CICS Transaction Server for z/OS

# CICSPlex SM Application Programming Guide

*Version 3 Release 1*

SC34-6468-04

CICS Transaction Server for z/OS

# CICSPlex SM Application Programming Guide

*Version 3 Release 1*

SC34-6468-04

**Note!**

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 169.

This edition applies to Version 3 Release 1 of CICS Transaction Server for z/OS, program number 5655-M15, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

# Contents

# Preface

This book provides programming information for the CICSPlex® System Manager (CICSPlex SM) element of CICS® Transaction Server for z/OS® Version 2 Release 1. It describes how to use the application programming interface (API) to access CICSPlex SM data and services.

## Who this book is for

This book is for application programmers who want to access the services of CICSPlex SM.

## What you need to know

It is assumed that you have experience writing programs in COBOL, C, PL/I, assembler language, or REXX. You should also have knowledge of the CICSPlex SM concepts and terminology introduced in the *CICSPlex SM Concepts and Planning* book.

For guidance information on how to use the CICSPlex SM API see the *CICSPlex SM Application Programming Reference*.

While you are using this book, you will need to refer to the *CICSPlex SM Resource Tables Reference* for descriptions of the resource tables that you can access. You may also need to refer to the following books:

*CICSPlex SM Managing Business Applications*
: For information about Business Application Services definitions.

*CICSPlex SM Managing Resource Usage*
: For information about real-time analysis and Monitoring definitions.

*CICSPlex SM Managing Workloads*
: For information about Workload Manager definitions.

## How to use this book

This book provides guidance information for the CICSPlex SM API.

It introduces the API, describes the various environments it supports, and provides examples of its use. If this is your first experience with the API, it will probably help to read through the guide more or less from start to finish.

## Notes on terminology

In the text of this book, the term **CICSPlex SM** (spelled with an uppercase letter 'P') means the IBM® CICSPlex System Manager element of CICS Transaction Server for z/OS. The term **CICSplex** (spelled with a lowercase letter 'p') means the largest set of CICS systems to be managed by CICSPlex SM as a single entity. Other terms used in this book are:

| Term | Meaning |
|---|---|
| **API** | Application programming interface |
| **ASM** | Assembler language |
| **CICS** | The CICS element of the CICS Transaction Server for z/OS |

## CICS System Connectivity

This release of CICSPlex SM can be used to control CICS systems that are directly connected to it.

For this release of CICSPlex SM, the connectable CICS systems are:

- CICS Transaction Server for z/OS 3.1
- CICS Transaction Server for z/OS 2.3
- CICS Transaction Server for z/OS 2.2
- CICS Transaction Server for OS/390® 1.3

You can use this release of CICSPlex SM to control systems running supported releases of CICS that are connected to, and managed by, your previous release of CICSPlex SM. However, if you have any directly-connectable release levels of CICS, as listed above, that are connected to a previous release of CICSPlex SM, you are strongly recommended to migrate them to the current release of CICSPlex SM, to take full advantage of the enhanced management services. See the *CICS Transaction Server for z/OS Migration from CICS TS Version 2.3* for information on how to do this.

Table 1 shows which supported CICS systems can be directly connected to which releases of CICSPlex SM.

*Table 1. Directly-connectable CICS systems by CICSPlex SM release*

| **CICS system** | **CICSPlex SM component of CICS TS 3.1** | **CICSPlex SM component of CICS TS 2.3** | **CICSPlex SM component of CICS TS 2.2** | **CICSPlex SM component of CICS TS 1.3** |
|---|---|---|---|---|
| CICS TS 3.1 | Yes | No | No | No |
| CICS TS 2.3 | Yes | Yes | No | No |
| CICS TS 2.2 | Yes | Yes | Yes | No |
| CICS TS 1.3 | Yes | Yes | Yes | Yes |
| TXSeries 4.3.0.4 | No | Yes | Yes | No |
| TXSeries 5.0 | No | Yes | Yes | No |

# Summary of changes

This book is based on the CICS Transaction Server for z/OS, Version 2 Release 3 edition of *CICSPlex SM Application Programming Guide*. It has been updated to incorporate changes made for CICS Transaction Server for z/OS, Version 3 Release 1.

Changes made since the last edition are marked by vertical bars in the left margin.

## Changes made to this book for CICS Transaction Server for z/OS, Version 3 Release 1

CICSPlex SM support for the CICS for Windows component of IBM TXSeries (also known as Windows NT 4.3 and Windows NT 5.0) is no longer provided in CICS Transaction Server for z/OS, Version 3 Release 1. Therefore, it is no longer possible to set up a CICSPlex SM remote MAS agent for Windows.

However, customers, who wish to do so, can continue to use the CICS Transaction Server Version 2.3 or Version 2.2 for CICSPlex SM support of TXSeries.

## Changes for CICS Transaction Server for z/OS, Version 2 Release 3

Descriptions of three new metadata tables are included:

- METANAME
- METAPARM
- PARMAVA

Three existing metadata tables are enhanced by the addition of new fields:

- ATTR
- OBJACT
- OBJECT

The OBJACT base table now returns metadata resource records for the CICSPlex SM GET, CREATE, SET, UPDATE and REMOVE commands. Previously OBJACT only returned records for valid base table actions issued by a PERFORM command. The OBJACT base table includes a new attribute, APIPERFORM, to differentiate between the two record types. The OBJACT base table is described in “Using meta-data resource tables” on page 48

## Changes for CICS Transaction Server, Version 2 Release 2

There are changes to the following metadata table attributes in support of the CICSPlex SM remote MAS for Windows:

**Table** **Attribute**

**OBJECT**
CURVALWNT

**OBJACT**
VALCICSWNT

**ATTR** VALCICSWNT, SETCICSWNT

See “Using meta-data resource tables” on page 48 for details.

The OBJSTAT resource table has changed. See “OBJSTAT” on page 27 for details.

A new section on CRESxxxx resource tables has been added. See “Using CRESxxxx resource tables” on page 64.

There has been a change in CICSPlex SM field naming conventions in this release. Data set name fields such as DSNAME, file name fields such as LOCFILE and REMFILE, and transient data queue name fields such as EXTRATDQ and INTRATDQ are now case-sensitive. When entering data set and file names into the CICSPlex SM interfaces (end user interface, API and the web user interface), ensure that you enter the data in the correct case. In previous releases of CICSPlex SM, the data set names and file names are automatically converted to upper case.

There are no other significant changes for this edition.

## Changes for CICS Transaction Server, Version 2 Release 1

There are no significant changes for this edition.

## Major changes to this book for CICS Transaction Server for OS/390 Release 3 include:

The following additions and changes have been made to the functions of CICSPlex SM for CICS Transaction Server for OS/390 Release 3.

- The information in this book derives from the old *CICSPlex System Manager Application Programming Interface* which as been split into two books the other being the *CICSPlex System Manager Application Programming Reference*.
- The Resource Table Summary has been moved to the *CICSPlex System Manager Resource Tables Reference*.
- Chapter 4, “Dealing with exception conditions,” on page 87 has been reorganized and expanded and now includes new data in Table 8 on page 92 and a new section “An example of a BAS error result set” on page 100.
- There is a new Appendix A, “BINCONRS, BINCONSC, and BINSTERR error codes,” on page 113.

# Chapter 1. An overview of the CICSPlex SM API

The CICSPlex SM application programming interface (API) provides you with access to CICS system management information and allows you to invoke CICSPlex SM services from an external program. The API can provide a single interface for programs designed to monitor and control the CICS systems in your enterprise. In addition, the API provides an interface to CICSPlex SM itself. So you can also write programs to access the administrative functions that control the way CICSPlex SM operates.

Some typical uses of the API include:

- Monitoring key resources in your CICS environment.
- Changing the status of CICS resources relative to other conditions in your enterprise.
- Controlling the flow of change to your CICS environment.
- Passing the information provided by CICSPlex SM to an automation product.
- Developing alternative display and report formats for CICS and CICSPlex SM data.
- Processing CICSPlex SM notifications about events such as:
  - Real-time analysis thresholds being reached
- Creating and maintaining CICSPlex SM definitions for Business Application Services, for workload management, real-time analysis, and resource monitoring.
- Creating and maintaining CICS resource definitions in the CICSPlex SM data repository.

## Supported environments and languages

The API can be called from programs running in a variety of environments:

- MVS batch
- MVS TSO
- MVS NetView®
- CICS/ESA
- CICS element of CICS Transaction Server for z/OS

**Note:** The CICSPlex SM API cannot be called from within a NetView RODM method. For details on the restrictions that apply to RODM method services, see the *NetView RODM Programming Guide*.

## Available interfaces

CICSPlex SM provides two interfaces for API users:

**Command-level interface**

This interface uses the CICS translator to accept EXEC CPSM statements and translate them into the appropriate sequence of instructions in the source language. These instructions are then linked to an interface stub routine that is supplied by CICSPlex SM.

The command-level interface is available for programs written in the following languages:

- Assembler Version 2 and later

- OS PL/I Optimizing Compiler Version 2.3 and later
- COBOL Compiler Version 1.3.2 and later
- C Version 2.1. and later

Table 2 shows which languages are supported by the command-level interface in each environment.

*Table 2. Programming languages supported by the command-level interface*

| Environment | Assembler | COBOL | PL/I | C |
|---|---|---|---|---|
| CICS/ESA and CICS TS | ✓ | ✓ | ✓ | ✓ |
| MVS Batch | ✓ | ✓ | ✓ | ✓ |
| MVS TSO | ✓ | ✓ | ✓ | ✓ |
| MVS NetView | ✓ | | ✓ | ✓ |

**Run-time interface**

The run-time interface supports programs written as REXX EXECs in the following MVS environments:

- Batch
- TSO
- NetView.

This interface consists of a REXX function package that is supplied by CICSPlex SM. The function package accepts commands in the form of text strings and generates the appropriate API calls.

## Connecting to CICSPlex SM

You can think of a CICSPlex SM API program as existing in or having access to three environments:

**User environment**

The program itself and the environment in which it runs, such as MVS or CICS.

**CICSPlex SM environment**

The data that CICSPlex SM maintains and the services it provides to the program.

**Managed resource environment**

The resources that CICSPlex SM manages and which the program can access.

Before your program can access the CICSPlex SM environment and the resources it manages, you must establish a connection to CICSPlex SM. This connection is called an API processing thread and serves two basic purposes:

- When a thread is created, the user is identified so that security validation and auditing of the program's operations can take place transparently.
- There are implicit relationships between some API functions, and those relationships are maintained at the thread level. Each thread is considered a unique API user and no resources can cross the boundary of a thread.

Once a thread is created, your program can issue commands within the context of the local CMAS. The local CMAS is dictated by where and how the connect command is issued:

- If issued in a CICS system, it is the local CMAS to that CICS system.
- If issued as a batch job and no CMAS is stated explicitly, the local CMAS is the last CMAS started.
- If issued as a batch job and a CMAS within the MVS image is included in the CONNECT command, it is that CMAS.

You can look at data from CMASs other than the local CMAS but you cannot change the context to point directly to them.

A simple API program would establish only a single thread. You could establish the thread, perform the desired operations, and then terminate the thread. A more complex program might maintain several concurrent threads to perform parallel operations that would be prohibited on a single thread or to simplify the correlation of commands and results.

You can use the following commands to manage an API thread:

**CONNECT**
: Establishes a connection to CICSPlex SM, defines an API processing thread, and provides default settings for the thread. The thread is maintained by the CMAS that is supporting your API session.

**DISCONNECT**
: Disconnects an API processing thread from CICSPlex SM and releases any resources associated with the thread.

**QUALIFY**
: Defines the CICSPlex SM context and scope for subsequent commands issued by the thread.

**TERMINATE**
: Terminates all API processing on all the threads created by the CICS or MVS task that issues the command.

These commands manage the connection between the user environment (your program) and CICSPlex SM; they do not affect the managed resources. Figure 1 on page 4 illustrates the impact these commands have on the API environment.

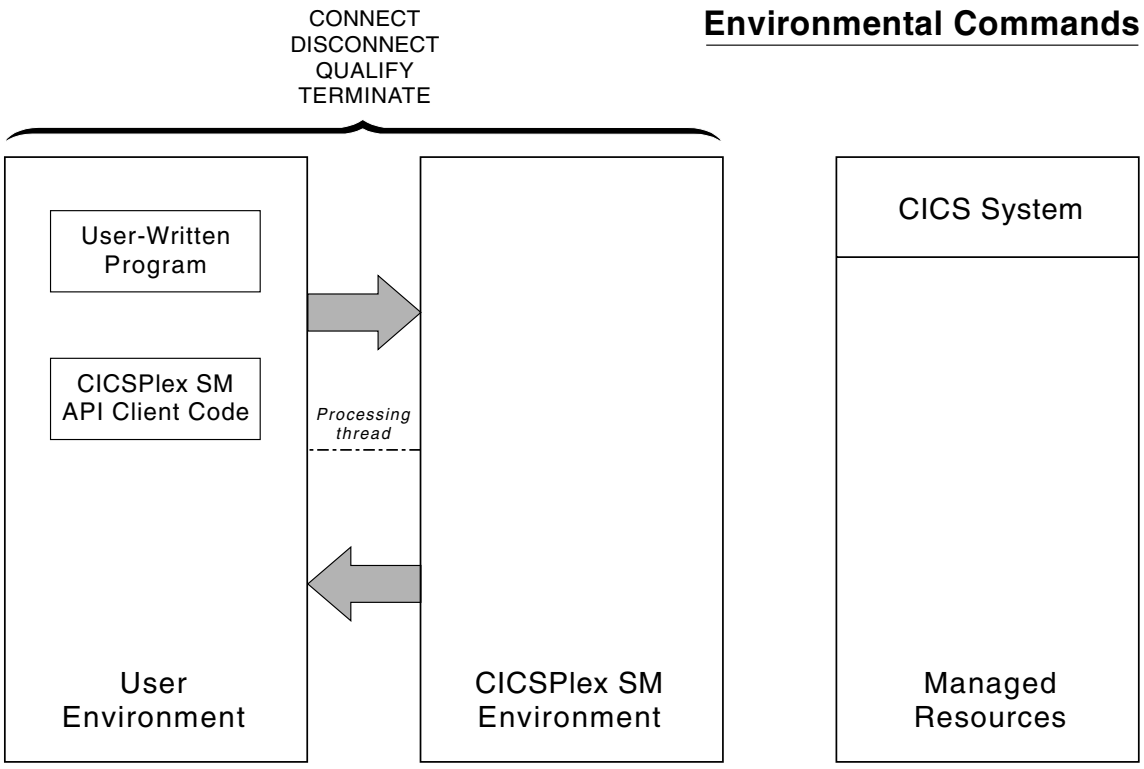



*Figure 1. API commands involved in managing a thread*

For complete descriptions of these commands, see *CICSPlex System Manager Application Programming Reference*.

## The connection process

The process of connecting to CICSPlex SM varies according to what type of program you write and where it runs. For programs written using the command-level interface, keep in mind the following requirements:

**CICS** A program written to run as a CICS application must be linked with the proper stub routine and must run in a CICS system that is being actively managed by CICSPlex SM as a local MAS.

A connection is established first to the MAS agent code that resides in the CICS system and then to the CMAS that controls that MAS. On the CONNECT command, you must specify a CONTEXT of the local CMAS.

**Batch, NetView, or TSO**
A program written to run as a batch job or under NetView or TSO must be linked with the proper stub routine and must run in the same MVS image as the CMAS to which you want to connect.

