following TSO/E macros:
  - IKJBCAST
  - IKJEDIT
  - IKJIFRIF
  - IKJTSO
  - INMEND
  - INMNODE
  - INMXP

Chapter 50. Overview of facilities for customizing TSO/E

The following sections contain tables that summarize the facilities you can use to customize TSO/E. Each table contains the name of the facility, a brief description of why you might want to use it, and a reference to additional information.

ISPF/PDF macro statements and exits

Table 130 is a summary of ISPF/PDF macro statements and exits you can use to customize the use of TSO/E from ISPF/PDF.

Table 130. ISPF/PDF macro statements and exits

ISPF/PDF macro or exit	Description	Reference
ISPMTCM macro statement	Use to specify which TSO/E commands users will be able to use from ISPF/PDF panels.	*z/OS ISPF Planning and Customizing*
Command start and stop exits	Use to monitor TSO/E commands users issue from ISPF/PDF panels, and to restrict individuals from using certain commands.	*z/OS ISPF Planning and Customizing*

JES2 exits and initialization statements

Table 131 and Table 132 on page 700 summarize some of the JES2 exits and initialization statements you can use to customize TSO/E jobs.

Table 131. JES2 exits

JES2 exit	Description	Reference
Exit 2 (Job statement scan)	Use to tailor JOB JCL statements, supply additional JOB statement parameters, and modify job-related control blocks.	*z/OS JES2 Installation Exits*
Exit 3 (Job statement accounting field scan)	Use to tailor JOB JCL accounting information, supply additional accounting field information, and modify job-related control blocks. (Although you can use Exit 2 to tailor accounting information, you can more easily access the information from Exit 3.)	*z/OS JES2 Installation Exits*
Exit 13 (IDTF screening and notification)	Use to screen incoming files as they arrive at the receiver's network node. The exit can delete the file, route it to another user, or allow it to be sent to the target addressee.	*z/OS JES2 Installation Exits*
Exit 20 (End of job input)	Use to cancel a job before it starts processing.	*z/OS JES2 Installation Exits*
Exit 22 (Cancel/Status)	Use to tailor the way the TSO/E STATUS and CANCEL commands work.	*z/OS JES2 Installation Exits*

Table 131. JES2 exits (continued)

JES2 exit	Description	Reference
Exit 38 (TSO/E RECEIVE Authorization)	Use during data set selection to determine whether the user has authority to receive the data.	*z/OS JES2 Installation Exits*

Table 132. JES2 initialization statements

JES2 initialization statement	Description	Reference
INTRDR	Use to specify the characteristics to be used for jobs submitted through JES2 internal readers. You can specify defaults for processing jobs submitted by TSO/E users.	*z/OS JES2 Initialization and Tuning Reference*
OUTCLASS	Use to specify the default characteristics that are to be used for processing SYSOUT data sets. You can specify: • Whether data sets are to be held for processing by the TSO/E OUTPUT command. • That the output class is to be punched (a requirement when using the TSO/E TRANSMIT and RECEIVE commands).	*z/OS JES2 Initialization and Tuning Reference*
STCCLASS	Use to specify the default characteristics to be associated with all started tasks. If your installation uses the TSO/E TRANSMIT and RECEIVE commands, you need to specify that SYSOUT data is to be written for jobs executed in time sharing classes.	*z/OS JES2 Initialization and Tuning Reference*
TSUCLASS	Use to specify default job processing characteristics that will be used for a user's TSO/E session. You can: • Specify the default message class that will be used for TSO/E session logs. • Identify the library that contains TSO/E logon procedures. • Specify that SYSOUT data is to be written for jobs executed in time sharing classes (a requirement when using the TSO/E TRANSMIT and RECEIVE commands).	*z/OS JES2 Initialization and Tuning Reference*

JES3 exits and initialization statements

Table 133 and Table 134 summarize some of the JES3 exits and initialization statements you can use to customize TSO/E jobs.

Table 133. JES3 exits

JES3 exit	Description	Reference
IATUX28	Use to tailor JOB JCL statements, supply additional JOB statement parameters, and modify job-related control blocks.	*z/OS JES3 Customization*
IATUX30	Use to tailor the use of the TSO/E OUTPUT, STATUS, and CANCEL commands.	*z/OS JES3 Customization*
IATUX33	Use to tailor JCL EXEC statements, JCL comment statements, and JES3 control statements. You can supply additional statements, and modify related control blocks.	*z/OS JES3 Customization*
IATUX34	Use to tailor JCL DD statements, and JCL comment statements. You can supply additional statements, and modify related control blocks.	*z/OS JES3 Customization*
IATUX38	Use to change the SYSOUT class for local SYSOUT data sets.	*z/OS JES3 Customization*
IATUX42	Use to accept or reject data sent via the TSO/E TRANSMIT command.	*z/OS JES3 Customization*
IATUX44	Use to tailor JCL statements other than JOB, EXEC, or DD statements.	*z/OS JES3 Customization*
IATUX60	Use to determine whether a data set that was to be received by a TSO/E user should be deleted or kept, if the TSO/E user can never log on to TSO/E with the proper security label to ever receive the data set.	*z/OS JES3 Customization*

Table 134. JES3 initialization statements

JES3 initialization statement	Description	Reference
SYSOUT	Use to specify SYSOUT class characteristics. If using the TSO/E TRANSMIT and RECEIVE commands, the output class must be punched.	*z/OS JES3 Initialization and Tuning Reference*
CIPARMS	Use to specify a converter/interpreter options list, which defines defaults for JCL keywords, such as whether users must specify a programmer name or account number in JOB statements.	*z/OS JES3 Initialization and Tuning Reference*

Table 134. JES3 initialization statements (continued)

JES3 initialization statement	Description	Reference
STANDARDS	Use to associate job processing characteristics with users' TSO/E sessions, and jobs submitted by users. You can specify: • A set of defaults (a converter/ interpreter options list) that will be used for users' TSO/E sessions, and for jobs submitted by TSO/E users. • The library containing the TSO/E logon procedures.	*z/OS JES3 Initialization and Tuning Reference*

MVS data sets

Table 135 is a summary of MVS data sets that contain TSO/E information.

Table 135. MVS data sets

MVS data sets	Description	Reference
Broadcast data set	Contains notices (messages sent to all users of a system), and mail (messages sent to individual users). You can change the amount of space reserved in the broadcast data set for notices using the IKJBCAST macro statement.	Part 4, “Maintaining the UADS, RACF data base, and broadcast data set,” on page 193
SYS1.CMDLIB	Contains service routines, utility programs, and TSO/E command processors. You may want to move TSO/E commands to SYS1.LPALIB, possibly improving the performance of TSO/E.	None
SYS1.HELP	Contains help information about TSO/E commands and messages.	*z/OS TSO/E Programming Guide*
SYS1.LNKLIB	Contains RACF commands, the data set initialization program, report writer, and utilities.	None
SYS1.LPALIB	Contains some TSO/E commands, such as MVSSERV. It also contains RACF-related information and installation-written exit routines.	None
SYS1.MACLIB	Contains the macro definitions for supervisor and data management macro instructions. It also contains RACF and TSO/E macros, such as IKJBCAST.	None
SYS1.PARMLIB	Contains IBM-supplied and installation-created lists of system parameter values. It contains parameters you can use to tailor the logon process, and the way TSO/E works with VTAM.	None

Table 135. MVS data sets (continued)

MVS data sets	Description	Reference
SYS1.PROCLIB	Contains the cataloged procedures used to perform system functions. By default, it contains all TSO/E logon procedures. You may want to use your own procedure library. For more information, see “Writing a logon procedure” on page 78.	None
RACF data base	Contains profiles for every entity—users, data sets, and groups—defined to RACF. The profiles describe the attributes and the authorities of each of the entities. If you use RACF to add users, logon information is stored in the user's profile.	*z/OS Security Server RACF Security Administrator's Guide* Part 4, “Maintaining the UADS, RACF data base, and broadcast data set,” on page 193
SYS1.UADS	If you use the TSO/E ACCOUNT command to add and maintain users, SYS1.UADS contains a list of TSO/E users and information about them, such as account numbers, logon procedures, and commands they can use.	*z/OS TSO/E Administration* Part 4, “Maintaining the UADS, RACF data base, and broadcast data set,” on page 193

SMF exits

Table 136 is a summary of the SMF exits you can use to customize TSO/E jobs.

Table 136. SMF exits

SMF exits	Description	Reference
IEFUJV	Use to validate or tailor JCL statements, and to cancel jobs before they start processing.	*z/OS MVS Installation Exits*
IEFUSO	Use to either cancel a job when the number of records written to a SYSOUT data set exceeds the limit, or extend the limit and allow the job to continue processing.	*z/OS MVS Installation Exits*
IEFUTL	Use to either cancel or extend a job after a time limit, such as the continuous wait time limit, has expired.	*z/OS MVS Installation Exits*

SYS1.PARMLIB members

Table 137 is a summary of SYS1.PARMLIB members you can use to customize TSO/E.