In these environments, if there is more than one CMAS in the MVS image, the API selects a suitable CMAS and establishes a connection. The following rules apply to the selection of a CMAS:

- The CMAS must be running the same version of CICSPlex SM as the run-time module (EYU9AB00).
- If the context specified on the CONNECT command is a CMAS, CICSPlex SM connects to that CMAS. If that CMAS is either not active or not running the appropriate version of CICSPlex SM, the CONNECT command fails.

- If the context specified on CONNECT is a CICSplex, CICSPlex SM selects a CMAS running the appropriate version that participates in the management of the CICSplex.
- If no context is specified on CONNECT, CICSPlex SM connects to the CMAS that was most recently started, provided it is running the appropriate version of CICSPlex SM.

The CICSPlex SM API also supports another type of batch environment. A program can issue API commands from an address space that is running a CICS system without itself being a CICS transaction. In other words, the program can run as a separate MVS task in the same address space as the CICS system. This type of program must be linked with the batch environment stub routine and the connection process is the same as for other batch programs.

**Note:** A program that is a CICS transaction must be run in a CICS system that is a CICSPlex SM local MAS.

For details on the stub routines that are required for each of these environments, see “Link editing your program” on page 83.

**Note:** For programs written in REXX, the connection process is the same as for a command-level program that runs in the same environment (batch, TSO, or NetView). No stub routine is required, but the REXX function package that is supplied by CICSPlex SM must have been properly installed.

**MVS restrictions:** Upon successful completion of a CONNECT request, a thread token is returned to the user. All subsequent commands referring to this thread token must be issued from the same MVS TCB that issued the comnnection request

## Security considerations

When an API program requests a connection to CICSPlex SM, the CMAS being connected to attempts to extract user authorization data from the environment. How the connection is established depends upon whether such authorization data exists and whether security is active in the CMAS:

**If security exists**

Regardless of whether CMAS security is active, if a security environment exists where the API program is running:

- The API security routine, EYU9XESV, is not called.
- The USER and SIGNONPARM options on the CONNECT command are ignored.
- The API program is connected with the user ID of the invoking user, as obtained from the accessor environment element (ACEE).

**Note:** If CMAS security is not active, the ACEE user ID is not validated by CICSPlex SM.

This type of security environment may exist when a program runs under TSO, batch, NetView, or a local MAS where CICS security is active. Security checking is performed by the environment where the API program is running.

**If security does not exist and CMAS security is not active**

- The API security routine, EYU9XESV, is not called.

- The USER and SIGNONPARM options on the CONNECT command are ignored.
- No signon is performed. However, the user ID specified in the XESV_CONN_USERID field of the security routine parameter block, EYUBXESV, is associated with the connection.

This type of security environment may exist when a program runs under a local MAS where CICS security is not active. Since CMAS security is not active, no security checking is performed.

**If security does not exist and CMAS security is active**

- The API security routine, EYU9XESV, is called.
- The USER and SIGNONPARM values from the CONNECT command are passed to EYU9XESV.
- A signon is performed using the user ID returned by EYU9XESV, but no password checking is performed. By default, EYU9XESV returns the default CICS user ID for the CMAS (the DFLT_UID value).

This type of security environment may exist when a program runs under a local MAS where CICS security is not active. Since CMAS security is active, security checking is performed by EYU9XESV.

Table 3 summarizes the levels of API security and the conditions under which they are implemented.

*Table 3. Possible API security environments*

| Environment Security | CMAS Security | No CMAS Security |
|---|---|---|
| YES | EYU9XESV not called. CONNECT options ignored. User ID=ACEE. | EYU9XESV not called. CONNECT options ignored. User ID=ACEE (not checked). |
| NO | EYU9XESV called. CONNECT options passed. User ID=As returned by EYU9XESV (signon with no password checking). | EYU9XESV not called. CONNECT options ignored. User ID= XESV_CONN_USERID (no signon). |

For a description of the USER and SIGNONPARM options, see the API CONNECT Command in the *CICSPlex System Manager Application Programming Reference*. For a description of EYU9XESV and information on customizing this security routine, see *CICS RACF® Security Guide*.

## Compatibility between environments

Once you have written a CICSPlex SM API program to run in one environment, you can take that program and run it in another environment with only minor modifications.

For example, if you want to take a CICS application written with EXEC CPSM commands and convert it to an MVS batch program, you should:

- Make the appropriate code changes, such as:
  - Remove any EXEC CICS commands that may be included
  - Add the necessary MVS calls
- Relink-edit the program with the batch environment stub routine.

**Note:** A REXX program can be moved from one MVS environment (batch, TSO, or NetView) to another without modification, provided you have not used any environment-specific functions.

Before you try to move an EXEC CPSM program to an environment other than the one for which it was written, you should review the following sections:


- "Language and environment considerations" on page 78
- "Translating your program" on page 80
- "Link editing your program" on page 83.


## Compatibility between releases of CICSPlex SM

Once you have written an API program to run under one release of CICSPlex SM, you can continue to access the data provided by that release, or you can access the data available from a later release of the product. In general, if you plan to access more than one release of the CICSPlex SM API, keep the following in mind:

**Run-time environment**

The run-time version of a CICSPlex SM API program is equal to the level of the CMAS to which it connects:

- For a program written to run as a CICS application, the run-time version is that of the CMAS to which the MAS is connected.
- For a program written to run as a batch job or under NetView or TSO, the version is determined by the version of the CICSPlex SM run-time module (EYU9AB00), which is distributed in the version's SEYUAUTH library.

The run-time version of a program must be greater than or equal to:

- The version of the stub routine module (EYU9AxSI) with which the program was link edited.
  - For CICS programs, the stub module is called EYU9AMSI and is distributed in the version's SEYULOAD library.
  - For batch, TSO, or NetView programs, the module is called EYU9ABSI and is distributed in the version's SEYUAUTH library.

  In addition, the version of the stub module for any separately link edited and called programs must be the same as the version used to link edit the program that issued the CONNECT command.
- The value specified on the VERSION option of the CONNECT command.

**Note:** For programs written in REXX, the run-time version must be greater than or equal to the version of the function package (EYU9AR00), which is distributed in the version's SEYUAUTH library.

**VERSION option**

The VERSION option on the CONNECT command controls which release of CICSPlex SM resource tables are available to your program (resource tables are the external representation of CICSPlex SM data).

- An API program cannot access data from a release of CICSPlex SM earlier than Release 2 (the release in which the API was introduced). The VERSION value must be set to 0120 or greater.
- An API program cannot access data from a release of CICSPlex SM later than the run-time module that you specify. The VERSION value must be less than or equal to the release of the run-time module.

- An API program can access data from a later release of CICSPlex SM than that which the program was originally written for, provided:
  - You compile your program using the appropriate copy books for the version specified.
  - Your program is compatible with the copy books for the version specified.

**CONTEXT option**

The CONTEXT option that is supported by various API commands determines which CICS systems your program receives data from. The CONTEXT value can be set to any CMAS or CICSplex running any currently supported release of CICSPlex SM. Note, however, that the release level of the CMAS or CICSplex must be the same as the release of the run-time module.

**CURRENT option**

When specifying the CURRENT option, the record pointer does not move (that is, a subsequent FETCH retrieves the same record). Previously, the record pointer moved to the next record. For further information, see "Positioning the record pointer in a result set" on page 30.

## Special considerations for REXX applications

If you have REXX application programs you should be aware of how CICSPlex SM behaves in the case where you apply a PTF to some members of a CICSplex but not others, you modify a REXX API program to put a value in a new table field introduced by the PTF and the REXX program then connects to a CMAS which has not had the PTF applied, and which therefore has no definition for the new field.

In this case:

1. The CMAS does not transmit the value of the new field to the maintenance point CMAS.
2. The maintenance point CMAS (which is at the highest level) transforms the record area to give a default value to the new field. The new value may be different from that originally specified by the REXX program.
3. The maintenance point CMAS then broadcasts the record back to the originating CMAS, but transforms the record back to remove the new field. At this point, the maintenance point repository will not contain the intended value, (it will contain the default value) and when it has broadcast the record back to the originating CMAS, this repository will have had the intended value removed.
4. If the same REXX program issues a TPARSE of the record, the value of the field is still the same as it was at the time it was created, and is not changed by the TPARSE. This might cause the program to indicate, wrongly, that the field contains the intended value, whereas, in the maintenance point repository, the field has the default value, and in the back-level CMAS repository, the field does not exist.
5. If a REXX API program subsequently connects to the back-level CMAS and issues a TPARSE of the record, the new field will not be populated by the TPARSE. In this case the field will have the normal REXX default value - the field value will be the same as the field name.

If the set of circumstances described above applies to you, and might cause you a problem, your REXX program should contain code to issue a QUERY to obtain and verify the record length.

## Migrating applications to a new release

In order to migrate your application programs to the new release so that they can benefit from the full function available there, see "Accessing resource tables from a new release." If you need to continue to run application programs in an earlier release CICSPlex SM environment whilst being able to manage this environment using the services provided in the new release, you must first read the sections on CICSPlex SM migration in the *CICS Transaction Server for z/OS Migration from CICS TS Version 2.3*.

## Accessing resource tables from a new release

You can access the most up-to-date CICSPlex SM resource tables by running an existing program under a new release of the API.

**Note:** To take full advantage of a new CICSPlex SM function (such as Business Application Services), however, you would have to modify an existing program or create a new one.

To run an existing API program under a new release of CICSPlex SM:

- Make sure the following are available to your program:
  - The run-time module for the new release (EYU9AB00 from the new release's SEYUAUTH library)
  - A CMAS that is running the new release
- Change the VERSION value on the CONNECT command to reflect the new release of CICSPlex SM (for example, 0310 for CICS Transaction Server for z/OS, Version 3 Release 1) and relink-edit the program using the stub module supplied in the new release.
- Review the possible effects of any changes to the CICSPlex SM resource tables.

  Attributes may be added to a resource table in a new release, which could affect your program's references to that table. And with the addition or modification of attributes, the length of a given resource table may change from one release to another. The resource table copy books that are distributed with the new release are a good source of information about such changes.

  **Note:** If there is no requirement to take advantage of the new function in the release it is possible to continue to run an existing API program unaltered, provided the VERSION value on the CONNECT command reflects the link-edit level used.

  If your program receives RESPONSE and REASON values of INVALIDPARM LENGTH when you run it under a new release of CICSPlex SM, the table length may have increased and your data buffer may not be long enough to accommodate the new resource table records.
- If you are using customized views of resource tables, you are advised to check that the names of any new resource tables do not duplicate the names of your customized views, as this could affect your processing. For further details, see "Building a customized resource table record" on page 17.

  For a complete list of new and changed resource tables in a given release, refer to *CICSPlex System Manager Resource Tables Reference*.

## Accessing resource tables from a previous release

To continue accessing the resource tables supplied with a previous release of CICSPlex SM:

- Specify the release of CICSPlex SM data that you want to access on the VERSION option of the CONNECT command.
- Use the run-time module (EYU9AB00) supplied with the release you want to access or a subsequent release that supports it.
- Use a version of the stub module (EYU9AxSI) that is less than or equal to the run-time module.

Table 4 illustrates some valid combinations of the VERSION option, stub module and run-time module for accessing data from different releases of CICSPlex SM.

*Table 4. Valid ways to access data from different releases*

| VERSION value | Stub module (EYU9AxSI) | Run-time module (EYU9AB00) | CMASs available | CMAS used | Data available |
|---|---|---|---|---|---|
| 0220 | V2R2 | V2R2 | V1R2 V1R3 V1R4 V2R1 V2R2 | V2R2 | V2R2 |
| 0120 | V1R2 | V1R2 | V1R2 | V1R2 | V1R2 |
| 0120 | V1R2 | V1R3 | V1R3 | V1R3 | V1R2 |
| 0120 | V1R3 | V1R3 | V1R2 V1R3 | V1R3 | V1R2 |
| 0120 | V1R4 | V1R4 | V1R2 V1R3 V1R4 | V1R2 | V1R2 |
| 0130 | V1R3 | V1R3 | V1R2 V1R3 | V1R3 | V1R3 |
| 0130 | V1R3 | V1R4 | V1R2 V1R3 V1R4 | V1R4 | V1R3 |
| 0130 | V1R4 | V1R4 | V1R2 V1R3 V1R4 | V1R4 | V1R3 |
| 0140 | V1R4 | V1R4 | V1R2 V1R3 V1R4 | V1R4 | V1R4 |
| 0210 | V2R1 | V2R1 | V1R2 V1R3 V1R4 V2R1 | V2R1 | V2R1 |

Table 5 shows some invalid combinations of the VERSION option, run-time module, and stub module and describes why they produce an error.

*Table 5. Common errors in accessing different releases*

| VERSION value | Stub module (EYU9AxSI) | Run-time module (EYU9AB00) | CMASs available | Error description |
|---|---|---|---|---|
| 0140 | V2R1 | V1R4 | V1R4 V2R1 | Stub module release level is greater than run-time module. |
| 0210 | V2R1 | V1R4 | V1R4 V2R1 | Stub module release level is greater than run-time module. |
| 0210 | V1R4 | V1R4 | V1R4 | VERSION value is greater than run-time module. |
| 0210 | V2R1 | V2R1 | V1R4 | No CMAS available at the required run-time level. |

**Note:** For programs written in REXX, the compatibility issues are similar. The Release 2 function package (which contains the necessary stub module) can run successfully with either the Release 2 or Release 3 run-time module. The Release 3 function package, however, cannot run with the Release 2 run-time module; the Release 3 module is required.

## Sample programs

Sample programs for each supported language are distributed with CICSPlex SM in source form. These samples are provided to illustrate the types of programs you can write and the commands you need to use in those programs.

The sample programs are distributed in members called EYUxAPIn, where x is a 1-character language identifier and n is a sequential program identifier. For example, EYUCAPI1 is sample program number 1 coded in C. They are all located in the SEYUSAMP library.

Details of the sample programs are shown in the following table.

*Table 6. Sample programs provided with CICSPlex SM*

| Language | Programs | Library |
|---|---|---|
| Assembler | EYUAAPI1 EYUAAPI2 EYUAAPI3 | SEYUSAMP |
| COBOL | EYULAPI1 EYULAPI2 EYULAPI4 | SEYUSAMP |
| PL/I | EYUPAPI1 EYUPAPI2 | SEYUSAMP |
| C | EYUCAPI1 EYUCAPI2 | SEYUSAMP |
| REXX | EYU#API1 EYU#API2 EYU#API3 | SEYUSAMP |

A listing is provided for each sample program (in one of its supported languages) in Appendix B, “Sample program listings,” on page 115.

**Note:** Additional sample CICSPlex SM API programs are available via the IBM CICS SupportPacs system at:

`http://www.ibm.com/software/htp/cics/downloads`

## Where to find more information

This book is divided into three parts. Background information on the CICSPlex SM API and guidance for writing an API program are contained in the following chapter in part one:

- Chapter 2, “Using the CICSPlex SM API,” on page 13 introduces a variety of concepts that you need to understand before you use the API.

Part two deals with EXEC CPSM programming considerations:

- Chapter 3, “Writing an EXEC CPSM program,” on page 67 describes how to write an API program using the command-level interface.
- Chapter 4, “Dealing with exception conditions,” on page 87 describes the tools and techniques you can use to handle errors in a CICSPlex SM API program.

Part three deals with REXX programming considerations:

- Chapter 5, “Writing a REXX program,” on page 103 describes how to write an API program using the REXX run-time interface.
- Chapter 6, “REXX error handling,” on page 109 explains how to handle REXX errors.

Complete descriptions of the API commands, as well as some REXX-specific functions and commands, are contained in the *CICSPlex System Manager Application Programming Reference* .

# Chapter 2. Using the CICSPlex SM API

This chapter introduces a variety of concepts that you need to understand before you use the CICSPlex SM API, including managed objects, resource tables, and result sets.

## CICSPlex SM managed objects

CICSPlex SM is an object-oriented system. This means that each resource in the CICSPlex SM environment is an instance of an object. Each object is considered to be a specific type and each has a unique, formally defined name.

## Types of managed objects

There are various types of objects in the CICSPlex SM environment. Some objects, such as CICS systems, programs, and transactions are real-world resources that CICSPlex SM manages. Definition objects, such as monitor specifications and workload definitions, are resources created solely for use within CICSPlex SM. An event is an example of a run-time object that is generated as a result of CICSPlex SM processing.

The CICSPlex SM managed objects can be grouped into the following categories:

- Managed CICS resources
  - CICS resources
  - Monitored CICS resources
- CICS resource definitions
- CICSPlex SM definitions
- CICSPlex SM manager resources
- CICSPlex SM notifications
- CICSPlex SM meta-data.

### Managed CICS resources

These objects represent actual CICS resources that exist in the CICS systems being managed by CICSPlex SM. Each object of this type describes a CICS resource that CICSPlex SM can report on and manipulate. Managed objects exist for all the resources that are available to CICSPlex SM using standard CICS interfaces. In some cases, the CICSPlex SM managed objects offer a more definitive representation of the resources than CICS does. For example, the LOCTRAN and REMTRAN objects, which CICSPlex SM uses to distinguish between local transactions and remote transactions, are combined by CICS as transactions.

In addition to the standard CICS resources, CICSPlex SM creates managed objects as a result of its resource monitoring activity. Monitored CICS resources contain a subset of the resource attributes, normally those that reflect the state and consumption characteristics of the resource. In addition, CICSPlex SM may provide derived attributes that show resource utilization as an average, rate, or percentage. MLOCTRAN and MREMTRAN are examples of monitored CICS resource objects; they are derived from the LOCTRAN and REMTRAN CICS resource objects. A monitored CICS resource object can exist after the associated CICS resource object is removed from the CICS system, or even after the system itself is shut down.

## CICS resource definitions

These objects represent definitions of CICS resources that CICSPlex SM can assign to, and possibly install in, CICS systems. The actual definitions are stored in the CICSPlex SM data repository as definition records. For example, the TRANDEF object represents a CICS transaction that can be assigned both locally and remotely to multiple CICS systems throughout the CICSplex.

Assigning CICS resources to CICS systems enables CICSPlex SM to manage those resources as a logical group, such as an application. In addition, CICSPlex SM can actually install instances of a resource in CICS systems that support the EXEC CICS CREATE command.

## CICSPlex SM definitions

These objects represent the definitions that are used by CICSPlex SM management applications. The actual definitions are stored in the CICSPlex SM data repository as definition records. For example, the MONSPEC object represents a user-defined monitor specification that CICSPlex SM uses to establish resource monitoring in a CICS system.

Any changes you make to CICSPlex SM definitions are automatically distributed throughout the CICSplex. In addition, certain definitions are bound to other definitions for the purpose of referential integrity. If you remove one of these definitions, all the related definitions are also removed. For example, removing a CPLEXDEF object causes all definition objects for that CICSplex to be automatically removed from all CMASs that manage the CICSplex.

## CICSPlex SM manager resources

These objects represent run-time resources that are either built from CICSPlex SM definitions or created by CICSPlex SM management applications during processing. You can manipulate a CICSPlex SM manager resource without necessarily affecting the underlying definition. The RTAACTV object is an example of a CICSPlex SM manager resource; it describes the currently installed RTADEF and STATDEF definition objects.

There are other CICSPlex SM manager resources that are not directly related to any definition. For example, the CRESCONN object is a Topology Services resource map that describes the CICS connections in an active MAS.

## CICSPlex SM notifications

CICSPlex SM notifications are really messages that are generated asynchronously by a CICSPlex SM managed object. Notifications describe an interesting event related to the object. CICSPlex SM manager resources can register interest in one or more of these events. When a notification is generated, the manager resource performs whatever processing is needed based on the event that occurred.

An API program can also register interest in events that generate CICSPlex SM notifications. The EMSTATUS, EMASSICK, and EMASWELL objects are examples of notification messages generated by the CICSPlex SM MAS agent. These notifications describe the current state of the MAS.

The ERMCxxxx objects are generated by CICSPlex SM when a Topology resource map is changed. CICSPlex SM maintains resource maps which describe the topology of certain CICS resources in the MASs. CICS resources for which resource maps are maintained have a corresponding ERMCxxxx notification object. The CICSPlex SM agent detects the installs and discards of these CICS resources and causes the Topology resource map to be updated. For example, if a file

definition is installed in a MAS, the Topology resource map will be changed and an ERMCFILE notification will be generated. The ACTION attribute of the ERMCFILE notification indicates that an install has occurred. Furthermore, for a local MAS, the CICSPlex SM MAS agent detects updates to these CICS resources. For example, if a program is disabled, the ERMCPRGM notification will be generated with the ACTION attribute indicating an update.

### CICSPlex SM meta-data

These objects describe the structure of CICSPlex SM managed resources. This information is maintained in an object directory that exists in each active CMAS.

An API program can request the following types of meta-data from the object directory:

**OBJECT**
General characteristics of an object

**OBJACT**
Valid actions for an object

**METADESC**
Basic description of an object's attributes

**ATTR** Complete description of an object's attributes

**ATTRAVA**
Valid EYUDA or CVDA values for an attribute

**METANAME**
All CVDAS, CVDAT, and EYUDA information

**METAPARM**
Description of a parameter for an action

**PARMAVA**
Description of the values allowed for a parameter

## CICSPlex SM resource tables

Each CICSPlex SM managed object is represented externally by a resource table. A resource table defines all the attributes of an object. The attributes represent the collection of data that is available for that object.

The formal object name is used as the name of the resource table that describes the object's attributes. You identify an object in your API program by specifying its resource table name. For example, to find out about the programs in one or more CICS systems, you could access the PROGRAM object. PROGRAM is the name of the CICSPlex SM resource table that describes CICS programs.

Each instance of an object is formatted as a resource table record that describes an actual resource in the CICSPlex SM environment. The object attributes are presented in the individual fields of a resource table record. It is important to note that a resource table is not itself an object. A resource table record is merely the format in which information about a managed object is returned by CICSPlex SM. This information includes the current attribute values, the actions that the object supports, and the releases of CICS for which the object is valid.

There is a resource table type for each type of CICSPlex SM managed object:

| Resource table type | Object type |
|---|---|
| **CICS Definition** | CICS resource definitions |
| **CICS Resource** | CICS resources |
| **CICS Monitored** | Monitored CICS resources |
| **CPSM Definition** | CICSPlex SM definitions |
| **CPSM Manager** | CICSPlex SM manager resources |
| **CPSM Notification** | CICSPlex SM notifications |
| **CPSM MetaData** | CICSPlex SM meta-data |
| **CPSM Configuration** | CICSPlex SM configuration definitions |

For a summary of the CICSPlex SM resource tables by type and complete descriptions of specific resource tables see the Resource table summary in the *CICSPlex System Manager Resource Tables Reference*.

**Restricted Resource Table Attributes**

Certain attributes in the CICSPlex SM resource tables are for internal use only; they cannot be modified or manipulated by an API program.

In CICS Resource and CICS Monitored tables, CICSPlex SM uses the following attributes to identify uniquely which CICS system contains the resource:

- EYU_CICSNAME
- EYU_CICSREL.

These attributes are included in every CICS Resource and CICS Monitored resource table record. You can specify these attributes in a GROUP command to summarize the records in a result set. However, you should not specify these attributes in an ORDER, SPECIFY FILTER, or SPECIFY VIEW command.

CPSM Definition and CICS Definition tables include a CHANGETIME attribute, which reflects the date and time at which the definition was last modified. CICS Definition tables also include a CREATETIME attribute, which is the date and time at which the definition was created. CICSPlex SM is solely responsible for maintaining the CHANGETIME and CREATETIME attributes; you should not attempt to modify these attribute values.

## Building a customized resource table record

Normally, when you create a result set, each resource table record contains the complete set of attributes in the format defined by CICSPlex SM. There may be times, however, when you want to work with a subset of those attributes or work with them in a different order. The SPECIFY VIEW command lets you decide which attributes of a resource table to include in a record and what order to present them in. In effect, you are building a temporary, custom-made resource table.

You can build views only for resource tables with a type of CICS Resource; you cannot build views for any other type of resource table. Also, a view can be built from the attributes of only one resource table at a time. You cannot combine attributes from different resource tables into a single view.

When you build a resource table view, you have to give it a name. The name you assign to a view takes precedence over any existing resource table names. This means you can redefine an existing resource table name to represent a subset of the attributes in a different order than they appear in the original table.

For ease of maintenance of your programs, you are recommended to give unique names to your customized resource table views. If you do not use unique names, you should be aware that you cannot access another view with the same name in the same processing thread, without the programming overhead of discarding the original view. You should also check, when you upgrade your version of CICSPlex SM, that any new resource tables do not duplicate your customized view names.

To tell CICSPlex SM which resource table attributes you want to include and in what order, you specify an order expression on the FIELDS option of the SPECIFY VIEW command. This expression is similar to the one you use when sorting records in a result set with the ORDER command. The order expression consists of a list of the attributes to be included in the view.

The syntax of an order expression for building a view is:

**Order Expression – Building a View**


where:

**attr** Is the name of an attribute in the resource table.

You can specify as many attribute names as you like, but the total length of an order expression, including commas and blank spaces, must not exceed 255 characters. If you do not specify an attribute name, the order expression will, by default, contain the name of the first attribute in the resource table, for example, the JOBNAME attribute in the CICSRGN resource table.

**Notes:**

1. You cannot specify the EYU_CICSNAME or EYU_CICSREL attributes in an order expression. To identify the CICS system from which a view record was collected, retrieve the OBJSTAT information along with the data. You can do this by specifying the BOTH option on the FETCH command.

2. An order expression must be followed by either blank spaces or null characters to the end of the specified buffer. That is, the buffer length you specify (using the LENGTH option) should not include any data other than an order expression.

For example, to build a limited view of the LOCTRAN resource table, you could specify:

```
TRANID,STATUS,USECOUNT,PROGRAM,PRIORITY,TRANCLASS.
```

Once a view is built, you can specify it on the OBJECT option of a GET command, just as you would the resource table itself. The resource table records returned by GET include only those attributes you named in the order expression on the SPECIFY VIEW command.

Any views that you build are associated with the specific processing thread on which you build them; they cannot be shared by other processing threads. When you terminate your processing thread, any views you built on it are discarded. You can also choose to discard a view at any time by using the DISCARD command.

## How to create copybooks for customized resource table records

You can build a structure for your customized view by using the SPECIFY VIEW, GET and FETCH commands to move the data into your structure. For example:

```
*****************************************************
*            SPECIFY VIEW                           *
*****************************************************
     STRING 'POOLNAME,MINITEMLEN,QUELENGTH,NUMITEMS,'
     'RECOVSTATUS,MAXITEMLEN,LASTUSEDINT,'
     'NAME,TRANSID,LOCATION.'
     DELIMITED BY SIZE INTO BUFFERA.
     MOVE 96 TO BUFFERL.
     EXEC CPSM SPECIFY
               VIEW('VTSQSHR')
               FIELDS(BUFFERA)
               LENGTH(BUFFERL)
               OBJECT('TSQSHR')
               THREAD(TTKN(1))
               RESPONSE(SMRESP)
               REASON(SMRESP2)
     END-EXEC.
```

*Figure 2. SPECIFY VIEW command to build a structure*

The associated structure will consist of each attribute specified in the SPECIFY VIEW FIELDS keyword and is shown in Figure 3 on page 19.

```
 01 VTSQSHR.
* Shared Temporary Storage Queue
   02 POOLNAME       PIC X(0008).
* TS Pool Name
   02 MINITEMLEN     PIC S9(0004) USAGE BINARY.
* Smallest item Length in bytes
   02 QUELENGTH      PIC S9(0008) USAGE BINARY.
* Total length in bytes . FLENGT
   02 NUMITEMS       PIC S9(0004) USAGE BINARY.
* Number items in queue
   02 RECOVSTATUS    PIC S9(0008) USAGE BINARY.
* Recovery Status
   02 MAXITEMLEN     PIC S9(0004) USAGE BINARY.
* Largest item length in bytes
   02 LASTUSEDINT    PIC S9(0008) USAGE BINARY.
* Interval since last use
   02 NAME-R         PIC X(0016).
* Queue Name                        -- RESERVED WORD --
   02 TRANSID        PIC X(0004).
* Trans that created tsqueue
   02 LOCATION       PIC S9(0008) USAGE BINARY.
* Queue Location
```

*Figure 3. Structure of a customized view*

Note that the EYU-CICSNAME, EYU-CICSREL, and EYU-RESERVED attributes or any field alignment or padding attributes are not used in this structure.

## Selecting managed objects

Any given API program is likely to be interested in only a subset of the CICSPlex SM managed objects. You can identify the managed objects you want to work with by:

- Setting the context and scope for your program
- Using filter expressions on individual commands.

## Setting the context and scope

The set of managed objects that your API program can work with is determined primarily by the context and scope associated with the processing thread. As with all CICSPlex SM operations, the context and scope of an API program identify the CICS systems on which the program can act.

In general, you can set the context and scope values as follows:

**CONTEXT**

For most operations in a CICSplex, the context is the name of the CICSplex. For operations related to CMAS configuration (such as defining CICSplexes or CMAS communication links), the context must be a CMAS name and for applications executing in a CICS local MAS the CMAS name must be the local CMAS name.

**SCOPE**

When the context is a CICSplex, the scope can be:

- The CICSplex itself
- A CICS system or CICS system group within the CICSplex
- A logical scope, as defined in a CICSPlex SM resource description (RESDESC)

When the context is a CMAS, the scope value is ignored. There are also a number of resources for which the scope value is ignored. These are

identified in the description of resource tables in *CICSPlex System Manager Resource Tables Reference* by the **SCOPE applies** field.

You can set a default context and scope for your program by using one of these commands:

**CONNECT**
Defines a default context and scope when the API processing thread is established.

**QUALIFY**
Changes the default context and scope for subsequent commands issued on the thread.

The values you set on either of these commands are in effect for all API commands that use context and scope.

Alternatively, you can specify context and scope values for individual API commands. The following commands support one or both of the CONTEXT and SCOPE options:

- CREATE
- GET
- LISTEN
- PERFORM OBJECT
- REMOVE
- UPDATE.

The context and scope values you set on any of these commands are in effect for that command alone. If you specified a default context and scope for the thread, the values on any of these commands temporarily override the default values. If you did not specify a default context and scope and you issue a command that expects these values (such as GET), you must specify a context and scope on the command.

## Using filter expressions

A request for CICSPlex SM managed object data can produce a large number of resource table records. The default is to return all the resource table records that exist for a given object within the current context and scope. For example, if you ask for PROGRAM object data, you receive a resource table record for every program in every CICS system in the current context and scope. However, if you are only interested in certain programs, you can use a filter expression to limit the number of records returned based on the current values of certain PROGRAM attributes.

### How you can use filter expressions

You can use filter expressions in one of two ways:

- With the CRITERIA option of a GET or PERFORM OBJECT command to filter the resource table records returned by that command. The filter expression is used only once and is discarded when the command that used it completes its processing.
- With a SPECIFY FILTER command to define a filter that can be used repeatedly.