Table 137. SYS1.PARMLIB members

Member name	Description	Reference
IEAICSxx	Use to associate: • Users with control performance groups • Users and commands with reporting performance groups.	*z/OS MVS Initialization and Tuning Reference*
IEAIPSxx	Use to tailor the way SRM manages work, such as TSO/E transactions, at your installation.	*z/OS MVS Initialization and Tuning Reference*
IEASYSxx	Use to tune your system and, for example, specify the total number of address spaces in your system.	*z/OS MVS Initialization and Tuning Reference*
SMFPRMxx	Use to tailor the way SMF records data. You can specify whether data will be recorded for TSO/E commands, and the maximum amount of time a session can remain idle before being canceled.	*z/OS MVS Initialization and Tuning Reference*
IKJTSOxx	Use to specify: • The authorized commands that will be used at your installation • The commands and programs to run on the command/program invocation platform • The commands not supported in the background • The name of the broadcast data set and associated processing options • The defaults for SEND, OPERATOR SEND, LISTBC, TRANSMIT, RECEIVE, ALLOCATE, CONSOLE, HELP, and TEST processing.	*z/OS MVS Initialization and Tuning Reference*
CONSOLxx	Use to define the characteristics of each MCS console in your system configuration, such as the console name, MVS command authority, and routing codes assignment.	*z/OS MVS Initialization and Tuning Reference*
TSOKEYxx	Use to tailor the values VTAM uses to initialize users' address spaces. You can specify the number of users who can be logged on to TSO/E, and the amount of time address spaces that will remain available after users have been disconnected.	*z/OS MVS Initialization and Tuning Reference*

TSO/E macro statements

Table 138 is a summary of TSO/E macro statements you can use to customize TSO/E.

Table 138. TSO/E macro statements

TSO/E macro statement	Description	Reference
IKJBCAST	Use to change the amount of space reserved in the broadcast data set for notices.	“IKJBCAST macro” on page 709
IKJEDIT	Use to tailor default characteristics that are used for data sets created via the EDIT command.	“IKJEDIT macro” on page 709
IKJIFRIF	Use to add, change, or delete information about users in the broadcast data set.	“IKJIFRIF macro” on page 714
IKJTSO	Use to tailor: • How often the ‘Logon Proceeding’ message is issued. • The number of unsuccessful attempts users can make to enter information in response to logon prompts.	“IKJTSO macro” on page 720
INMEND	Use to mark the end of a list of INMNODE macros.	“INMEND macro” on page 721
INMNODE	In setting up the TRANSMIT and RECEIVE environment, use INMNODE to associate system identifiers with network node names.	“INMNODE macro” on page 721
INMXP	In setting up the TRANSMIT and RECEIVE environment, use INMXP to specify the installation controls, and the defaults for certain parameters on the TSO/E TRANSMIT and RECEIVE commands. For example, INMXP can specify the default output class for transmitted data, and the default log data set name.	“INMXP macro” on page 722

VTAM exits

Table 139 is a summary of VTAM exits you can use to customize TSO/E.

Table 139. VTAM exits

VTAM exits	Description	Reference
IKTGETXT	Use to edit input data.	*z/OS Communications Server: SNA Resource Definition Reference*
IKTIDSX1	Use to replace or supplement IBM-supplied output editing for 3270 terminals.	*z/OS Communications Server: SNA Resource Definition Reference*

Table 139. VTAM exits (continued)

VTAM exits	Description	Reference
IKTIDSX2	Use to supplement IBM-supplied input editing for 3270 terminals.	*z/OS Communications Server: SNA Resource Definition Reference*
IKTIDSX3	Use to replace IBM-supplied attention handling for 3270 (LU0) terminals.	*z/OS Communications Server: SNA Resource Definition Reference*
IKTIDSX4	Use to replace or supplement IBM-supplied input editing for 3270 terminals.	*z/OS Communications Server: SNA Resource Definition Reference*
IKTINX2	Use to initialize installation-written I/O managers.	*z/OS Communications Server: SNA Resource Definition Reference*
IKTRTX1	Use to replace or supplement IBM-supplied output editing for 3767/3770 and 2741 terminals.	*z/OS Communications Server: SNA Resource Definition Reference*
IKTRTX2	Use to supplement IBM-supplied input editing for 3767/3770, 2741, WTTY or TWX terminals.	*z/OS Communications Server: SNA Resource Definition Reference*
IKTRTX3	Use to replace IBM-supplied attention handling for 3767/3770 (LU1) terminals.	*z/OS Communications Server: SNA Resource Definition Reference*
IKTRTX4	Use to replace or supplement IBM-supplied input editing for 3767/3770 (LU1) terminals.	*z/OS Communications Server: SNA Resource Definition Reference*
IKTWTX1	Use to replace or supplement IBM-supplied output editing for TWX and WTTY terminals.	*z/OS Communications Server: SNA Resource Definition Reference*

VTAM statements

Table 140 is a summary of the VTAM statements you can use to define TSO/E to VTAM.

Table 140. VTAM statements

VTAM statements	Description	Reference
LOGCHAR macro statement	Use to define the LOGON command you want to use at your installation.	*z/OS Communications Server: SNA Customization*
APPL definition statement	Use to define TCAS and TSO/E address spaces to VTAM.	*z/OS Communications Server: SNA Resource Definition Reference*

Chapter 51. Macro syntax

This section describes the syntax of the following macros:

- IKJBCAST

  Use the IKJBCAST macro to specify the maximum number of messages that TSO/E can store in the notices section of the broadcast data set.
- IKJEDIT

  Use the IKJEDIT macro to specify the characteristics of EDIT data set types.
- IKJIFRIF

  Use the IKJIFRIF macro to add, delete, or change user IDs in the broadcast data set.
- IKJTSO

  Use the IKJTSO macro to specify certain limits related to LOGON processing.
- INMEND, INMNODE, INMXP

  Use the INMEND, INMNODE, and INMXP macros to code the installation options CSECT (INMXPARM) for the TSO/E TRANSMIT and RECEIVE commands. Code the INMXP macro first, followed by INMNODE, then INMEND. Terminate the assembly with an END statement.

Coding the macro instructions

The following paragraphs describe the notation used to define the macro syntax in this publication.

1. The set of symbols listed below are used to define macro instructions, but should never be written in the actual macro instruction:

   hyphen
   - (except where required in the IKJEDIT macro)

   underscore
   _

   braces { }

   brackets
   [ ]

   ellipsis
   . . .

   blank ƀ

   The special uses of these symbols are explained in paragraphs 4-8.
2. Uppercase letters and words, numbers, and the set of symbols listed below should be written in macro instructions exactly as shown in the definition:

   apostrophe
   '

   asterisk
   *

   comma
   ,

   equal sign
   =

parentheses
( )

period .

3. Lowercase letters, words, and symbols appearing in a macro instruction definition represent variables for which specific information should be substituted in the actual macro instruction.

   Example: If **name** appears in a macro instruction definition, a specific value (for example, ALPHA) should be substituted for the variable in the actual macro instruction.

4. Hyphens join lowercase letters, words, and symbols to form a single variable.

   Example: If member-name appears in a macro instruction definition, a specific value (for example, BETA) should be substituted for the variable in the actual macro instruction. Exception: Hyphens are required where noted in the IKJEDIT macro.

5. An underscore indicates a default option. If an underscored alternative is selected, it need not be written in the actual macro instruction.

   Example: The following representation indicates that either A or B or C should be selected; however, if B is selected, it need not be written because it is the default option.

   ```
   A      {A}
   B  or  {B}
   C      {C}
   ```

6. Braces group related items, such as alternatives.

   Example: The following representation indicates that a choice should be made among the items enclosed within the braces. If A is selected, the result is ALPHA=(A,D). If B is selected, the result can be either ALPHA=(,D) or ALPHA=(B,D).

   ```
            {A}
   ALPHA=(  {B},D)
            {C}
   ```

7. Brackets also group related items; however, everything within the brackets is optional and may be omitted.

   Example: The following representation indicates that a choice can be made among the items enclosed within the brackets or that the items within the brackets can be omitted. If B is selected, the result is: ALPHA=(B,D). If no choice is made, the result is: ALPHA=(,D).

   ```
            [A]
   ALPHA=(  [B],D)
            [C]
   ```

8. An ellipsis indicates that the preceding item or group of items can be repeated more than once in succession.

   Example: The following representation indicates that ALPHA can appear alone or can be followed by ,BETA any number of times in succession.

   ```
   ALPHA[,BETA]...
   ```

Note: To designate register 0 and register 1 on a macro invocation, use (0) and (1), respectively. You cannot use a symbolic variable to designate these registers.

IKJBCAST macro

Use the IKJBCAST macro to specify how many 129-byte records TSO/E is to reserve in the notices section of the broadcast data set for messages. The syntax of the IKJBCAST macro follows.

```
[name]   IKJBCAST  {BCLMT={number|100}}
```

name
Symbolic identifier for the macro; begin *name* in column 1.

value: 1-8 alphanumeric characters, beginning with an alphabetic character

BCLMT
Reserve space for messages in the notices section of the broadcast data set

number
The maximum number of 129-byte records

value: an integer in the range 1-1000

Example

To reserve space for 350 129-byte records in the broadcast data set, code

```
IKJBCAST BCLMT=350
```

IKJEDIT macro

Use the IKJEDIT macro to specify default physical characteristics and processing attributes of data sets to be processed using the EDIT command. The syntax of the IKJEDIT macro follows; separate each specified operand by a comma.

```
               {          {tso-defined&cont;
2}{,tso-defined2 }     }
               { DSTYPE1=( {            }&cont;
{              }...) }
               {          {user-defined}{,&cont;
user-defined }     }


                 {         { n }&cont;
{,n }    }{           { name    }{ ,name    }    }
                 { ,BLOCK=({   }&cont;
{   }...)}{,CHECKER=( {         }{          }...)}
                 {         {nn3}{,&cont;
nn3}   }{            {IPDSNEXC4}{,IPDSNEXC4}    }


                       {            { ASIS   }{ ,ASIS   }    }
                       { ,CONVERT5=(&cont;
{ CAPS   }{ ,CAPS   }...)}
                       {            {CAPSONLY&cont;
}{,CAPSONLY}    }


                            {            {  name   }&cont;
{  ,name  }     }
                            { ,DATEXIT=({          }&cont;
{         }...) }
                            {            {ICDQRNME&cont;
6}{ICDQRNME6}      }


                                {,FIXED7=(d-m{,&cont;
d-m}...)}
 [name] IKJEDIT

                                  {          { &cont;
FIXED }{ ,FIXED}      }
                                  {,FORMAT8=(&cont;
{FXDONLY}{,FXONLY}...)  }
                                  {          {  VAR  }&cont;
{ ,VAR  }      }


                                      {,PRMPTR9=(&cont;
name{,name}...)}


                                         [,USEREXT=(name[&cont;
,name]...)]]


                                          {            {&cont;
DATASET}{,DATASET}     }
                                          {,USERSRC&cont;
10=({ INCORE}{ ,INCORE}...) }
                                          {            {&cont;
INLIST }{,INLIST }     }


                                              {,VAR11&cont;
=(d-m{,d-m}...)}
```

1 If you explicitly specify parameters on the DSTYPE operand, ensure that a one-to-one positional correspondence exists with parameters you specify (explicitly or implicitly) on all other operands. Indicate missing corresponding parameters by a comma.

2 See Figure 144 on page 714 for the default data set types (implicit specification).

[3] See Figure 144 on page 714 for the default block size (implicit specification) for particular data set types.

[4] See Figure 144 on page 714 for the data set types for which IPDSNEXC is the default syntax checker (implicit specification).

[5] See Figure 144 on page 714 for the default data conversion attributes (implicit specification) for particular data set types.

[6] ICDQRNME is the default name of the IBM-supplied exit routine invoked for the EDIT RENUM subcommand for VSBASIC data set types. ICDQRNME is the only default.

[7] See Figure 144 on page 714 for the default and maximum LRECL (implicit specification) for particular data set types.

[8] See Figure 144 on page 714 for the default record format (implicit specification) for particular data set types.

[9] See Figure 144 on page 714 for the default compiler names (implicit specification).

[10] See Figure 144 on page 714 for the default type of input (implicit specification) acceptable to the compiler/processing routine specified in PRMPTR.

[11] See Figure 144 on page 714 for the default and maximum LRECL (implicit specification) for particular data set types.

DSTYPE
Explicit default data set type for use with the EDIT command

tso-defined
Predefined data set type

value: as noted in Figure 144 on page 714 (The maximum number of data set types you can specify is 12.)

user-defined
A data set type that you define

value: 1-8 alphanumeric characters, beginning with an alphabetic character. (The maximum number of data set types you can specify is 10.)

BLOCK
Explicit default block size for a data set created using the EDIT command

n Block size, in bytes

value: an integer in the range 8-32,760. (See the explanations under FIXED and VAR for additional information concerning the specification of *n*.)

CHECKER
Check lines in a data set for proper syntax in a syntax-checking routine

name
The name of your syntax-checking routine. Only applicable for a DSTYPE=*user-defined*. (Install the routine prior to system generation.)

value: 1-8 alphanumeric characters, beginning with an alphabetic character

CONVERT
Explicit default style of the input data passed to the EDIT command processor

ASIS Allow input data to remain as entered by the user

CAPS Convert input data to uppercase

CAPSONLY
Always convert input data to uppercase (restrict the user to the specification of the CAPS operand on the EDIT command)

DATEXIT
Process the data set in an exit routine when the user issues the COPY, MOVE, or RENUM subcommand of EDIT

name
The name of your exit routine

value: 1-8 alphanumeric characters, beginning with an alphabetic character

FIXED
Explicit default and maximum length of logical fixed-length records in data sets created using the EDIT command. Code the hyphen between *d* and *m* exactly as shown.

d Default logical record length in bytes

value: an integer in the range 1-255

If you specify a corresponding *n* value in the BLOCK parameter, ensure that it is a multiple of *d*.

If variable-length record format is the default for the corresponding data set type, code a value of 0 for *d*, when you define the maximum length of a fixed-length record.

m Maximum logical record length in bytes

value: an integer in the range 1-255

Specify a value for *m* that is a multiple of, or equal to, the value of *d*.

If you specify a corresponding *n* value in the BLOCK parameter, ensure that it is a multiple of *m*.

FORMAT
Explicit default record format for a data set created using the EDIT command

FIXED
Allow fixed-length record format

FXDONLY
Only allow fixed-length record format

VAR Allow variable-length record format

PRMPTR
Process the source program in a routine when the user issues the RUN subcommand of EDIT

name
The name of your routine

value: 1-8 alphanumeric characters, beginning with an alphabetic character

USEREXT
Interpret the information from the subfield of the *data-set-type* parameter on the

EDIT command in an exit routine and encode it into bytes 0 and 1 of the option word in the syntax checker parameter list

name
The name of your exit routine. Only applicable for a DSTYPE=*user-defined*.

value: 1-8 alphanumeric characters, beginning with an alphabetic character

USERSRC
Input to the processing routine you specified in the PRMPTR parameter

DATASET
The only valid input is a sequential data set

INCORE
The EDIT command processor may pass as input via the input stack

- an in-storage data set - if the size of the data set does not exceed 4096 bytes
- a sequential data set

INLIST
The EDIT command processor may pass as input via the INLIST operand on the invocation of the processing routine

- an in-storage data set - if the size of the data set does not exceed 4096 bytes
- a sequential data set

VAR
Explicit default and maximum length of logical variable-length records in data sets created using the EDIT command. Code the hyphen between *d* and *m* exactly as shown.

d Default logical record length in bytes

value: an integer in the range 5-255

Specify a value for *d* that is less than, or equal to, n-4 of the corresponding *n* value you specified in the BLOCK parameter.

m Maximum logical record length in bytes

value: an integer in the range 5-255

Specify a value for *m* that is greater than, or equal to, the value of *d*.

Specify a value for *m* that is less than, or equal to, n-4 of the corresponding *n* value you specified in the BLOCK parameter.

Example

To specify:

1. a *user-defined* data set type - USER01
   - with an explicit default record format of variable-length records
   - and implicit defaults for all other attributes
2. a *tso-defined* data set type - DATA
   - with an explicit default block size of 2400
   - and implicit defaults for all other attributes
3. and implicit defaults for all other *tso-defined* data set types

Code:

```
EDIT IKJEDIT DSTYPE=(USER01,DATA),BLOCK=(,2400),FORMAT=(FIXED)
```

```
Data                       Default     Maximum
Set      Block  Record      LRECL       LRECL        Data       Checker   Prompter   Prompter
Type     Size   Format    F       V   F       V    Conversion    Name       Name       Input

IBM-Defined Data Set Types

FORTE     400   FXDONLY   80      0   80      0     CAPSONLY   --------   --------   DATASET
FORTG     400   FXDONLY   80      0   80      0     CAPSONLY   --------   --------   DATASET
FORTH     400   FXDONLY   80      0   80      0     CAPSONLY   IPDSNEXC   --------   DATASET
ASM      3120   FXDONLY   80      0   80      0     CAPSONLY   --------   ASM        DATASET
TEXT     3120   VAR        0    255  255    255     ASIS       --------   --------   DATASET
DATA     3120   FIXED     80      0  255    255     CAPS       --------   --------   DATASET
CLIST    3120   VAR        0    255  255    255     CAPSONLY   --------   --------   DATASET
CNTL     3120   FXDONLY   80      0   80      0     CAPSONLY   --------   --------   DATASET
COBOL     400   FXDONLY   80      0   80      0     CAPSONLY   --------   COBOL      DATASET
PLI       400   VAR        0    104  100    104     CAPS       IPDSNEXC   PLIC       DATASET
FORTGI    400   FXDONLY   80      0   80    ---     CAPSONLY   --------   FORT       DATASET
VSBASIC  3120   VAR        0    255   80    255     CAPSONLY   --------   VSBASIC    INLIST
GOFORT   3120   VAR        0    255  255    255     CAPS       IPDSNEXC   GOFORT     INCORE
PLIF      400   FXDONLY    80     0  100      0     CAPSONLY   PLIFSCAN   --------   DATASET

Any User-Defined Data Set Types

----     3120   FIXED     80    255  255    255     CAPS       --------   --------   --------

---- Null Value
```

Figure 144. Attribute defaults for parameters not explicitly specified on the IKJEDIT macro

IKJIFRIF macro

Use the IKJIFRIF macro to add, delete, or change directory entries in the broadcast data set. For more details on using the IKJIFRIF macro instruction, see "Maintaining directory entries in the broadcast data set" on page 206.