  Once a filter is defined, you can use it with these commands to limit the resource table records being processed:

  – COPY

- DELETE
- FETCH
- GET
- GROUP
- LISTEN
- LOCATE
- MARK
- PERFORM OBJECT
- PERFORM SET
- REFRESH
- SET
- UNMARK

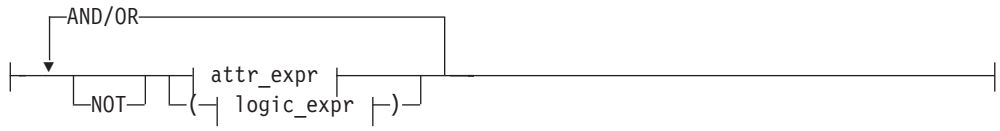
A filter expression that you define with the SPECIFY FILTER command is available to your program until you either discard it (with the DISCARD command) or terminate the processing thread.

## How to build a filter expression

A filter expression is a character string that defines logical expressions to be used in filtering resource table records. A filter expression can be made up of one or more attribute expressions in the form:

**Filter Expression**


**logic_expr:**




**attr_expr:**


where:

**attr** Is the name of an attribute in the resource table.

You can name the same attribute more than once in a filter expression.

**Note:** In a filter expression you cannot specify the EYU_CICSNAME or EYU_CICSREL attributes, or attributes with a maximum length over 256 bytes.

**oper** Is one of the following comparison operators:

- **<** Less than
- **<=** Less than or equal to
- **=** Equal to
- **>=** Greater than or equal to
- **>** Greater than
- **¬=** Not equal to

**value** Is the value for which the attribute is being tested. The value must be a valid one for the resource table attribute.

**Generic values**

If the attribute accepts character data, this value can be a generic. Generic values can contain:

- An asterisk (*), to represent any number of characters, including zero. The asterisk must be the last or only character in the specified value. For example:

  ```
  TRANID=PAY*.
  ```

- A plus sign (+), to represent a single character. A + can appear in one or more positions in the specified value. For example:

  ```
  TRANID=P++9.
  ```

**Notes:**

1. Generic value checking is applied only to the filter value. For example, a filter value of USERID=S* returns resource table records that have a user ID starting with S. However, a filter value of USERID=SMITH does not return resource table records that appear to contain generic characters, for example, those with a user ID of S*.
2. For hexadecimal data types, the data must be converted to hexadecimal prior to appending the asterisk (*) for the generic search. The plus sign (+) is not supported for hexadecimal data types.
3. The Web User Interface does not support the use of generic characters in attribute filters in WLM active views such as EYUSTARTWLMATAFF.

**Imbedded blanks or special characters**

If the value contains imbedded blanks or special characters (such as periods, commas, or equal signs), the entire value string must be enclosed in single quotes. For example:

```
TERMID='Z AB'.
```

To include a single quote or apostrophe in a value, you must repeat the character, like this:

```
DESCRIPTION='October''s Payroll'.
```

**Note:** Be sure to consider the quoting conventions of your programming language when using single quotes in a CICSPlex SM value string.

**Hexadecimal data**

If the attribute has a datatype of HEX the value must be in hexadecimal notation.

For example, the NAME attribute of the REQID resource table is a HEX datatype. To specify a name equal to 01234567 the value, using hexadecimal notation, would be

```
NAME=F0F1F2F3F4F5F6F7.
```

**AND/OR**

Combines attribute expressions into compound logic expressions using the logical operators AND and OR, like this:

```
attr_expr AND attr_expr.
```

Filter expressions are evaluated from left to right. You can use parentheses to vary the meaning of a filter expression. For example, this expression:

```
attr_expr AND (attr_expr OR attr_expr).
```

has a different meaning than this one:

```
(attr_expr AND attr_expr) OR attr_expr.
```

**NOT** Negates one or more attribute expressions.

You can negate a single attribute expression, like this:

```
NOT attr_expr.
```

You can also negate multiple attribute expressions or even a whole filter expression, like this:

```
NOT (attr_expr OR attr_expr).
```

Note that you must place parentheses around the attribute expressions (or the filter expression) to be negated.

**Note:** A filter expression must be followed by either blank spaces or null characters to the end of the specified buffer. That is, the buffer length you specify (using the LENGTH option) should not include any data other than a filter expression.

For example, the following is a simple filter expression that you could use to select LOCTRAN objects representing local transactions that are enabled and have a storage violation count greater than zero:

```
STATUS=ENABLED AND STGVCNT>0.
```

You can build more complex filter expressions to select objects with a very specific combination of attributes. For example, to select LOCTRAN objects that:

- Have a transaction ID starting with P
- Have a program name starting with PAY
- Are enabled
- Have a nonzero use count and storage violations, or have been restarted.

you could specify a filter expression like this:

```
(TRANID=P* AND PROGRAM=PAY* AND STATUS=ENABLED) AND
((USECOUNT>0 AND STGVCNT>0) OR NOT RESTARTCNT=0).
```

Note that the RESTARTCNT attribute in this example could also have been specified with the greater than operator instead of the NOT operator.

## Parameter expressions

Some objects support additional capabilities (in addition to facilities provided by filters) in order to reduce the number of records in a result set. If this is the case, parameters will be documented in the *CICSPlex System Manager Resource Tables Reference* describing parameters for GET.

The syntax to be used for these parameters is the same as that described in “Modifying managed resources” on page 38. For example, to build a result set containing completed task history (HTASK) records that finished between 05:00pm and 05:05pm on the 17th of July 2006, use the following parameter expression:

```
PARM('STARTDATE(07/17/2006) STARTTIME(17:00)
     INTERVAL(300).')
```

# Working with result sets

CICSPlex SM places the resource table records that you select in a result set. A result set is a logical group of resource table records that can be accessed, reviewed, and manipulated by an API program.

A result set can be created in one of two ways:

- By a direct API request to obtain resource data. The GET command is the primary means of collecting resource data and creating a result set.
- By an API request that manipulates one result set to create another. COPY is an example of a command that can create a new result set from the records in an existing one. The result set from which records are being copied is referred to as the source result set. The one being copied to is the target result set.

The resource table records in a result set must all represent one type of managed object. That is, a result set that contains PROGRAM resource table records cannot also contain LOCTRAN resource table records. The resource table records must also be collected from the same CICSPlex SM context. So a result set that contains records from one CICSplex cannot be used to hold records from any other CICSplex. Once a result set is created, its resource type and context are fixed. The only way to change the type or context of a result set is to completely replace the contents of the result set with new resource table records.

Keep in mind that a resource table record in a result set is not the actual managed object; it is a report of the managed object's attributes at the point in time when data was collected. This is an important distinction because the actual managed object may have changed or may no longer exist by the time the resource table record is returned to your program. The number of records returned may vary as managed objects come and go, but the structure of the records in a result set remains constant.

A simple API program might deal with only one result set at a time. Each new request for data could create a result set that replaces the previous one. A more complex program might maintain several result sets concurrently and control the retention of those result sets more directly.

## An overview of result set commands

You can use the following commands to create result sets and manage the resources that they represent:

**GET** Returns a result set containing selected resource table records that represent instances of a managed resource.

**PERFORM**
Performs an action on one or more managed resources. PERFORM SET acts upon the resource table records in an existing result set. PERFORM OBJECT does not require a result set to exist; it creates one implicitly.

**REFRESH**
Refreshes the data for some or all of the managed resources as represented by resource table records in a result set.

**SET** Modifies the attributes of one or more managed resources as represented by resource table records in a result set.

These commands affect not only the resource table records in a result set, but also the managed resources that those records represent. Figure 4 on page 25

illustrates the relationship of these commands to the API environment.

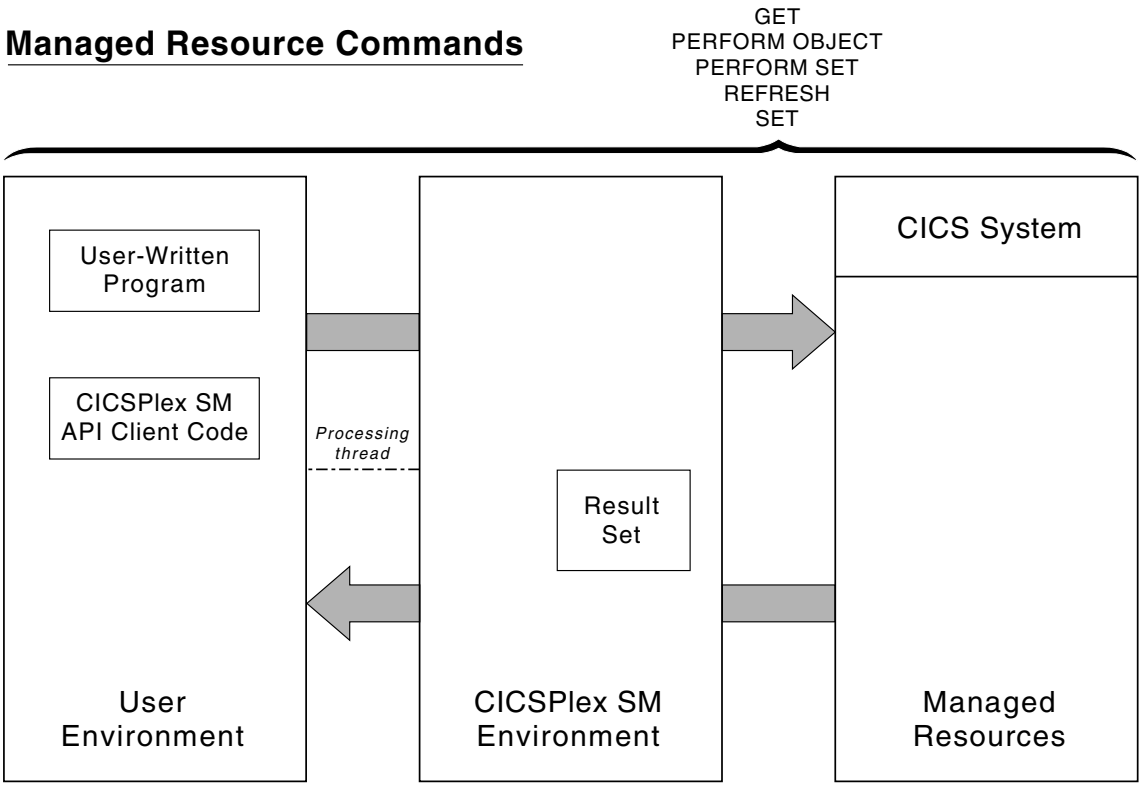



*Figure 4. API commands that manipulate managed resources*

Once a result set is created, you can perform various operations on the records it contains. You can sort, mark, copy, delete, and summarize the records in a result set. Most importantly, perhaps, you can retrieve records from a result set into local storage where they can be processed by your program.

You can use the following commands to manipulate one or more records in a result set:

**COPY** Copies some or all of the resource table records in one result set to another result set.

**DELETE**
Deletes one or more resource table records from a result set.

**FETCH**
Retrieves data and status information for one or more resource table records in a result set.

**GROUP**
Returns a summarized result set by grouping some or all of the resource table records in a result set.

**LOCATE**
Positions the record pointer within a result set.

**MARK** Marks selected resource table records in a result set.

**ORDER**
Sorts the resource table records in a result set.

**UNMARK**
Removes the marks placed on resource table records by a previous MARK command.

These commands affect only the current contents of a result set; they have no impact on the managed resources that the result set represents. Figure 5 illustrates the relationship of these commands to the API environment.
CICSPlex SM also provides tools for managing result sets as a whole: filters and

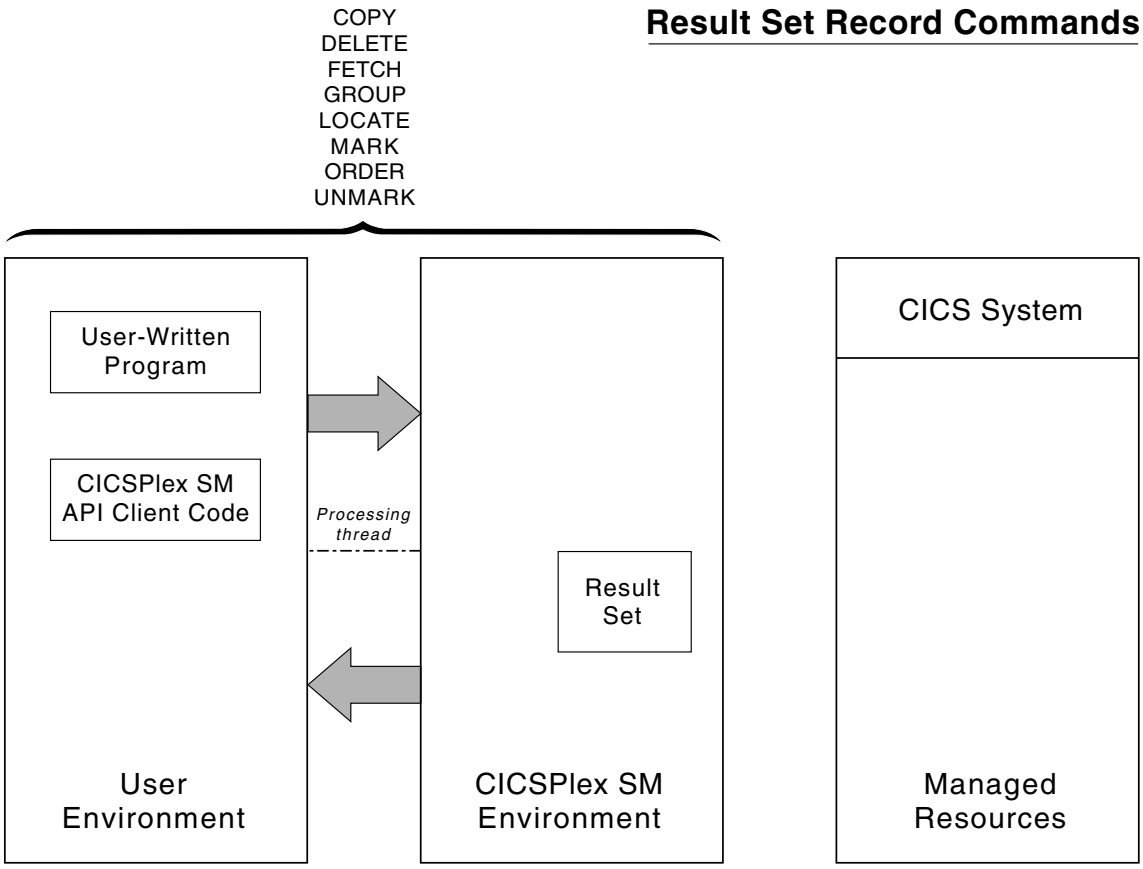



*Figure 5. API commands that manipulate result set records*

views for controlling the contents of a result set and commands for reviewing and discarding result sets.

You can use the following commands to manage result sets and their contents:

**DISCARD**
Discards a result set.

**QUERY**
Retrieves information about a result set and the resource table records it contains.

**SPECIFY FILTER**
Defines an attribute or value filter that can be used to control the contents of a result set.

**SPECIFY VIEW**
Builds a customized view of a resource table that can be used to control the contents of a result set

These commands affect only an existing or newly created result set; they have no impact on the managed resources that the result set represents. Figure 6 illustrates the relationship of these commands to the API environment.