IKJIFRIF generates the parameter list and linkage to the broadcast data set interface routine that adds, deletes, or changes directory entries.

IKJIFRIF references the broadcast data set.

Note: With dynamic broadcast support, the SYSLBC DDNAME is no longer required. If the SYSLBC DDNAME is not specified, the currently active broadcast data set will be used by the macro. If the SYSLBC DDNAME is specified, the referenced data set will be used by the macro.

IKJIFRIF has two forms:

1. The list form defines storage (12 fullwords) for the parameter list that is passed to the broadcast data set interface routine.
2. The execute form sets up the parameter list and invokes the broadcast data set interface routine.

Input requirements

In your program that invokes IKJIFRIF:

- The data definitions must include the mapping macros for the CVT and the TSVT (IKJTSVT) in the module.
- The execution portion must have register 13 point to an 18-word save area.

Your program that invokes IKJIFRIF can execute in either 24- or 31-bit addressing mode. The broadcast data set interface routine accepts input above or below 16 MB in virtual storage. However, if executing in 24-bit addressing mode, the parameters you pass must reside below 16 MB in virtual storage.

The parameters must be in the primary address space.

Register conventions

- The execute form of IKJIFRIF alters registers 0, 1, 14, and 15.

Output

- IKJIFRIF returns a return code in register 15 or in the variable *retcode*. See the syntax of the execute form of the macro for information on *retcode*.

IKJIFRIF (list form)

Use the list form of the macro to define storage for the parameter list that is passed to the broadcast data set interface routine. The syntax of the list form of the IKJIFRIF macro follows.

```
 label     IKJIFRIF    MF=L
```

label
The symbolic identifier of the macro.

value: 1-8 alphanumeric characters, beginning with an alphabetic character

MF Macro form

L List form

IKJIFRIF (execute form)

Use the execute form of the macro to set up the parameter list and invoke the broadcast data set interface routine. The syntax of the execute form of the IKJIFRIF macro follows.

- All parameters are keyword parameters.
- An address may be either a 24-bit or a 31-bit address.
- When using register notation, enclose the specified register in parentheses.

```
                         ECT=ectadr,PSCB=pscbadr,UPT=uptadr,

 [label]     IKJIFRIF    {ADD=addadr|DEL=deladdr|ALT=altadr},
g
                         [RETCODE=retcode,] MF=(E,parmlist-name)
```

label
The symbolic identifier of the macro

value: 1-8 alphanumeric characters, beginning with an alphabetic character

ECT
Environment control table

ectadr
Address of the ECT

value:

1. RX-type address
2. a register, 2–12, containing the address.

PSCB

Protected step control block

pscbadr

Address of the PSCB

value:

1. RX-type address
2. a register, 2–12, containing the address.

UPT

User profile table

uptadr

Address of the UPT

value:

1. RX-type address
2. a register, 2–12, containing the address.

ADD

Add either a directory entry for one user ID, or a list of directory entries for a number of user IDs, to the broadcast data set.

addadr

A storage area containing the only user ID or the first user ID in a chain of user IDs

value:

1. RX-type address
2. a register, 2–12, containing the address.

DEL

Delete either a) a directory entry for the user ID, or b) a chain of directory entries for a number of user IDs, from the broadcast data set.

deladr

A storage area containing the only user ID or the first user ID in a chain of user IDs

value:

1. RX-type address
2. a register, 2–12, containing the address.

ALT

Change either a) a directory entry for one user ID to a new user ID, or b) a chain of directory entries for a number of user IDs to corresponding new user IDs, in the broadcast data set.

altadr

A storage area containing the only user ID or the first user ID in a chain of user IDs

value:

1. RX-type address
2. a register, 2–12, containing the address.

RETCODE

Return code

retcode
A fullword that will contain the return code from IKJIFR00

value: RX-type address

MF Macro form

E Execute form

parmlist-name
Name or address of the storage area for the parameter list

value:
1. the *label* specified on the list form of the macro
2. a register, 2–12, containing the address

IKJIFRIF parameter list

The execute form of IKJIFRIF sets up a parameter list where register 1 points to the parameter list shown in Figure 145.

Upon return from the execute form of IKJIFRIF, the parameter list may contain error indicators. Error indicators include:
- Register 15 and optionally the RETCODE parameter you specified
- Abend code field in the parameter list
- Reason code field in the parameter list
- Service return code field in the parameter list
- Service reason code field in the parameter list

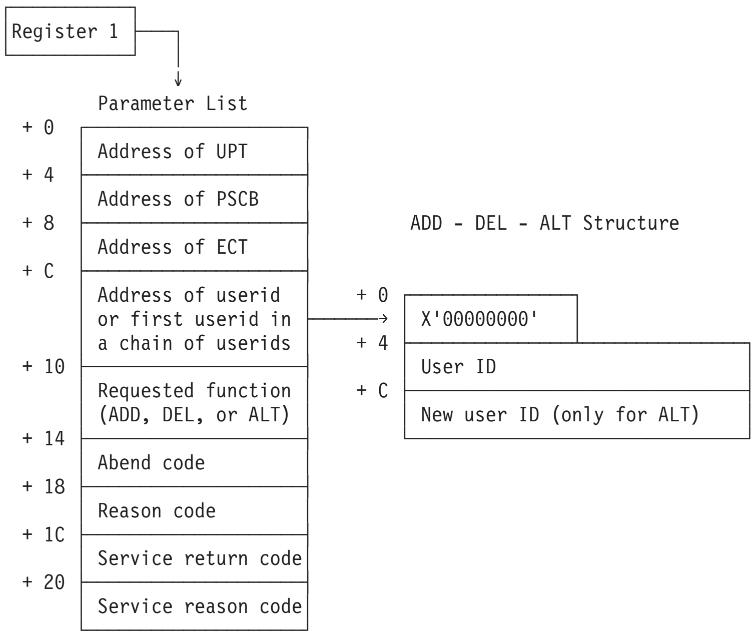


Figure 145. Parameter list structure for IKJIFRIF (single user ID)

Figure 145 shows, right to the parameter list, the organization of the storage area for a single directory entry. Multiple directory entries can be added, deleted, or changed in the broadcast data set with a single invocation of the IKJIFRIF macro instruction. Figure 146 on page 718 shows the organization of the storage area for three directory entries.

Note that the address at offset X'C' in the parameter list (the address of a user ID or the first user ID in a chain of user IDs) needs to start on a fullword boundary in order for the IKJIFRIF macro to evaluate this structure. See the following examples.

Offset (Hex)	Number of bytes	Contents or meaning
0	4	For a single directory entry this must be X'00000000'. For a chain of user IDs this is a pointer to the next entry. Note that the last entry must be set to X'00000000'.
4	8	User ID *n* to be added, deleted, or changed.
C	8	New user ID *n*, if the ALT function of the IKJIFRIF macro is invoked.

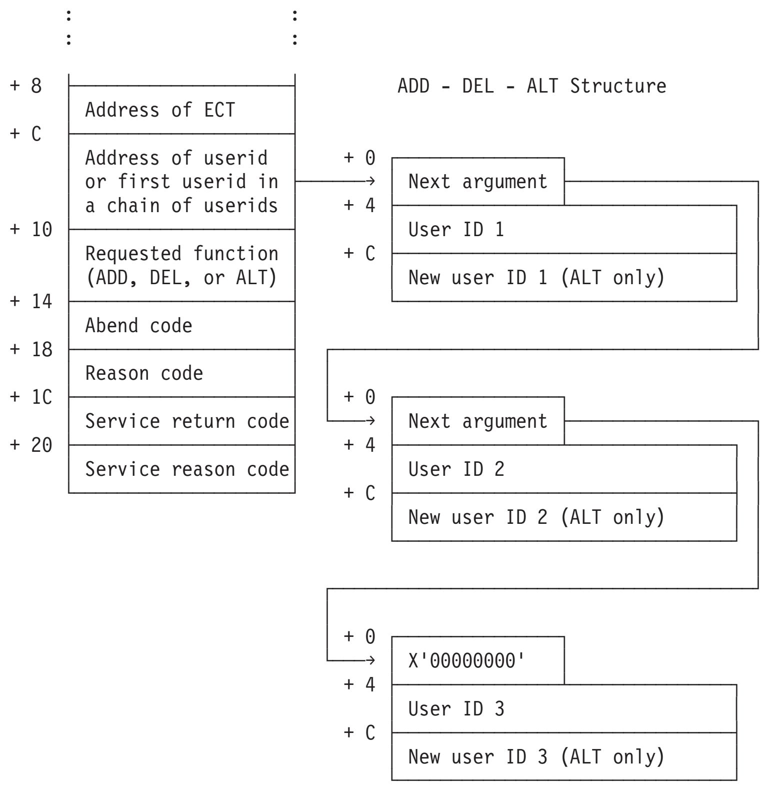


Figure 146. Parameter list structure for IKJIFRIF (multiple directory entries)

Return codes from IKJIFRIF

Upon return from the execute form of IKJIFRIF register 15, and optionally the RETCODE parameter you specified, contains a return code. The possible return codes and their explanations are as follows:

Return code	Explanation
0	Processing completed successfully
4	Processing unsuccessful - recovery environment could not be established
8	Processing unsuccessful - caller is in 24-bit addressing mode, but the argument address(es) were not valid 24-bit address(es)
12	Processing unsuccessful - incorrect interface function specified

Return code	Explanation
16	Processing unsuccessful - user parameters were incorrect
20	Processing unsuccessful - An internal error occurred. The service return code field of the macro-generated parameter list contains further diagnostic information.
24	Processing unsuccessful - an abend occurred. The abend code field of the macro-generated parameter list contains further diagnostic information.
28	Processing incomplete. Deallocation failed. Check the broadcast data set to see if the function was successfully performed to the directory entry or entries. Also check the service return code field and the service reason code field for return codes from the dynamic allocation routine.
32	Processing unsuccessful - TSO/E Release 4 or higher is not installed.
36	Processing unsuccessful - too many arguments in the chain of user IDs for the requested function were passed to the broadcast data set interface routine and caused the amount of storage requested by the GETMAIN macro to become a negative amount.

Example 1

Operation: Add a directory entry to SYS1.BRODCAST. ADDADR is described as follows:

1. Pointer to next entry, or zero for last entry
2. New directory entry

```
*       Initialization and set up code for the module               *
*       Include a pointer to the CVT                                *
               .
               .
               .
CVTMAP    CVT DSECT=YES    *            map the CVT                 *
TSVTMAP   IKJTSVT          *            map the TSVT                *
PARMLIST  IKJIFRIF MF=L    * define storage for the parameter list  *

*     Issue IKJIFRIF to set up and initialize the parameter list     *

ADDID     IKJIFRIF UPT=UPTADR,PSCB=PSCBADR,ECT=ECTADR,                X
                   ADD=ADDADR,MF=(E,PARMLIST)

*           Include code to handle error conditions                 *
*              Define storage for the userid(s)                     *

ALIGN     DS 0F     * Alignment to word boundary                    *
ADDADR    DS 0CL12  * Name field as specified below                 *
NEXT      DS F      * Zero, if only one directory entry to be added  *
USERID    DS CL8    * New directory entry                           *
```

Example 2

Operation: Change a directory entry in SYS1.BRODCAST. ADDADR is described as follows:

1. Pointer to next entry, or zero for last entry
2. Existing directory entry
3. New directory entry

```
*       Initialization and set up code for the module              *
*       Include a pointer to the CVT                               *
              .
              .
              .
CVTMAP    CVT DSECT=YES    *             map the CVT               *
TSVTMAP   IKJTSVT          *             map the TSVT              *
PARMLIST  IKJIFRIF MF=L    * define storage for the parameter list *

*    Issue IKJIFRIF to set up and initialize the parameter list    *

ALTID     IKJIFRIF UPT=UPTADR,PSCB=PSCBADR,ECT=ECTADR,               X
                   ALT=ALTADR,MF=(E,PARMLIST)

*          Include code to handle error conditions                 *
*             Define storage for the user IDs                      *

ALIGN     DS 0F     * Alignment to word boundary                   *
ALTADR    DS 0CL20  * Name field as specified below                *
NEXT      DS F      * Zero, if only one directory entry to be changed*
OLDID     DS CL8    * Existing directory entry                     *
NEWID     DS CL8    * New directory entry                          *
```

Example 3

Operation: Delete a directory entry from SYS1.BRODCAST. ADDADR is described as follows:

1. Pointer to next entry, or zero for last entry
2. Directory entry to be deleted

```
*       Initialization and set up code for the module              *
*       Include a pointer to the CVT                               *
              .
              .
              .
CVTMAP    CVT DSECT=YES    *             map the CVT               *
TSVTMAP   IKJTSVT          *             map the TSVT              *
PARMLIST  IKJIFRIF MF=L    * define storage for the parameter list *

*    Issue IKJIFRIF to set up and initialize the parameter list    *

DELID     IKJIFRIF UPT=UPTADR,PSCB=PSCBADR,ECT=ECTADR,               X
                   DEL=DELADR,MF=(E,PARMLIST)

*          Include code to handle error conditions                 *
*             Define storage for the user ID(s)                    *

ALIGN     DS 0F     * Alignment to word boundary                   *
DELADR    DS 0CL12  * Name field as specified below                *
NEXT      DS F      * Zero, if only one directory entry to be deleted*
USERID    DS CL8    * Directory entry to be deleted                *
```

IKJTSO macro

Use the IKJTSO macro to specify:

- the total number of times a terminal user may be prompted for operands at logon before TSO/E automatically cancels the logon attempt
- how much time the terminal user may wait without a terminal response during a logon attempt

The syntax of the IKJTSO macro follows.

```
[name]  IKJTSO  {LOGLINE={lines|10}}{,LOGTIME={sec|300}}
```

name
Symbolic identifier for the macro

value: 1-8 alphanumeric characters, beginning with an alphabetic character

Begin *name* in column 1.

LOGLINE
Set logon input-line limit

lines
The maximum number of input lines; that is, the total number of times a terminal user can be prompted for operands

value: an integer in the range 1-16777215

LOGTIME
Set logon wait-time response limit

sec
The maximum number of seconds

value: an integer in the range 1-16777215

Example

To set a 10 minute wait-time response limit and a 5 input-line logon limit, code:

```
LIMITS IKJTSO LOGTIME=600,LOGLINE=5
```

INMEND macro

The INMEND macro terminates the list of INMNODE macros. The syntax of the INMEND macro follows.

```
ƀ    INMEND
```

INMNODE macro

The INMNODE macro builds a table that establishes correspondencebetween system identifiers and network node names.

- For each system on which the TRANSMIT and RECEIVE commands will be executed, code the macro once.
- For loosely-coupled JES3 complexes and JES2 multi-access spool complexes, code the macro two or more times specifying the same *node-name* but different *smf-ids*.
- For a multiprocessor, code the macro once.
- You only have to code the INMNODE macro for the host node (the node, into which, you will link-edit this copy of INMXPARM). You do not have to code an INMNODE macro for every node in the network.

  If you code an INMNODE macro for nodes other than the host, and one of the other nodes has the same *smf-id* as the host node, assemble the INMNODE macro for the host ahead of the INMNODE macro for the node with the duplicate *smf-id*.

The syntax of the INMNODE macro follows.

```
ƀ   INMNODE   node-name,smf-id
```

node-name
The name of a network node

value: must be one of the following:
- The same as a node name you specified on the NJERMT JES3 initialization statement or on the NODE(nnnn) JES2 initialization statement, or
- If you do not define a node name to JES, the default node name NODE0001 (or N001).

smf-id
The system identifier for a particular processor or a multiprocessor

value: must be the same as a system identifier you specified in the SID parameter of the SMFPRMxx PARMLIB member

Note: If you omit the *smf-id* for the host node, TSO/E uses a value of eight question marks (????????) for the *node-name* associated with the transmitted data.