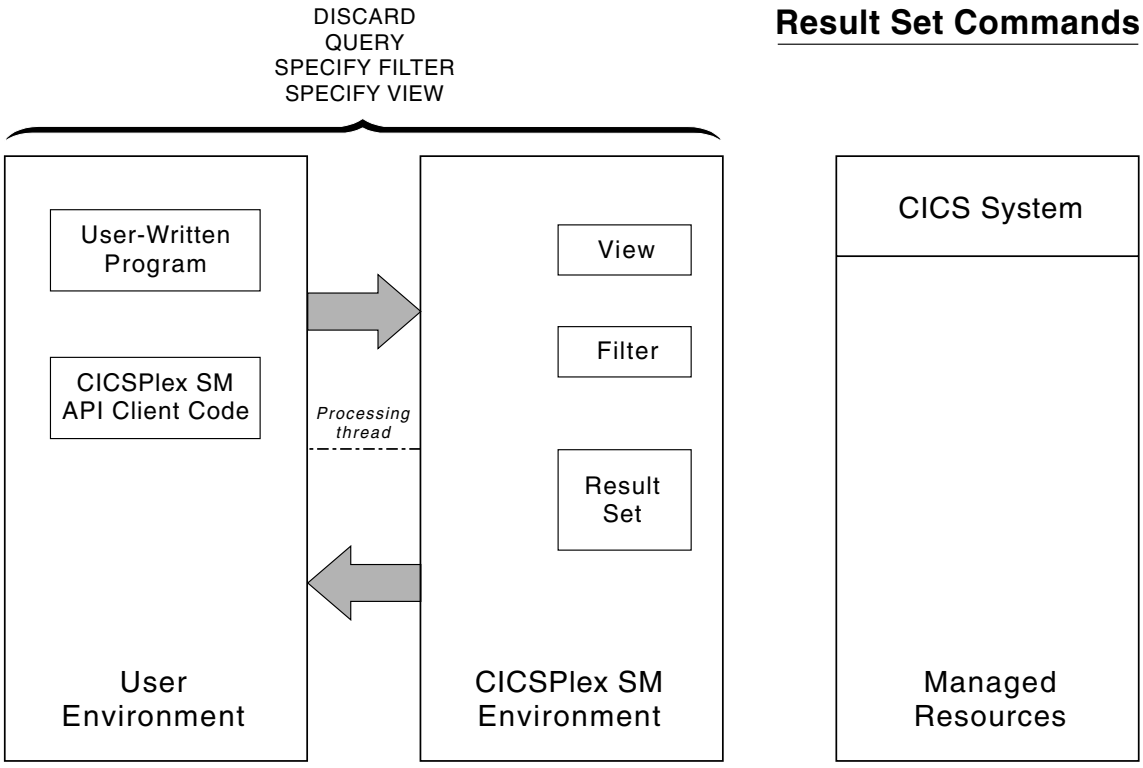



*Figure 6. API commands that manipulate result sets*

## Retrieving records from a result set

Once you have created a result set (using the GET command), you can transfer some or all of the records it contains to local storage for processing. You can use the FETCH command to retrieve a single resource table record, multiple selected records, or the entire result set at one time.

Each resource table record that you retrieve contains current data about the managed resource that it represents. Each record also contains certain status information that is maintained by CICSPlex SM.

This status information is presented as a resource table called OBJSTAT. The contents of the OBJSTAT resource table are described below:

### OBJSTAT

The OBJSTAT resource table provides status information for a specific record in a result set.

**Name Description**

**RECORDNUM**
The number of the record within the result set.

**CONTEXT**
The context in effect when the data for the record was collected.

**CICSNAME**
The name of the CICS system from which the data was collected.

**CICSREL**
The release level of the CICS system from which the data was collected.

**OBJECT**
The name of the managed object to which the data refers.

**OBJTYPE**
The data type of the managed object:

**1** CPSM resource

**2** Logical view

**RECTYPE**
The type of record data:

**1** Detail data

**2** Summary data

**LASTOPER**
The last operation performed against the object:

**1** COPY operation

**2** DELETE operation

**3** GET operation

**4** MARK operation

**5** REFRESH operation

**6** PERFORM OBJECT operation

**7** PERFORM SET operation

**8** SET operation

**9** UNMARK operation

**STATUS**
The current record status:

```
1... ....    X'80'      The record is MARKED
.... ...1    X'01'      Operation error
```

The attribute is not valid in the version of CICS if the bit is set on.

**CNTRECORDS**
The record count. For RECTYPE=1 the record count is zero. For RECTYPE=2 the record count will reflect the number of detail records.

**KEYLEN**
The length of the key data.

**KEYDATA**
The native key data

**RESERVE1**
Reserved area for future use.

In effect, each record in a result set contains a pair of resource tables: an instance of the OBJSTAT resource table followed by an instance of the resource table that

was requested. The managed resource data and the OBJSTAT status information can be retrieved either as a pair or separately, depending on the option you specify with the FETCH command:

**DATA** Retrieves only the specified resource table data.

**STATUS**
Retrieves only the OBJSTAT status information.

**BOTH** Retrieves both the resource table data and the OBJSTAT status information.

Figure 7 illustrates the information available in result set records and the FETCH commands you can use to retrieve that information.
The result set referenced by TOKENA was created by issuing a GET command for

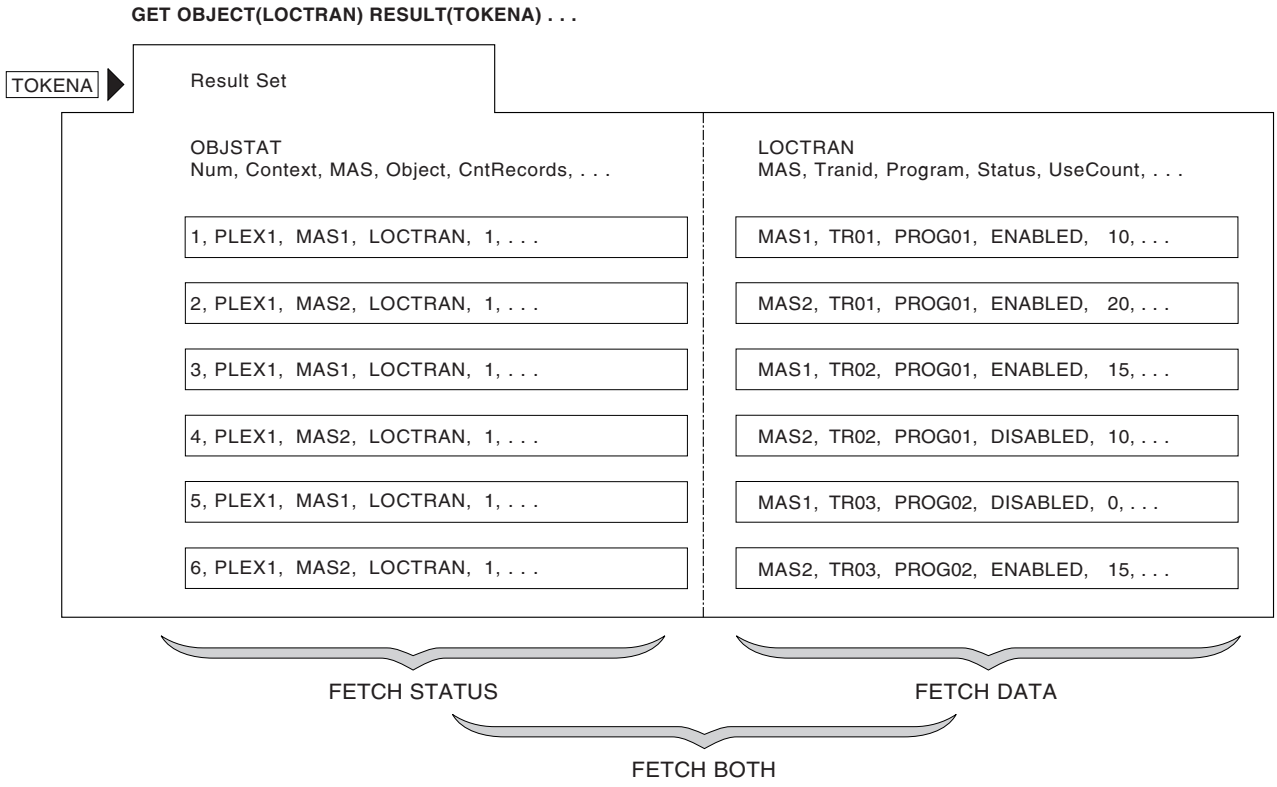



*Figure 7. Using FETCH to retrieve result set records*

LOCTRAN records. Each record in the result set consists of LOCTRAN data and OBJSTAT data.

You can use the FETCH commands shown in Figure 7 to selectively retrieve some or all of the data. For example, Figure 8 on page 30 shows the output of a FETCH DATA command.

**FETCH DATA ALL RESULT(TOKENA) INTO(AREA1) . . .**

| |
|---|
| MAS1, TR01, PROG01, ENABLED, 10, . . . |
| MAS2, TR01, PROG01, ENABLED, 20, . . . |
| MAS1, TR02, PROG01, ENABLED, 15, . . . |
| MAS2, TR02, PROG01, DISABLED, 10, . . . |
| MAS1, TR03, PROG02, DISABLED, 0, . . . |
| MAS2, TR03, PROG02, ENABLED, 15, . . . |

*Figure 8. Sample FETCH DATA output*

## Positioning the record pointer in a result set

CICSPlex SM maintains a current record pointer in each result set. When you first create a result set (with a GET command, for example), the pointer is positioned at the top of the result set. The first command that you issue against the result set affects the first record.

In most cases, when you issue FETCH commands to retrieve records from the result set, the record pointer is positioned to the next record in the result set (that is, the record following the last record that was fetched). However, certain API commands always act upon the last record that was fetched. When you issue any of these commands after a FETCH command, the record pointer is not advanced to the next record:

- COPY
- DELETE
- MARK
- UNMARK
- PERFORM SET CURRENT
- REFRESH CURRENT
- SET CURRENT

The record pointer in a result set may actually move either forward or backward, depending on the direction in which you are retrieving records. If you issue a FETCH command and no records are found that match the specified criteria, no records are retrieved. In that case, the pointer is positioned to the top or bottom of the result set, depending on the direction the pointer was moving.

If you issue a FETCH command and there is insufficient storage to retrieve all of the records, the pointer is positioned at the last record that would have been retrieved if there had been enough space. The pointer is not positioned at the last record that was actually retrieved. To be certain of the pointer's location, you should use the LOCATE command to explicitly position it within the result set.

The GET and FETCH commands leave the record pointer in specific, predefined positions, but other API commands do not. Many API commands manipulate records or update the data in a result set. The position of the record pointer after one of these commands depends on a combination of factors, including the options that you specified on the command. The pointer may have moved forward or backward one or more records, or it may be positioned to the top or bottom of the result set. If you specified the CURRENT option, the record pointer does not move; it remains positioned on the current record after the command is complete.

For this reason, CICSPlex SM provides the LOCATE command, which lets you explicitly position the record pointer within a result set. If you want to use the record pointer after issuing any of these commands, first use the LOCATE command to reposition it:

- COPY
- DELETE
- GETDEF
- GROUP
- MARK
- ORDER
- PERFORM OBJECT
- PERFORM SET
- REFRESH
- SET
- UNMARK.

## Processing selected records in a result set

If you want to process a subset of the resource table records in a result set, you can identify the records you are interested in by:

- Using the SPECIFY FILTER command to define a filter for selecting records, as described in “Using filter expressions” on page 20.
- Using the MARK and UNMARK commands to mark the records.

### Using MARK and UNMARK

The MARK command enables you to mark some or all of the resource table records in a result set for future reference. The UNMARK command removes existing marks from selected records. Once you have marked records in a result set, you can refer to the records that are either marked or not marked in subsequent commands. The following API commands support the MARKED and NOTMARKED options:

- COPY
- DELETE
- FETCH
- GROUP
- LOCATE
- PERFORM SET
- REFRESH
- SET

For example, Figure 9 on page 32 shows a result set in which selected resource table records have been marked. The MARKED option is then used with the FETCH command to retrieve only those records that are marked.

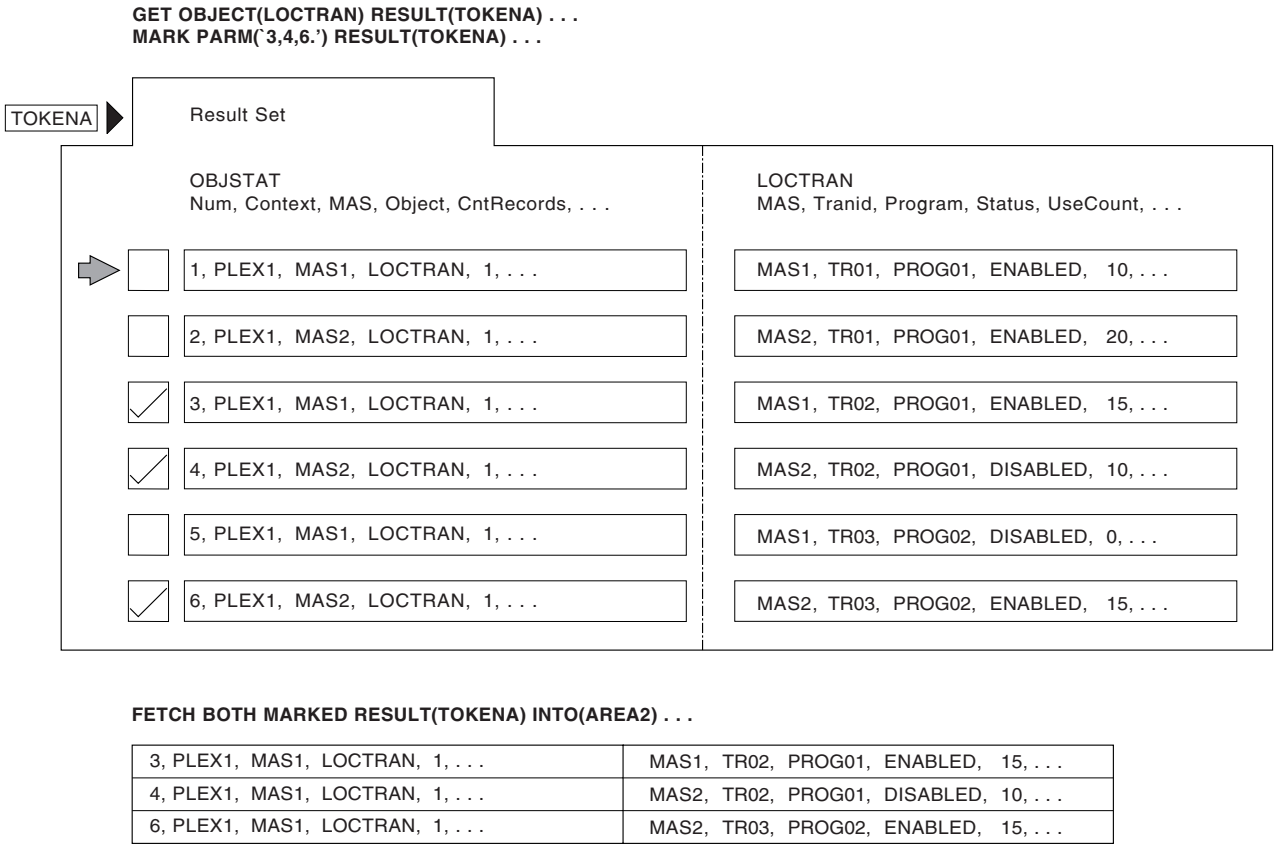



*Figure 9. Marking and retrieving records in a result set*

## Identifying the records to be marked

By default, when you issue a MARK or UNMARK command, only the current resource table record is marked or unmarked. But there are a variety of ways that you can identify the records to be marked:

- To mark a specific record other than the current record, use the POSITION option and identify the record by its relative position in the result set.
- To mark one or more records that meet previously defined filtering criteria, use the FILTER or NOTFILTER option.
- To mark all the records in a result set, use the ALL option.

In addition to these options, you can use the PARM option to identify a list of records to be marked. To use the PARM option, you specify a character string of record numbers in a parameter expression. The parameter expression can contain:

- Individual record numbers, separated by commas.
- Ranges of record numbers, with the low and high numbers separated by a colon.