INMXP macro

The INMXP macro defines installation controls and defaults for certain parameters on the TRANSMIT and RECEIVE commands, such as the unit for allocating temporary space, transmission size limits, and data set names. The syntax of the INMXP macro follows. Separate each specified parameter by a comma.

```
              {        {ALWAYS}}{          &cont;
{  n1,n2   }   }{        {  n3  }}
              {CIPHER= {YES   }}{,OUTWARN=(&cont;
{         } )}{,OUTLIM={      }}
              {        {NO    }}{         &cont;
 {10000,5000}  }{       {30000 }}


                 {        {logselector}}&cont;
{        {logname-suffix}}
                 {,LOGSEL= {           }}&cont;
{,LOGNAME={             }}
                 {        {  LOG1     }}&cont;
{        {   MISC1      }}


  b    INMXP        {        {   name    }}&cont;
{        {sysout-class}}
                    {,USRCTL= {           }}&cont;
{,SYSOUT= {           }}
                    {        {NAMES.TEXT1}}&cont;
{        {     *1     }}


                       {      {unit-name}&cont;
}               {        {     }}
                       {,VIO= {         }}&cont;
[,SYSCTL=dsname] {,SPOOLCL={class}}
                       {      {   __2   }&cont;
}               {        {  B  }}
```

1 See the explanation under each parameter as to when TSO/E uses the indicated default.

2 See the explanation under the VIO parameter as to the default TSO/E uses.

CIPHER
Data encryption control

ALWAYS
The user must encrypt every transmission. (The specification of ALWAYS is equivalent to forcing every user to specify the ENCIPHER operand on the TRANSMIT command or for you to provide encryption using the TRANSMIT encryption exit, INMXZ03 or INMXZ03R.)

YES Data encryption is a user option

NO Do not allow the user to encrypt any transmission. (The specification of NO overrides the specification of the ENCIPHER operand on the TRANSMIT command and does not allow you to provide encryption with the TRANSMIT encryption exit, INMXZ03 or INMXZ03R.)

OUTWARN
Warn a user who is transmitting a large file

n_1 Give the user the first warning after the transmission of the designated number of records

value: a decimal integer

n_2 Give the user the second and subsequent warnings each time after the transmission of the designated number of records

value: a decimal integer

OUTLIM
Terminate a transmission

n_3 If a user attempts to transmit n_3 records, the system terminates the transmission

If n_3 is less than or equal to 16,777,215, the system passes the value to JES as the OUTLIM value.

If n_3 exceeds 16,777,215, the system does not pass the value to JES; however, n_3 still serves as the limit for the TRANSMIT command.

value: a decimal integer

For additional information on output limits, refer to

- *z/OS MVS JCL Reference* - the OUTLIM DD statement
- *z/OS MVS Programming: Authorized Assembler Services Guide* - SVC 99 SYSOUT output limit specification DALOUTLM (key=X'001B')

LOGSEL

The default middle qualifier(s) for the log data set name. (The name in the :LOGSEL tag in the control section of the NAMES data set takes precedence.)

logselector

The name(s) of the middle qualifier(s)

value: 1-8 alphanumeric characters with the first character being alphabetic or one of the special characters (#, @, $) for each name. Separate the names by periods. (See Note following LOGNAME for maximum length restrictions.)

LOGNAME

The default suffix qualifier(s) for the log data set name. (The name in the LOGNAME operand of the TRANSMIT command, the :LOGNAME tag in the control section of the NAMES data set, or the :LOGNAME tag in a nickname definition takes precedence.)

logname-suffix

The name(s) of the suffix qualifier(s)

value: 1-8 alphanumeric characters with the first character being alphabetic or one of the special characters ($, #, or @) for each name. Separate the names by periods. (See the following note for maximum length restrictions.)

Note: TSO/E prefixes the name of the log data set with the user-specified *dsname-prefix* from the PROFILE command, such that the name is equivalent to *prefix.logselector.logname-suffix*. The maximum length of the name is 44 characters including the periods and the *prefix*.

In the absence of any explicit specification, the default log data set name is *userid*.LOG.MISC.

USRCTL

A name for the NAMES data set

name

The name of the data set

value: 1-8 alphanumeric characters with the first character being alphabetic or one of the special characters ($, #, @) for each name. Separate the names by periods.

Note: TSO/E prefixes the name of the NAMES data set with the user-specified *dsname-prefix* from the PROFILE command. The maximum length of the name is 44 characters including the periods and the *prefix*.

In the absence of any explicit specification, the default NAMES data set name is *userid*.NAMES.TEXT.

SYSOUT
Default SYSOUT class for messages written by utility programs (for example, IEBCOPY)

sysout-class
The designated SYSOUT class

value: A-Z, 0-9, or *

Note: In the absence of any explicit specification, the system writes the messages to the terminal (SYSOUT=*).

VIO
A device type on which the system can allocate temporary space for use by the TRANSMIT and RECEIVE commands

- If you do not specify VIO, the value defaults to the UNIT specification for a user in the UADS.
- If no UNIT specification exists, the value defaults to an installation-defined default or, if none exists, to the system default (SYSALLDA).

In each of the three cases, IEBCOPY may fail.

unit-name
The name of the device type

value:
1. an esoteric name (for example, SYSDA)
2. a specific DASD device (for example, 3350)

Note: For *unit-name* device type that you designated as VIO at system generation, use a VIO to ensure the integrity of sensitive data.

SYSCTL
The name for an alternate NAMES data set. (You can use this parameter in conjunction with a routine you write to provide a global standard set of 'nicknames' within the installation. For example, the routine could manipulate the entries in an internal directory and store the resulting output - 'nicknames' - in the SYSCTL data set. End users could then use the standard set without having to be concerned about defining theirs on an individual basis.)

dsname
The name of the data set. (The name must conform to MVS data set naming conventions. TSO/E does not prefix the data set name.)

value: 1-8 alphanumeric characters with the first character being alphabetic or national for each name. Separate the names by periods. The maximum length of the name is 44 characters including the periods.

SPOOLCL
Specify the output class default. Use the SPOOLCL operand *class* to specify your installation's output class default. If you do not specify a different SPOOLCL operand, then a default of 'B' is used.

class
: The name of the SPOOLCL class

value: A-Z, 0-9, or *

Note: * is a literal character. It is valid only if * is specified as a valid class.

Example

```
INMXP     CIPHER=NO,OUTWARN(5000,2000),OUTLIM=15000,VIO=3350
INMNODE   BOST,168A      CPU #1 in a JES2 complex
INMNODE   BOST,168B      CPU #2 in a JES2 complex
INMNODE   POK,303C
INMNODE   NYC,303A
INMNODE   SJRLVS1,3081
INMEND
END
```

Part 11. Appendixes

Appendix A. Executing the terminal monitor program

The terminal monitor program (TMP) provides an interface between the user, command processors, and the TSO/E control program. It obtains commands, gives control to command processors, and monitors their execution. The TMP is attached as APF-authorized and executes in either supervisor state or problem program mode. The TMP entry points (IKJEFT01, IKJEFT1A, and IKJEFT1B) can only be invoked in one of the following ways:

- Using the EXEC statement in the logon procedure for foreground execution
- Using the EXEC statement in the input stream for background execution
- Using the EXEC statement in the JCL portion of a transaction program (TP) profile for a standard TP scheduled by the APPC/MVS transaction scheduler.
- Using the EXEC statement in the logon procedure for foreground execution to execute Session Manager, instructing Session Manager to invoke the TMP that TSO/E provides
- Using a program that passes control to the TMP

When you log on to TSO/E, the TMP is invoked from your logon procedure. You can also execute the TMP in the background by submitting JCL. Executing the TMP in the background allows you to execute TSO/E commands independent of the terminal, which is convenient if a job takes a long period of time to execute. It is also a useful way to perform certain customization tasks, such as maintaining the UADS and broadcast data sets.

This appendix describes the JCL statements that are required to execute the TMP in the background. As an alternative to writing JCL statements, you can use the SUBMIT command to generate the JCL needed to execute commands in the background. See *z/OS TSO/E User's Guide* for more information.

Writing JCL for command execution

Figure 147 on page 730 illustrates the JCL statements needed to execute the TMP in the background.

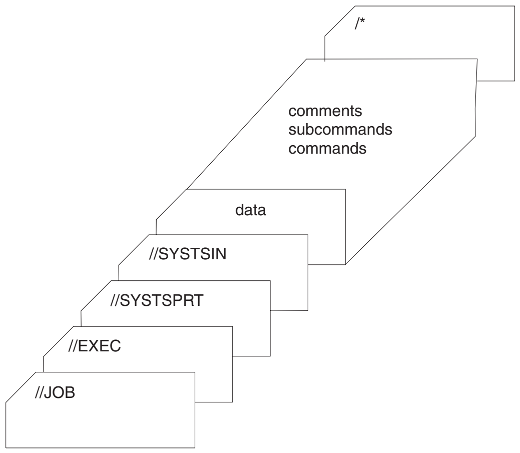


Figure 147. JCL needed to process commands in the background

The JCL required to execute the TMP as a batch job includes the following:

1. A JOB statement, including a jobname and operands that specify the processing options.
2. An EXEC statement that specifies IKJEFT01 (the TMP) as the program to be executed. The format is:

   ```
   //stepname EXEC PGM=IKJEFT01,DYNAMNBR=nn,PARM='command'
   or
   //stepname EXEC PGM=IKJEFT01,DYNAMBR=nn,PARMDD=instrdd
   ```

   Note: The TMP must not run as a V=R. Running the TMP as V=R can cause unpredictable results.