The whole parameter expression must end with a period.

For example, to mark records 1, 3, 6 through 9, and 24 in a result set, you would specify:

```
PARM('1,3,6:9,24.')
```

When you use the PARM option, you must also use the PARMLEN option to specify the length of the buffer that contains the parameter expression.

**Notes:**

1. Negative values and 0 are not valid record numbers. If you specify an invalid record number, the MARK (or UNMARK) command returns RESPONSE and REASON values of INVALIDPARM PARM.
2. If you mistakenly specify the higher value in a range first (such as 9:6), CICSPlex SM reverses the values to produce a valid range.
3. If you mistakenly specify a single value preceded or followed by a colon (such as 6:), the colon is ignored. CICSPlex SM marks only the specified record.

## Identifying records that could not be marked

When you are marking or unmarking records, it might be useful to know if all the records you identified were successfully processed. For example, you might mistakenly ask CICSPlex SM to mark or unmark a record that was previously deleted from the result set. Or you might identify a record number that is out of range for the result set.

You can use the COUNT option on a MARK or UNMARK command to determine the number of records that could not be marked or unmarked. You can also use the INTO and LENGTH options to identify a buffer to receive a list of records that could not be marked. When deciding on the length of the INTO buffer, keep in mind that it must be long enough to hold the maximum number of record numbers that could result from your MARK request (in the event that none of them can be marked). Furthermore, all record numbers are listed individually (not by range) in the INTO buffer and are separated by commas. So if you specified the PARM option like this:

```
PARM('1,3:6,12,15.')
```

the INTO buffer would have to be long enough to hold the following character string:

```
1,3,4,5,6,12,15
```

If the INTO buffer you specify is not long enough to hold a complete list of records that could not be marked, you receive a RESPONSE value of WARNING AREATOOSMALL. In that case, the INTO buffer returns a partial list of records and the LENGTH value is set to the buffer length that would be required for a complete list. You could then resubmit the MARK command with the appropriate LENGTH value to determine which records could not be marked.

## How to remove the marks in a result set

You can use the UNMARK command to remove some or all of the marks placed on resource table records by a previous MARK command. However, if you want to mark other records at the same time, you can save a step by using the RESET option of the MARK command.

By default, the records you specify on a MARK command are marked in addition to any records that are already marked in the result set. That is, any resource table records that were marked previously remain marked unless you use the RESET option. RESET wipes the result set clean of any previous marks. So the records identified on the current MARK command are the only records marked when processing is complete. Using the RESET option on a MARK command is an alternative to using the UNMARK command before the MARK command.

**Note:** Any marks that you placed on resource table records are also removed when you use the COPY command to copy those records from one result set to another.

## Summarizing the records in a result set

If you want to analyze or modify a large number of records in a result set, you might find it useful to summarize those records. The GROUP command lets you summarize the records in a result set based upon the value of some resource table attribute.

When you issue a GROUP command, CICSPlex SM summarizes the records in one result set to create a new, summarized result set. A summarized result set is a special type of result set. It contains summary resource table records that correspond to one or more records in the source result set.

For example, you could use the GROUP command to summarize a result set that contains LOCTRAN resource table records. If you want to group the records according to the value of the STATUS attribute, the summarized result set would contain, at most, two records: one representing those records with a STATUS value of ENABLED, and one representing those with a STATUS of DISABLED. Figure 10 illustrates this use of the GROUP command.
In general, you can work with a summarized result set in the same ways that you

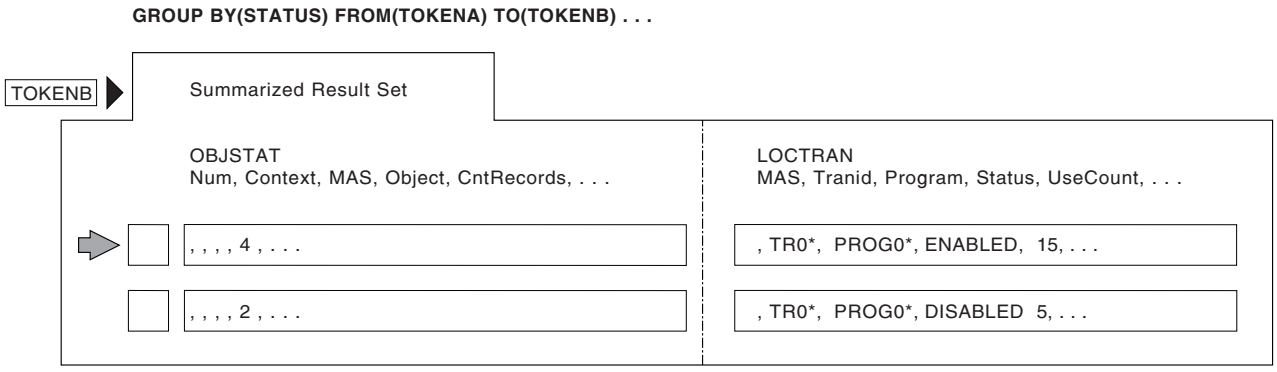



*Figure 10. Using GROUP to summarize result set records*

do a regular result set. You can use the FETCH command to retrieve records from a summarized result set. You can also retrieve the individual records of the source result set on which the summary is based. The DETAIL option of the FETCH command lets you retrieve that subset of records in the source result set that correspond to a particular summary record.

Figure 11 shows an example of fetching the detail records associated with a summary record. In this case, the summary record was a LOCTRAN record that represented all enabled transactions.

**FETCH DETAIL RESULT (TOKENB) INTO (AREA3) . . .**

| |
|---|
| MAS1, TR01, PROG01, ENABLED, 10, . . . |
| MAS2, TR01, PROG01, ENABLED, 20, . . . |
| MAS1, TR02, PROG01, ENABLED, 15, . . . |
| MAS2, TR03, PROG02, ENABLED, 15, . . . |

*Figure 11. Sample FETCH DETAIL output*

You can modify the records in a summarized result set using the PERFORM or SET commands. This is equivalent to modifying all the records in the source result set that are represented by a given summary record. However, since each record in a summarized result set has a single OBJSTAT record associated with it (rather than

one for each of the source records being modified), you may want to use the FETCH DETAIL command to determine the results of a summary action.

The OBJSTAT records in a source result set are not summarized when you issue a GROUP command. So the OBJSTAT records in a summarized result set do not represent the OBJSTAT information for all of the source records. However, the OBJSTAT records in a summarized result set do include a summary count, which indicates how many source records were combined to produce each summary record.

A summarized result set and its source result set should be thought of as a pair to be used together. They share certain attributes and the summarized result set has certain dependencies on the source result set:

- A summarized result set cannot exist without the source result set from which it was built. If you discard a source result set, all the summarized result sets that were built from it are also discarded.
- You can reuse a summarized result set only to resummarize the records in the same source result set. An existing summarized result set cannot be used as the target of a GROUP command for a different source result set.
- A summarized result set cannot be used as the source of a COPY command.
- If you modify a source or summarized result set in any way, all the summarized result sets that have been built from the source result set are rebuilt.

  **Note:** To prevent this from happening, you can specify the NOREFRESH option on the PERFORM or SET command.

## Specifying summary expressions

The attributes of a summary record are set according to a summary option that is appropriate for the attribute's data type. For each resource table attribute, CICSPlex SM defines a default summary option. CICSPlex SM uses these defaults when summarizing records unless you explicitly override them.

You tell CICSPlex SM how to summarize the attributes in a record by specifying a summary expression on the SUMOPT option of the GROUP command. A summary expression is a character string that consists of one or more summary options and the resource table attributes to which they apply.

The syntax of a summary expression is:

**Summary Expression**


where:

**sumopt**
Is the summary option to be used for the specified resource table attributes:

**AVG** Provides the average attribute value. Valid for numeric fields only.

**DIF** Provides those characters that are common to all underlying records and displays an asterisk (*) for those not common. Valid for character fields only.

**LIKE** Provides the CVDA or EYUDA value, if all records contain a common value. Otherwise, displays N/A. Valid for CVDA and EYUDA fields only.

**MAX** Provides the maximum attribute value.

**MIN** Provides the minimum attribute value.

**SUM** Provides the sum of the attribute values. Valid for numeric fields only.

You can specify the same summary option more than once in a summary expression.

**attr** Is the name of an attribute in the resource table.

You can specify as many attribute names for each summary option as you like.

**Note:** A summary expression must be followed by either blank spaces or null characters to the end of the specified buffer. That is, the buffer length you specify (using the LENGTH option) should not include any data other than a summary expression.

For example, you could use a summary expression like this when grouping LOCTRAN records:

```
SUM(USECOUNT) MAX(PRIORITY,TWASIZE).
```

By default, the values for these attributes would be averaged. But this summary expression specifies that each summary record should include the sum of all USECOUNT values and the maximum PRIORITY and TWASIZE values.

Table 7 shows the valid summary options for the various datatypes.

*Table 7. Valid summary options by attribute data type*

| | AVG | DIF | LIKE | MAX | MIN | SUM |
|---|---|---|---|---|---|---|
| ADDRESS | | | | X | X | |
| AVG | X | | | X | X | X |
| AVG3 | X | | | X | X | X |
| BIN | X | | | X | X | X |
| BIT | | | | X | X | |
| CHAR | | X | | X | X | |
| CODEBIN | X | | | X | X | X |
| COMPID | | | | X | X | |
| CVDAS | | | X | X | X | |
| CVDAT | | | X | X | X | |
| DATE | | | | X | X | |
| DATETIME | | | | X | X | |
| DEC | | | | X | X | |
| DECSTP | | | | X | X | |
| EYUDA | | | X | X | X | |
| HCHAR | | X | | X | X | |

*Table 7. Valid summary options by attribute data type (continued)*

| | | | | | | |
|---|---|---|---|---|---|---|
| HEX | | X | | X | X | |
| HHMM | | | | X | X | |
| INTVMSEC | X | | | X | X | X |
| INTVSEC | X | | | X | X | X |
| INTVSTCK | X | | | X | X | X |
| INTVUSEC | X | | | X | X | X |
| INTV16US | X | | | X | X | X |
| PCT | X | | | X | X | X |
| PCT3 | X | | | X | X | X |
| RATE | X | | | X | X | X |
| RATE3 | X | | | X | X | X |
| RATIO | X | | | X | X | X |
| RESTYPE | | | | X | X | |
| SCLOCK | X | | | X | X | X |
| SUM | X | | | X | X | X |
| SUM3 | X | | | X | X | X |
| TEXT | | X | | X | X | |
| TIMESTP | | | | X | X | |

## Sorting the records in a result set

The records in a result set are normally sorted by the key attributes for that resource table. In the case of CICS Resource and CICS Monitored tables, records are sorted by the CICS system from which they were collected. In working with result sets, you may find it easier to process the records if they are in some logical order of your own choosing. The ORDER command lets you sort the records in a result set according to the values of a particular resource table attribute.

You tell CICSPlex SM how to sort the records by specifying an order expression on the BY option of the ORDER command. An order expression is a character string that consists of one or more attribute names to be used in sorting the resource table records.

The syntax of an order expression for sorting records is:

**Order Expression – Sorting Records**


where:

**attr** Is the name of an attribute in the resource table.

You can specify as many attribute names as you like, but the total length of an order expression, including commas and blank spaces, must not exceed 255 characters.

**Note:** You cannot specify the EYU_CICSNAME or EYU_CICSREL attributes in an order expression.

**/D** Indicates the attribute values should be sorted in descending order. By default, the values are sorted in ascending order.

**Note:** An order expression must be followed by either blank spaces or null characters to the end of the specified buffer. That is, the buffer length you specify (using the LENGTH option) should not include any data other than an order expression.

For example, to sort a result set of LOCTRAN records by transaction ID and enabled status, you could specify:

```
TRANID,STATUS.
```

In this example, transaction ID is the primary sort key and enabled status is the secondary sort key.

To sort records in descending order of use count, add /D to the end of the attribute name, like this:

```
USECOUNT/D
```

## Modifying managed resources

This section describes various ways in which you can modify the resources managed by CICSPlex SM. The actions described here are issued against resource table records in a result set. However, the changes that you request are made to the actual resources which those records represent.

## Modifying resource attributes

You can change the current value of a resource attribute by using the SET or UPDATE command. SET modifies the attributes of a CICS resource, while UPDATE modifies CICSPlex SM and CICS definitions. The MODIFY option of these commands accepts a modification expression, which is a character string that defines the attribute changes to be made.

A modification expression can be made up of one or more attribute expressions in the form:

**Modification Expression**


where:

**attr** Is the name of a modifiable attribute in the resource table.

**value** Is the value to which you want the attribute set. The following restrictions apply:

- The value must be a valid one for the attribute.
- If the value contains imbedded blanks or special characters (such as periods, commas, or equal signs), the entire value string must be enclosed in single quotes, like this:

  ```
  DESCRIPTION='Payroll.OCT'
  ```

- To include a single quote or apostrophe in a value, you must repeat the character, like this:

```
DESCRIPTION='October''s Payroll'
```

**Note:** Be sure to consider the quoting conventions of your programming language when using single quotes in a CICSPlex SM value string.

**Note:** A modification expression must be followed by either blank spaces or null characters to the end of the specified buffer. That is, the buffer length you specify (using the LENGTH option) should not include any data other than a modification expression.

For example, to disable one or more local transactions (LOCTRAN), you could specify:

```
STATUS=DISABLED.
```

in the MODIFY option of a SET command.

If you issue a SET command against CICS systems that do not support the requested modification, the request is ignored for those CICS systems. If your context and scope consist solely of CICS systems that do not support the modification, you receive RESPONSE and REASON values of NOTAVAILABLE SCOPE.

To change the task storage location of a CICS transaction definition (TRANDEF), you could specify:

```
TASKDATALOC=ANY
```

in the MODIFY option of an UPDATE command.

Note that the MODIFY option of UPDATE is valid only for CICS Definition resource tables.

For a list of the attributes for each resource and their valid values, refer to the *CICSPlex System Manager Resource Tables Reference*.

## Performing an action against a resource

In addition to modifying individual attributes, you can also perform actions against many resources by using either of the PERFORM commands; PERFORMáOBJECT or PERFORMáSET. The difference between these two commands is that PERFORMáSET performs an action against the resource table records in an existing result set, while PERFORMáOBJECT first creates a result set and then performs the requested action.

Some actions are self-contained and self-explanatory; specifying the action is enough to indicate the changes to be made to the resource. For example, you can discard a local file by issuing the DISCARD action against a LOCFILE resource table record.

Some actions are self-contained and self-explanatory; specifying the action is enough to indicate the changes to be made to the resource. For example, you can discard a local file by issuing the DISCARD action against a LOCFILE resource table record.

Other actions require you to specify additional parameters. For these actions a parameter expression you might require a parameter expression to obtain the function you need. A parameter expression can be made up of one or more parm expressions in the form:

**Parameter Expression**




**parm_expr::**


where:

- `parm_name` is the name of a parameter associated with the action.
- `parm_value` is the value associated with the specified parameter name, if applicable.

Multiple instances of `parm_exp` should be delimited by spaces. The parameter expression buffer is terminated with a period (.).

For example:

- To disable a local file, you must indicate how to handle a file that is currently busy. To do that, you could specify the following parameter expression:

  ```
  PARM('BUSY(cvda).')
  ```

  where `cvda` is a valid CVDA value for the file busy condition.
- To request a dump of a CICS region (CICSRGN) specifying the dump code and title of the dump, you could use the following parameter expression:

  ```
  PARM('DUMPCODE(PMR12345) TITLE('Doc for PMR12345').')
  ```

  Note that if the parm_value contains special characters such as spaces or periods, the value must be enclosed in single quotes. Also note that all parameter values are folded to upper case.

The PERFORM OBJECT command does not require an existing result set, as it will effectively run a GET command followed be a PERFORM SET. In this case any parameter expression may be passed on the GET or PERFORM SET phase of the command depending on whether the parameter expression is valid on the GET or PERFORM SET as follows:

- If the parameter expression is valid for GET it is used on the GET phase of the PERFORM OBJECT command
- If the parameter expression is valid for PERFORM it is used on the PERFORM SET phase of the PERFORM OBJECT command.
- If the parameter expression is valid for GET and PERFORM it is used on the GET and PERFORM SET phases of the PERFORM OBJECT command. This rule may mean that some actions are not possible via PERFORM OBJECT. Instead separate GET and PERFORM SET commands may be required to achieve the desired results.
- If the parameter is not valid for GET or PERFORM, then an INVALIDPARM PARM condition will be raised.

# For a list of the valid actions for each resource and their required parameters, refer
# to the *CICSPlex System Manager Resource Tables Reference*.

## Working with CICSPlex SM and CICS definitions

When you work with CICSPlex SM and CICS definitions there are some special API commands and command options available.

### Creating, updating, and removing definitions

You can use the following API commands to maintain the CICSPlex SM and CICS definitions in your data repository:

**CREATE**
Creates a new CICSPlex SM or CICS definition using the attribute values you specify. The new definition is stored in the data repository.

**UPDATE**
Updates an existing CICSPlex SM or CICS definition according to the attribute values you specify. The updated definition replaces the existing definition in the data repository.

**REMOVE**
Removes a CICSPlex SM or CICS definition from the data repository.

**Notes:**

1. Before you can update or remove a definition you must use the FETCH command to retrieve the appropriate resource table record from a result set.
2. For CICSPlex SM definitions that have a CICSplex as their context (such as workload management or real-time analysis definitions), any changes you make are automatically distributed to all the CMASs involved in managing the CICSplex.

| With each of these commands, you use the FROM option to supply a CICSPlex SM
| Definition or CICS definition resource table record for the definition you are working
| with. The record must include all of the attributes in the resource table for the
| definition. If you do not want to specify certain optional attributes, you must set
| those fields to null (that is zero) values.

As an alternative, when you are updating CICS definitions, you can use the RESULT and MODIFY options of the UPDATE command. These options enable you to modify multiple definitions at one time (this is the equivalent of issuing the ALTER action command from the CICSPlex SM end-user interface).

To update CICS definitions, identify a result set that contains CICS Definition resource table records in the RESULT option. Then use the MODIFY option to specify the changes to be made to the definitions. MODIFY accepts a modification expression, as described in “Modifying resource attributes” on page 38.

### The CHANGETIME and CREATETIME attributes

When you work with existing CICSPlex SM or CICS definitions, keep in mind that the first 8 bytes of each record contain an attribute called CHANGETIME, which reflects the date and time at which the record was last modified. CICS Definition records also include a CREATETIME attribute, which is the date and time at which the definition was created.

The CHANGETIME and CREATETIME attributes are maintained internally by CICSPlex SM; you should not attempt to modify these attribute values. When you

update or remove a definition resource table record, the CHANGETIME and CREATETIME values you return to CICSPlex SM must be the same values you received.

## Using the PARM option

For most CICSPlex SM and CICS definitions, all of the information needed to process an API request is included in the attributes of the resource table. Some definitions, however, allow you to supply optional data and some actually require additional data. For those definitions, you have to specify the PARM option on the appropriate API command:

- CREATE
- UPDATE
- REMOVE
- GET

The PARM option accepts a parameter expression, which is a character string that defines the parameters required for a definition to be processed.

For example, suppose you want to create an LNKSMSCG definition, which is a CICSPlex SM definition that describes the association between a CICS system group and a monitor specification (MONSPEC). Before CICSPlex SM can process your request, it must know how to handle other links that may be affected by the change. So when you issue the CREATE command, you must specify a parameter expression like this on the PARM option:

```
PARM('FORCE.')
```

which tells CICSPlex SM that all CICS systems in the CICS system group are to inherit the new specification.

The PARM option is especially useful when working with CICS definitions. For each CICS Definition resource table there is another resource table that describes the definition's association with a resource group (RESGROUP), if one exists. For example, the CONNDEF resource table represents a connection definition and the CONINGRP resource table represents an association between a connection definition and a resource group. The RESGROUP parameter provided with the CREATE and GET commands for CICS Definitions simplifies the processing of these records.

When you create a CICS Definition record, you can identify an existing resource group to which the definition should be added. To do this, use the PARM option to identify the resource group like this:

```
PARM('RESGROUP(resgroup).')
```

Using the RESGROUP parameter automatically creates an xxxINGRP record (such as a CONINGRP record), which describes the association between the CICS definition and its resource group.

When you use the GET command to request CICS Definition records from the data repository, you can select definitions according to the resource group to which they belong. To do this, use the PARM option to identify the resource group like this:

```
PARM('RESGROUP(resgroup).')
```

which tells CICSPlex SM to select CICS definitions only from the specified resource group. If you do not use the PARM option, CICSPlex SM selects definitions from all resource groups, according to the other criteria you specify on the GET command.

**Note:** For a complete list of the CREATE, UPDATE, REMOVE, and GET parameters required (or supported) by a given resource table, see the *CICSPlex System Manager Resource Tables Reference*.

## Asynchronous processing

Most CICSPlex SM API commands normally function in a synchronous manner, where your program issues a request and then waits until command processing is complete. However, some of the API commands also support asynchronous processing. This allows you to request data or perform actions without waiting for the request to complete. This support also enables you to receive notification when events of interest occur in the CICSplex.

The API commands you can use to request asynchronous processing are:

- LISTEN
- CANCEL
- GET
- PERFORM OBJECT
- PERFORM SET
- REFRESH
- SET.

The LISTEN command is, by its very nature, an asynchronous request because you are asking to be notified whenever a certain event occurs. The CANCEL command simply cancels an outstanding LISTEN request. The other commands can be used in either a synchronous or asynchronous manner. If you specify the NOWAIT option on any of these commands, the request is processed asynchronously.

The API commands you can use to monitor and receive the results of asynchronous processing are:

- ADDRESS
- RECEIVE.

Figure 12 on page 44 illustrates the relationship of these commands to the API environment.

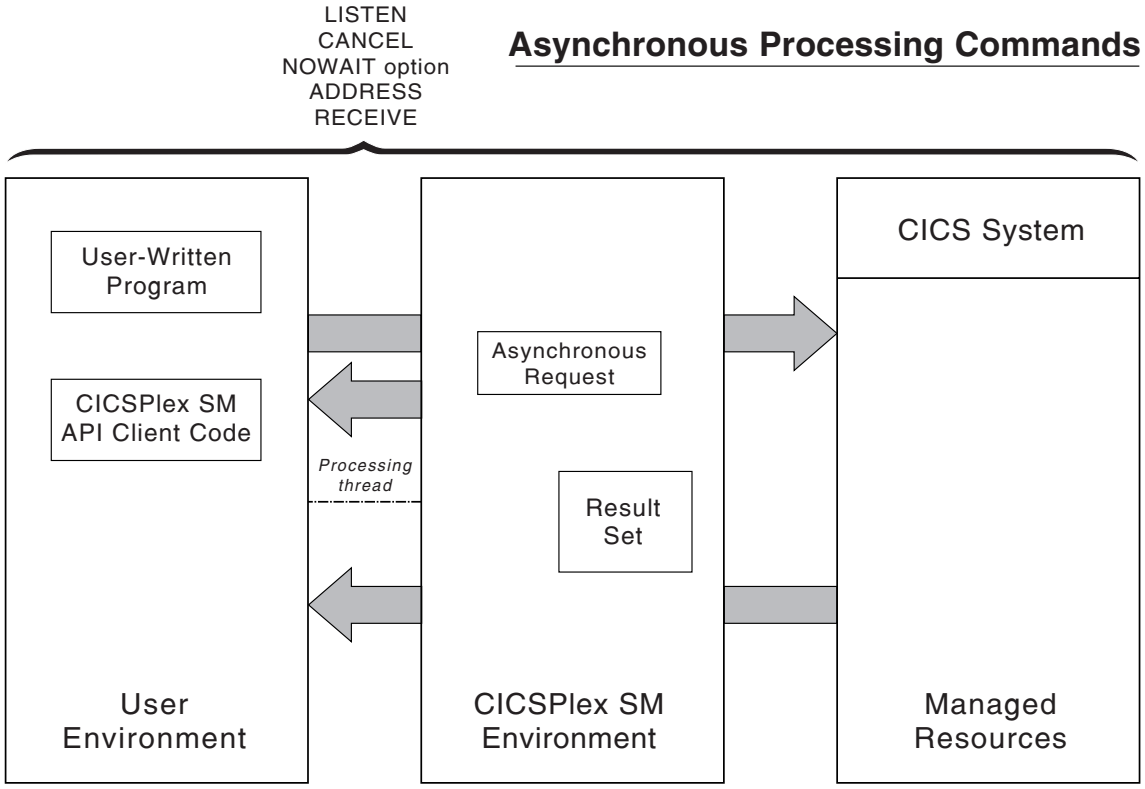



*Figure 12. API commands for asynchronous processing*

## Using the LISTEN command

Many of the resources that are managed by CICSPlex SM can notify the system when events occur that are considered significant to the CICSplex. Such events are not scheduled and cannot be anticipated, so a program designed to process these notifications must do so asynchronously. You can identify the event notifications you are interested in by using the LISTEN command.

The events that can be listened for are represented by resource tables with a type of CPSM Notification. For example, an EMASSICK notification is produced by a MAS when a condition occurs that adversely affects the health of the CICS system. For a list of the CPSM Notification resource tables, and complete descriptions of other resource table's, see *CICSPlex System Manager Resource Tables Reference*.

When you issue a LISTEN command, the resulting notifications are added to an outstanding data queue for the API processing thread. The number of completed asynchronous requests, including event notifications and requests issued with the NOWAIT option, is reported by the SENTINEL option of the ADDRESS command. You can retrieve the event notifications by issuing a RECEIVE command.

## Using the NOWAIT option

If you specify the NOWAIT option on a GET, PERFORM OBJECT, PERFORM SET, REFRESH, or SET command, the request does not complete processing immediately. Instead, the request is scheduled for processing, the command returns a RESPONSE value of SCHEDULED, and control returns to your program.

While the asynchronous request is executing, your program can perform other processing, even issuing another CICSPlex SM API command. However, as long as a command is active, the result set it has been given to process is unavailable. A RESPONSE value of INUSE is returned if you try to access a result set that is still being processed by an asynchronous request.

An ASYNCREQ resource table record is produced when the asynchronous request completes. The number of completed asynchronous requests, including ASYNCREQ records that represent requests issued with the NOWAIT option, is reported by the SENTINEL option of the ADDRESS command. You can retrieve ASYNCREQ records by issuing a RECEIVE command.

The ASYNCREQ resource table includes much of the information that is normally returned by the command itself. Because control returns to your program before the command completes processing, that information is not available to the command. The information returned in the ASYNCREQ resource table includes:

- The command that was issued.
- The associated result set token.
- The RESPONSE and REASON values returned by the command.
- The diagnostic data normally returned in a FEEDBACK resource table record, if the RESPONSE value is not OK.
- A user-defined token that identifies the asynchronous request, if one was specified.

**Note:** To access the ASYNCREQ data from a REXX program, you can use either the CICSPlex SM TPARSE command with the ASIS option or the REXX SUBSTR function.

## Using tokens to identify a request

To keep track of the asynchronous requests your program issues, you can assign each request a unique identifying token. This allows your program to correlate LISTEN requests and requests made with the NOWAIT option with the results of a subsequent RECEIVE command. The CICSPlex SM API makes no use of any tokens you define. User token values are simply held until the associated requests are complete and then returned to your program by the RECEIVE command. You can use any 1- to 4-character value as an identifying token. For example, you might specify:

- A literal constant
- An offset of a service routine
- The address of a data structure.

## Using the ADDRESS command

When you issue a CONNECT command and an API processing thread is established, two control fields are created in the MVS address space or CICS system where the program is running. By requesting the addresses of these thread control fields, you can determine if asynchronous output is available without the need for polling or suspending processing.

You can use the ADDRESS ECB() SENTINEL() command to request the addresses of these fields:

**ECB** The ECB is posted by the API each time an asynchronous request completes and is added to the thread's outstanding data queue. With the ECB address, you can:

- Test the appropriate MVS post bits to determine if output is available.
- Issue an MVS WAIT command in a batch, TSO, or NetView program.
- Issue an EXEC CICS WAITCICS or WAIT EXTERNAL command in a CICS program.

The ECB field is cleared whenever the counter value in the SENTINEL field reaches 0.

**SENTINEL**
The sentinel is a 4-byte counter of completed asynchronous requests associated with the thread.

The sentinel value increases each time an asynchronous request completes. Examples of completed asynchronous requests include:

- An event occurs that is named in a LISTEN command
- A command that was issued with the NOWAIT option completes processing.

The sentinel value decreases when a RECEIVE command is issued.

**Notes:**

1. You should use the ADDRESS command before issuing the RECEIVE command. If the sentinel value is 0, it means there are no completed asynchronous requests to be received.
2. Because of the nature of asynchronous processing, the sentinel value may understate the actual number of outstanding requests at any point in time. When processing multiple asynchronous requests, you should issue the RECEIVE IMMEDIATE command until a response of NODATA is returned to ensure that all output has been received.

## Using the RECEIVE command

You can use the RECEIVE command to determine if any of the asynchronous requests you issued have completed. RECEIVE returns the output from those requests. The output returned can be:

- A resource table record representing an event named in a previous LISTEN command
- An ASYNCREQ resource table record representing completion of an asynchronous GET, PERFORM, REFRESH, or SET request.

**Note:** Before you issue the RECEIVE command, you should issue the ADDRESS command and check the SENTINEL value to determine if there are any outstanding asynchronous requests to be received. If the sentinel value is 0, there are no outstanding asynchronous requests to be received.

As an example, your program might issue a LISTEN command and a GET command with the NOWAIT operand on the same API thread. Then, in response to a RECEIVE command, you would receive either an ASYNCREQ resource table record for the GET command or a resource table record associated with the event you were listening for.

Alternatively, you can use multiple API threads to separate the output returned by subsequent RECEIVE commands. For example, you might create one thread and use it only for receiving event notifications from the LISTEN command. You might

also create another thread for use by other API functions. In this way, you can control what output is returned by the RECEIVE commands issued against each thread.

Another reason you might want to create multiple API threads is because each thread can have only 256 asynchronous requests outstanding at one time. If your program issues a large number of asynchronous requests on a single API thread, you should issue the RECEIVE command at regular intervals. If a processing thread reaches its maximum of 256, asynchronous requests are discarded and are not processed.

By default, the RECEIVE command waits until asynchronous output is available before returning control to your program. This means processing is suspended until an asynchronous request completes. As an alternative to waiting indefinitely, you can specify one of these options on the RECEIVE command:

**DELAY(`data-value`)**
: Checks for asynchronous output, waits the specified number of seconds for output to become available, and then returns control to the processing thread, with or without output.

**IMMEDIATE**
: Checks for asynchronous output and then immediately returns control to the processing thread, whether or not any output is available.

## Using CICSPlex SM tokens

Many of the CICSPlex SM API commands are interrelated; you use them in conjunction with each other to accomplish the objectives of your program. For example, you issue a GET command to build a result set and then issue a FETCH command to access the resource table records in that result set.

To correlate the results of various operations with subsequent requests that you make, CICSPlex SM assigns 4-byte tokens to the following objects of the API environment:

- Processing threads
- Result sets
- Filters
- Views
- LISTEN requests.