   If you are executing commands that dynamically allocate data sets, specify the DYNAMNBR parameter.

   You may use the PARM parameter to specify the first command to be executed. The command that is executed is taken from the first line of input indicated by the SYSTSIN DD statement.

   If a command or program ABENDs, the TMP will place a return code of 12 (X'C') in register 15 and terminate the job step normally. No further commands or subcommands will be executed after the command that ABENDed.

   Note: If you need to pass a command string longer than 100 characters to IKJEFT01, you should use the PARMDD= keyword of the EXEC statement of the JCL, as shown above. The PARM='command' form of passing a parameter is limited to a maximum of 100 characters.

   If a command or program returns a non-zero return code to the TMP, the TMP does save this return code and continues processing the next command or subcommand. The return code from the next command replaces the return code. The return code returned by the batch IKJEFT01 job (which is the job condition code) is basically the return code of the last command that has been executed, so the IKJEFT01 return code (such as a job condition code) is the rolling return code from each command invoked with the condition code, which is the last return code of the last command.

There are two alternative entry points available for background execution that provide additional return code and ABEND support. They can be used by substituting PGM=IKJEFT01 on the EXEC statement with one of the following:

- PGM=IKJEFT1A
  - If a command or program being processed by IKJEFT1A ends with a system abend, IKJEFT1A causes the job step to terminate with a X'04C' system completion code. IKJEFT1A also returns the completion code from the command or program in register 15.
  - If a command or program being processed by IKJEFT1A ends with a user abend, IKJEFT1A saves this completion code in register 15 and then terminates.
  - If a command, program or REXX exec being processed by IKJEFT1A returns a non-zero return code to IKJEFT1A, IKJEFT1A saves this return code in register 15 and then terminates. Non-zero return codes to IKJEFT1A from CLISTs will not affect the contents of register 15 and the TMP will continue processing. However, commands invoked by CLISTs can have an affect.
  - For a non-zero return code or an abend from a command or program that was not given control directly by IKJEFT1A or a CLIST running under IKJEFT1A, no return code is saved in register 15 and IKJEFT1A does not terminate.
  - If the command terminates with a non-zero return code, IKJEFT1A saves the return code in register 15, flushes the CLIST, and terminates. This is different than what happens when a command is invoked directly by a REXX exec.
- PGM=IKJEFT1B
  - If a command or program being processed by IKJEFT1B ends with a system or user abend, IKJEFT1B causes the job step to terminate with a X'04C' system completion code. IKJEFT1B also returns the completion code from the command or program in register 15.
  - If a command, program or REXX exec being processed by IKJEFT1B returns a non-zero return code to IKJEFT1B, IKJEFT1B saves this return code in register 15 and then terminates. Non-zero return codes to IKJEFT1B from CLISTs do not affect the contents of register 15 and the TMP continues processing.
  - For a non-zero return code or abend completion code from a program or command that was not given control by IKJEFT1B, no return code is saved in register 15 and IKJEFT1B does not terminate.
- For PGM=IKJEFT1A and PGM=IKJEFT1B, if the job step is terminated with a X'04C' system completion code, the conditional disposition for each data set that is still allocated at step termination will be honored:
  - A DD statement can be freed prior to step termination either by using the DYNALLOC macro (SVC 99) in a program or by executing the TSO/E FREE command. If a DD statement that has a conditional disposition specified has been freed prior to step termination, this DD statement will not have the conditional disposition honored because no data set allocation exists for standard deallocation processing in the initiator to process.
  - The conditional disposition for a data set can only be specified by coding it in the JCL statement for the DD or by using the DYNALLOC macro (SVC 99) and specifying the DALCDISP text unit in the list of text units specified in the SVC 99 request block. The conditional disposition for a data set cannot be specified by using the TSO/E ALLOCATE command.

If the data set is allocated using the DYNALLOC macro (SVC 99) in a command or program, all of the following requirements must occur in order for the conditional disposition to be honored:

- The TMP must have been started using alternate entry point IKJEFT1A or IKJEFT1B.
- The command or program must be invoked directly by the TMP.
- The command or program must issue the DYNALLOC macro (SVC 99) specifying the DALCDISP text unit in the list of text units specified in the SVC 99 request block.
- If the IKJEFT1A alternate entry point was used, the command or program must abend with a system abend.
- If the IKJEFT1B alternate entry point was used, the command or program must abend with either a system or a user abend .

If the command or program that issued the DYNALLOC macro (SVC 99) does not abend, but a subsequent command or program does abend, the conditional disposition specified in the DYNALLOC macro will not be honored. This is because the TMP treats each command or program as a logical "step" and performs all clean up processing (including, but not limited to, data set disposition) at the termination of each command or program invoked directly by the TMP.

Example 1 - Conditional Disposition honored:

```
//jobcard
//TSO EXEC PGM=IKJEFT1A
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 command_processor_a   parameters
 command_processor_b   parameters
```

- `command_processor_a` issues a DYNALLOC macro (SVC 99) specifying the DALCDISP text unit and then abends with a system abend at some point after the DYNALLOC is successful

Because the command or program that issued the DYNALLOC macro (SVC 99) was invoked directly by the TMP and abended before control was returned to the TMP, the conditional disposition specified in the DALCDISP text unit pointed to by the SVC 99 request block will be honored.

Example 2 - Conditional Disposition not honored:

```
//jobcard
//TSO EXEC PGM=IKJEFT1A
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 command_processor_a   parameters
 command_processor_b   parameters
```

- `command_processor_a` issues a DYNALLOC macro (SVC 99) specifying the DALCDISP text unit and then terminates normally
- `command_processor_b` then abends with a system abend

Because `command_processor_a` (which issued the DYNALLOC macro (SVC 99)) did not abend, the TMP frees all data sets allocated by `command_processor_a`. Since this is a normal completion, the conditional disposition of the data sets is not honored.

Example 3 - Conditional Disposition honored:

```
//jobcard
//TSO EXEC PGM=IKJEFT1A
//CONDDISP DD DSN=data.set.name,DISP=(NEW,CATLG,DELETE),
//     UNIT=SYSALLDA,SPACE=(TRK,(1,1))
```

```
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 command_processor_a   parameters
 command_processor_b   parameters
```

where

- `command_processor_a` runs normally
- `command_processor_b` then abends with a system abend

Because the data set was allocated by the JCL DD statement, and was not freed, it is still allocated at the termination of the job step. Since the job step will terminate with a system abend code X'04C' (because `command_processor_b` abended with a system abend), the conditional disposition for all data sets in the step will be honored.

Example 4 - Conditional Disposition not honored:

```
//jobcard
//TSO EXEC PGM=IKJEFT1A
//CONDDISP DD DSN=data.set.name,DISP=(NEW,CATLG,DELETE),
//     UNIT=SYSALLDA,SPACE=(TRK,(1,1))
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 command_processor_a   parameters
 FREE FILE(CONDDISP)
 command_processor_b   parameters
```

- `command_processor_a` runs normally
- `command_processor_b` then abends with a system abend

Because the data set allocated by the JCL DD statement was freed, the normal disposition (CATLG) for the data set will be honored. The data set will no longer be allocated when the job step ends (with system abend code X'04C' because `command_processor_b` abended with a system abend). Even though the job step will end with a system abend code, the data set has already been freed and therefore the initiator has no data set allocated for which to perform disposition processing.

For more information about the DYNALLOC macro and the SVC 99 Request Block, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

For more information about the TSO/E ALLOCATE and FREE commands, see *z/OS TSO/E Command Reference*.

3. A SYSTSPRT DD statement that controls output from your job. This DD statement can refer to a system printer or to a sequential or partitioned data set. If the data set is partitioned, you must specify the member name on the DD statement as DSN=pdsname(*membername*).

   Messages issued by programs using the TSO/E I/O service routines are written to the data set indicated by the SYSTSPRT DD statement.

4. A SYSTSIN DD statement that controls input to your job. Use this statement to indicate which commands and subcommands are to be executed.

   You can specify the input data directly following the SYSTSIN DD statement, or you can refer to a sequential or partitioned data set. If the data set is partitioned, you must specify the member name on the DD statement as DSN=pdsname(*membername*). Each command or subcommand, such as subcommands of TSO/E EDIT, must begin on a separate statement.

   It is suggested that the SYSTSIN DD be defined as a fixed block format data set, with an LRECL of 80.

   If SYSTSIN is a fixed length data set (FB), the last 8 bytes of the record will be treated as a sequence number and ignored.

If SYSTSIN is a fixed length data set with ASA control characters (FBA), the first byte of the record will be treated as a carriage control character and ignored.

If SYSTSIN is a variable length data set (VB), the first 8 bytes of the record will be treated as a sequence number and ignored.

If SYSTSIN is a variable length data set with ASA control characters (VBA), the first 9 bytes of the record will be treated as a sequence number, followed by a carriage control character and ignored.

Programs that use the TSO/E I/O service routines to obtain input receive their input from the data set indicated by the SYSTSIN DD statement. The records can be up to 72 bytes in length.

Use the SYSPROC DD statement to execute CLISTs implicitly. If a CLIST is a member of a partitioned data set, you can execute the CLIST implicitly by simply specifying the member name. For more information on executing CLISTs, see *z/OS TSO/E CLISTs*.

Considerations for executing commands in the background

- When the TMP executes in the background, some TSO/E commands are processed differently than when they execute in the foreground. For example, a job that invokes the TMP in the background is authorized to execute the ACCOUNT command, if the "USER=" field is not specified. Therefore, for security reasons, you should either:
  - RACF-protect or password-protect the UADS
  - Write a JCL exit to limit access to the background TMP.

  For more information about the differences in command processing in the background, see *z/OS TSO/E User's Guide*.
- An unqualified data set name will not be prefixed with a user ID, because "no prefixing" of data set names is the default in the background. If you want a user ID prefixed to the data set names, and are not defined to RACF, include the following command at the beginning of the SYSTSIN stream:

  ```
  profile prefix(userid)
  ```

  If you are defined to RACF, you do not have to include the `profile prefix(userid)` command in the SYSTSIN stream. Instead, specify `USER=userid` on the JOB card, and the user ID you specify is used as the prefix.

  Read also the following item about the requirement that a user must have logged on and off at least once.
- If your batch job uses commands or subcommands in the SYSTSIN stream that use user IDs not defined as TSO/E users (for example, SEND or RECEIVE), consider the following requirement:

  The TMP requires user logon information found in either the user attribute data set SYS1.UADS, or the TSO/E segment of the RACF data base. This information is built through the LOGON and LOGOFF command. Therefore, it is required that each referenced user is defined to either UADS or to RACF, and has performed a LOGON and LOGOFF command at least once.

Considerations for programs that invoke the TMP

The TMP must be given control by attaching it as a job step task in a state similar to that provided when JCL is used to cause the system to invoke it. Note that the TMP should be attached as the one and only JOBSTEP TCB.
- APF-authorized
- Primary addressing mode

- AMODE(31)
- Swappable
- Key 8 PSW and application subpools
- Registers:

Register	Contents
1	Address of primary mode parameter list, specifying the first command to be executed.
13	Address of a caller-provided key 8 save area.
14	Return address.
15	Address of entry point into the TMP.

See “Linkage Conventions” in *z/OS MVS Programming: Assembler Services Guide* for details.

Dynamic allocation can be used to satisfy the requirements for DD statements prior to passing control to the TMP.

Only one invocation of the TMP is supported during a step of a job, and only one instance of the TMP can be concurrently active within an address space.

Use of system services by the caller of the TMP during the TMP session can impact the timing and delivery of session services, and the TMP and the commands that it invokes can impact the timing and operation of the caller.

The TMP can alter execution status and the resources owned by the job during its operation. For example,
- APF-authority can be relinquished.
- Data set allocation can be changed.
- Commands given control during the session can change any key 8 storage.

  ⋮

Some of these changes may remain in effect when control is returned to the caller of the TMP. When control is returned to the caller, all registers except for register 15 will contain the same values passed to the TMP on entry. Register 15 will contain a return code. See Figure 147 on page 730 for more information about that return code.

Appendix B. Example logon pre-prompt exit IKJEFLD

Figure 148 on page 738 shows a sample unauthorized logon pre-prompt exit.

```
IKJEFLD TITLE 'TSO/E LOGON PRE-PROMPT INSTALLATION EXIT EXAMPLE'
IKJEFLD CSECT
 TITLE 'TSO/E LOGON PRE-PROMPT INSTALLATION EXIT EXAMPLE PROLOGUE'
************************************************************************
*                                                                      *
* MODULE NAME - IKJEFLD                                                *
*                                                                      *
* CSECT  NAME - IKJEFLD                                                *
*                                                                      *
* DESCRIPTIVE NAME - TSO/E INSTALLATION EXIT EXAMPLE                   *
*                                                                      *
* FUNCTION - EXAMPLE FOR TSO/E LOGON PRE-PROMPT EXIT                   *
*                                                                      *
* OPERATION - IKJEFLD PERFORMS THE FOLLOWING FUNCTION:                 *
*                                                                      *
*    1 - SAVES CALLER'S REGISTERS IN CALLER'S SAVE AREA                *
*    2 - ESTABLISHES ADDRESSABILITY TO MODULE IKJEFLD                  *
*    3 - SAVES PARAMETER POINTER (R1)                                  *
*    4 - DOES A GETMAIN FOR PROGRAM'S SAVE AREA                        *
*    5 - CHAINS THE CALLER'S SAVE AREA AND THIS SAVE AREA              *
*    6 - DOES A GETMAIN FOR PROGRAM'S WORK AREA                        *
*    7 - ESTABLISHES ADDRESSABILITY TO IKJEFLD'S WORK AREA             *
*    8 - PROCESSES THE EXIT JCL STATEMENTS                             *
*    9 - DOES A FREEMAIN OF DYNAMIC WORK AREA                          *
*   10 - DOES A FREEMAIN OF DYNAMIC SAVE AREA                          *
*   11 - LOADS REGISTER 14 WITH RETURN ADDRESS                         *
*   12 - SETS THE RETURN CODE IN REGISTER 15 TO ZERO                   *
*   13 - LOADS REGISTERS R0 - R12 WITH CALLER'S ENTRY CONTENTS         *
*        FROM CALLER'S REGISTER SAVE AREA                              *
*   14 - RETURNS TO CALLER                                             *
*                                                                      *
* RECOVERY OPERATION -                                                 *
*                                                                      *
*          YOU MUST PROVIDE EXIT RECOVERY                              *
*                                                                      *
* DEPENDENCIES - NONE                                                  *
*                                                                      *
*                                                                      *
 TITLE 'IKJEFLD MODULE ATTRIBUTES'
* MODULE ATTRIBUTES - REENTRANT,                                       *
*                     REFRESHABLE,                                     *
*                     REUSABLE                                         *
*                                                                      *
************************************************************************
 TITLE 'REGISTER EQUATES'
************************************************************************
* REGISTER EQUATES FOLLOW                                              *
************************************************************************
*     REGISTER EQUATES AND USAGE                                       *
*                                                                      *
         SPACE 3
R0       EQU   0                     UNPREDICTABLE
R1       EQU   1                     ON ENTRY R1 POINTS TO THE
*                                    PARAMETER LIST. AT OTHER TIMES
*                                    THE VALUE OF R1 IS UNPREDICTABLE
R2       EQU   2                     UNPREDICTABLE
R3       EQU   3                     UNPREDICTABLE
R4       EQU   4                     UNPREDICTABLE
R5       EQU   5                     UNPREDICTABLE
R6       EQU   6                     UNPREDICTABLE
R7       EQU   7                     UNPREDICTABLE
R8       EQU   8                     UNPREDICTABLE
R9       EQU   9                     UNPREDICTABLE
R10      EQU   10                    OUTPUT BUFFER
R11      EQU   11                    DATA AREA BASE REGISTER
R12      EQU   12                    CODE AREA BASE REGISTER
R13      EQU   13                    SAVE AREA BASE REGISTER
R14      EQU   14                    RETURN ADDRESS
R15      EQU   15                    RETURN CODE
 TITLE 'ENTRY CODE'
************************************************************************
* IKJEFLD ENTRY CODE                                                   *
*                                                                      *
* DESCRIPTIVE NAME - STANDARD ENTRY LINKAGE CODE ROUTINE               *
*                                                                      *
* FUNCTION - ESTABLISHES STANDARD LINKAGE BETWEEN PROGRAMS             *
*                                                                      *
* OPERATION -                                                          *
*                                                                      *
*    1)   PERFORMS STANDARD LINKAGE BETWEEN INVOKER OF EXIT AND        *
```

Appendix C. Accessibility

Accessible publications for this product are offered through the z/OS Information Center, which is available at www.ibm.com/systems/z/os/zos/bkserv/.

If you experience difficulty with the accessibility of any z/OS information, please send a detailed message to mhvrcfs@us.ibm.com or to the following mailing address:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size.

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the z/OS Information Center using a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol giving information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- ? means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.
- ! means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1!

(KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

- * means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

  Note:

  1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
  2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
  3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.
- + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times; that is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

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 right 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 give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: 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 states 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming interface information

This document describes intended Programming Interfaces that help the customer to maintain and customize an z/OS system. It contains information about the facilities that customers can use to tailor TSO/E functions.

Trademarks

Index

Product Number: 5650-ZOS

Printed in USA

SA32-0976-00