So, for example, each processing thread has a unique, 4-byte identifying token. You must specify a thread token on each API command that your program issues to identify the thread where it should be processed. Likewise, once a result set or filter is created, you refer to it on subsequent commands by supplying the token value assigned to it by CICSPlex SM. And each LISTEN request is given a token so that you can cancel the request using the CANCEL command.

**Notes:**

1. CICSPlex SM assigns a token to views for internal use only. Externally, you refer to a view by the name which you assigned to it.
2. There is a limit to the number of CICSPlex SM tokens available to each processing thread. In general, the number of result sets, filters, views, and LISTEN requests created on a processing thread cannot exceed 255.

Token values are not only unique for individual objects, but the structure of the tokens varies by object type. So a thread token cannot be mistaken by CICSPlex SM for any other type of token. If you specify an invalid token (such as, a result set token on the FILTER option), you receive a RESPONSE value of INVALIDPARM.

## Using meta-data resource tables

The GETDEF command is used to obtain records describing the structure of the CICSPlex SM managed objects, including general characteristics, valid actions, and object attributes.

The OBJECT option of the GETDEF command identifies the type of meta-data to be retrieved. The contents of the following meta-data resource tables are described below:

- ATTR
- ATTRAVA
- METADESC
- METANAME
- METAPARM
- OBJACT
- OBJECT
- PARMAVA

## ATTR

The ATTR resource table provides detailed information for a specific attribute of a managed object.

**Attribute**
**Description**

**OBJECT**
The name of the managed object to which the specific attribute belongs.

**TABLEVER**
The version of the table identified by the OBJECT attribute.

**NAME** The name of the specific attribute. 1 to 12 characters in length.

**ID** The ID of the attribute

**LENGTH**
The length of the data associated with the attribute. Not to be confused with the length of the ATTR attribute NAME.

**OFFSET**
The offset in the resource table at which the attribute data begins.

**DATATYPE**
The data type of the attribute data:

**COMPID**
CICSPlex SM component ID

**BINARY**
Binary

**RATE** Rate to 1 decimal place

**PERCENT**
Percentage to 1 decimal place

**SUM** Sum of values to 1 decimal place

**RATIO**
Ratio

**AVERAGE**
Average to 1 decimal place

**TIMESTP**
Time stamp

**BIT** Bit string

**TEXT** Text

**CHAR** Character

**EYUDA**
EYUDA

**CVDAS**
Standard CVDA

**CVDAT**
Terminal CVDA

**RESTYPE**
Restype

**DECIMAL**
Packed Decimal

**DECDATE**
Date in decimal form

**ILABEL**
Internal Label

**HHMM**
Binary Hours/Minutes

**SCLOCK**
CICS CMF system clock

**INTUSEC**
Interval in microseconds

**INTMSEC**
Interval in milliseconds

**INT16US**
Interval in 16 microseconds

**INTSEC**
Interval in Seconds

**INTTSTP**
Interval Timestamp Delta

**DATETIME**
Date Time Group

**DECTSTP**
Decimal Timestamp

**ADDRESS**
Address

**CNUMERIC**
Coded Numeric

**HIDCHAR**
Non Display Character

**HEX** Hexadecimal

**TBLVER**
Resource table version

**RATE3**
Rate to 3 decimal places

**PERCENT3**
Percentage to 3 decimal places

**SUM3** Sum of values to 3 decimal places

**AVERAGE3**
Average to 3 decimal places

**DECTIME**
Time in units of tenths of a second

**DECTIMES**
Time in units of seconds

**SUMOPT**
The default summary option used for the attribute:

**AVG** Average

**DIFF** Difference

**MIN** Minimum

**MAX** Maximum

**SUM** Summary

**LIKE** Like

**IDATATYPE**
A numeric value which represents the internal data type

**0** Component

**4** Numeric

**8** Rate

**12** Percent

**16** Sum

**20** Ratio

**24** Average

**28** Timestamp

**32** Bit

**36** Text

**40** Character

**44** EYUDA

**48** CVDA standard

**52** CVDA terminal

**56** Resource type

**60** Packed decimal

**64** Packed decimal date

**68** Internal label field

**72** HHMM

**76** Interval store clock

**80** Interval microseconds

**84** Interval milliseconds

**88** Interval 16 microseconds

**92** Interval seconds

**96** Store clock delta

**100** Date time group

**104** Packed decimal timestamp to tenths of seconds

**108** Address

**112** Codes numeric

**116** Non-display character

**120** Hexadecimal

**124** Table version

**128** Binary derived rate to 3 decimal places

**132** Binary derived percent to 3 decimal places

**136** Binary derived sum to 3 decimal places

**140** Binary derived average to three deecimal places

**144** Packed decimal time to seconds

**148** Packed decimal time to tenths of seconds

**SETVALID**
Whether or not the attribute may be set/modified: Y or N

**REQUIRED**
Whether or not the attribute is required for CREATE: Y or N

**AVAAVAIL**
Whether or not attribute value assertion information is available for the attribute: Y or N. When available, refer to the ATTR attributes:
AVACOUNT

Use the ATTRAVA resource table to obtain attribute value assertion information.

**CICSVALAVAIL**
Whether or not CICS validity data is available: Y or N. When available, refer to the ATTR attributes:
VALCICSESA
VALCICSVSE
VALCICSOS2

VALCICSWNT
VALCICSES2

**HDRTXTAVAIL**
Whether or not attribute header text is available: Y or N. When available, refer to the ATTR attributes:
HDRTEXT

**VALSETAVAIL**
Whether or not value set information is available: Y or N. When available, refer to the ATTR attributes:
VALCOUNT

Use the ATTRAVA resource table to obtain value set information.

**SOURCE**
The source of the attribute data:

**V** Created by CPSM

**I** Acquired from CICS INQ

**S** Acquired from CICS STATS

**P** Acquired from CICS CMF data

**KEY** Whether or not the attribute participates in the key of the managed object: 0 or n, where 0 means the attribute is not part of the key, and n means the part number of the key.

**AVACOUNT**
The number of attribute value assertions for the attribute. This value corresponds to the number of ATTRAVA resource table records available with a LISTTYPE value of AVA for the attribute. Only present if the AVAAVAIL attribute is Y.

**VALCOUNT**
The number of value set values for the attribute. This value corresponds to the number of ATTRAVA resource table records available with a LISTTYPE value of VALUE for the attribute. Only present if the VALSETAVAIL attribute is Y.

**VALCICSESA**
First byte of flags indicating whether or not the attribute is valid in different versions of CICS/ESA:

```
1... ....    X'80'      CICS/MVS 2.1.2
.1.. ....    X'40'      CICS/ESA 3.3.0
..1. ....    X'20'      CICS/ESA 4.1.0
...1 ....    X'10'      CICS Transaction Server for
                        OS/390 Release 1
.... 1...    X'08'      CICS Transaction Server for
                        OS/390 Release 2
.... .1..    X'04'      CICS Transaction Server for
                        OS/390 Release 3
.... ..1.    X'02'      CICS Transaction Server for
                        z/OS Version 2 Release 1
.... ...1    X'01'      CICS Transaction Server for
                        z/OS Version 2 Release 2
```

The attribute is not valid in the version of CICS if the bit is set on. VALCICSES2 contains a second byte of flags.

**VALCICSVSE**

Flags indicating whether or not the attribute is valid in different versions of CICS/VSE:

```
1... ....    X'80'      CICS/VSE 2.2.0
.1.. ....    X'40'      CICS/VSE 2.3.0
..1. ....    X'20'      CICS/VSE 4.1.0
...1 1111               Reserved
```

The attribute is not valid in the version of CICS if the bit is set on.

**VALCICSOS2**

Flags indicating whether or not the attribute is valid in different versions of CICS OS/2:

```
1... ....    X'80'      CICS OS/2 2.0.1
.1.. ....    X'40'      CICS OS/2 3.0.0
..1. ....    X'20'      CICS OS/2 3.1.0
...1 1111               Reserved
```

The attribute is not valid in the version of CICS if the bit is set on.

**VALCICSWNT**

Flags indicating whether or not the attribute is valid in different versions of TXSeries:

```
1... ....    X'80'      CICS for TXSeries 4.3.0
.1.. ....    X'40'      CICS for TXSeries 5.0.0
..11 1111               Reserved
```

The attribute is not valid in the version of CICS if the bit is set on.

**VALCICSES2**

Second byte of flags indicating whether or not the action is valid in different versions of CICS/ESA:

```
1... ....    X'80'      CICS Transaction Server for z/OS, Version 2 Release 3
.1.. ....    X'40'      CICS Transaction Server for z/OS, Version 3 Release 1
..11 1111               Reserved
```

The action is not valid in the version of CICS if the bit is set on. The first byte of flags is contained in VALCICSESA.

**SETCICSESA**

First byte of flags indicating whether or not the attribute is modifiable in different versions of CICS/ESA:

```
1... ....    X'80'      CICS/MVS 2.1.2
.1.. ....    X'40'      CICS/ESA 3.1.0
..1. ....    X'20'      CICS/ESA 4.1.0
...1 ....    X'10'      CICS Transaction Server for
                        OS/390 Release 1
.... 1...    X'08'      CICS Transaction Server for
                        OS/390 Release 2
.... .1..    X'04'      CICS Transaction Server for
                        OS/390 Release 3
.... ..1.    X'02'      CICS Transaction Server for
                        z/OS Version 2 Release 1
.... ...1    X'01'      CICS Transaction Server for
                        z/OS Version 2 Release 2
```

The attribute is not modifiable in the version of CICS if the bit is set on. A second byte of flags is contained in SETCICSES2.

**SETCICSVSE**
Flags indicating whether or not the attribute is modifiable in different versions of CICS/VSE:

```
1... ....    X'80'      CICS/VSE 2.2.0
.1.. ....    X'40'      CICS/VSE 2.3.0
..1. ....    X'20'      CICS/VSE 4.1.0
...1 1111               Reserved
```

The attribute is not modifiable in the version of CICS if the bit is set on.

**SETCICSOS2**
Flags indicating whether or not the attribute is modifiable in different versions of CICS OS/2:

```
1... ....    X'80'      CICS OS/2 2.0.1
.1.. ....    X'40'      CICS OS/2 3.0.0
..1. ....    X'20'      CICS OS/2 3.1.0
...1 1111               Reserved
```

The attribute is not modifiable in the version of CICS if the bit is set on.

**SETCICSWNT**
Flags indicating whether or not the attribute is modifiable in different versions of TXSeries:

```
1... ....    X'80'      CICS for TXSeries 4.3.0
.1.. ....    X'40'      CICS for TXSeries 5.0.0
..11 1111               Reserved
```

The attribute is not modifiable in the version of CICS if the bit is set on.

**SETCICSES2**
Second byte of flags indicating whether or not the attribute is modifiable in different versions of CICS/ESA:

```
1... ....    X'80'      CICS Transaction Server for z/OS, Version 2 Release 3
.1.. ....    X'40'      CICS Transaction Server for z/OS, Version 3 Release 1
..11 1111               Reserved
```

The attribute is not modifiable in the version of CICS if the bit is set on

**IGNVALUE**
The value that signifies Not Applicable or Ignore for the attribute.

**LOWVALUE**
The lowest value allowed in the range of valid values for the attribute.

**HIGHVALUE**
The highest value allowed in the range of valid values for the attribute.

**HDRTEXT**
The header text of the attribute. Only present if the HDRTXTAVAIL attribute value is Y.

**DESC** The description of the attribute.

**DEFAULT**
The default value for the attribute, if any.

**UCHAR**
Whether or not the attribute value is uppercase: Y or N.

**CICSSETAVAIL**
Indicates whether or not the SET command is valid for an attribute: Y or **N**.

When set to **Y** the following to the ATTR attributes indicate the levels of different CICS products for which the command is valid:

- SETCICSESA
- SETCICSVSE
- SETCICSOS2
- SETCICSWNT
- SETCICSES2

**SORT** Indicates whether or not the attribute participates in ORDER

**Y** The attribute participates in ORDER

**N** The attribute does not participate in ORDER

**FILTER**
Indicates whether or not the attribute participates in SPECIFY FILTER

**Y** The attribute participates in SPECIFY FILTER

**N** The attribute does not participate in SPECIFY FILTER

**SUMMARISE**
Eligibility of the attribute for summarizing

**Y** The attribute may be summarized

**N** The attribute may not be summarized

**VIEWMOD**
Eligibility of the attribute for view support

**Y** The attribute is eligible for view support

**N** The attribute is not eligible for view support

**INHERIT**
Indicates whether or not the attribute participates in inheritance

**Y** The attribute participates in inheritance

**N** The attribute does not participate in inheritance

## ATTRAVA

The ATTRAVA resource table provides an acceptable value for a specific attribute of a managed object. The set of ATTRAVA base tables for a specific attribute provide the list of all acceptable values.

However, please note that the attribute may support a range of values (for example, zero to 999) and there are no ATTRAVA base tables for the range values. There also may not be an ATTRAVA base table for the default value for the attribute. The default value, and the highest and lowest in range values can be found from the ATTR base table for the attribute.

Information in this resource table is only available when the AVAAVAIL or VALSETAVAIL attributes of the ATTR resource table have a value of Y.

**Attribute**
**Description**

**OBJECT**
The name of the managed object to which the specific attribute belongs.

**TABLEVER**
The version of the table identified by the OBJECT attribute.

**NAME** The name of the specific attribute. 1 to 12 characters in length.

**AVAVALUE**
A value for the attribute.

**LISTTYPE**
Indicates if the AVAVALUE data is an attribute value assertion or other acceptable value for the attribute:

**AVA** A value derived from an attribute value assertion

**VALUE**
At present, this is only used to return the special value meaning "ignore"

**IOTYPE**
Indicates whether the attribute value is used for input, output, or input and output operations:

**I** Input

**O** Output

**B** Input and output

# METADESC

The METADESC resource table provides basic structure and layout information for a specific attribute of a managed object.

**Attribute**
**Description**

**NAME** The name of the specific attribute. 1 to 12 characters in length.

**LENGTH**
The length of the data associated with the attribute. Not to be confused with the length of the METADESC attribute NAME.

**OFFSET**
The offset in the resource table at which the attribute data begins.

**DATATYPE**
The data type of the attribute data:

**0** Component Identifier

**4** Binary Numeric

**8** Binary Derived Rate

**12** Binary Derived Percent

**16** Binary Derived Sum

**20** Binary Derived Ratio

**24** Binary Derived Average

**28** System/370 Timestamp

**32** Bit

**36** Text

**40** Character

**44** EYUDA

**48** CVDA Standard

**52** CVDA Terminal

**56** Resource Type

**60** Packed Decimal

**64** Packed decimal date

**68** Internal Label Field

**72** Binary HHMM

**76** Interval Store Clock

**80** Interval Microseconds

**84** Interval Milliseconds

**88** Interval 16 Microseconds

**92** Interval Seconds

**96** Interval Store Clock delta

**100** Date Time Group

**104** Packed Decimal Timestamp to tenths of seconds

**108** Address

**112** Coded Numeric

**116** Non Display Character

**120** Hexadecimal

**124** Table version

**128** Binary derived rate to 3 decimal places

**132** Binary derived percent to 3 decimal places

**136** Binary derived sum to 3 decimal places

**140** Binary derived average to three decimal places

**144** Packed decimal timestamp to seconds

**148** Packed decimal timestamp to tenths of seconds

**INHERIT**
Whether or not the attribute value is inheritable: Y or N. Valid only for CPSM Definition resource tables that participate in CICSplex inheritance.

## METANAME

The METANAME resource table provides information about all CVDASs, CVDATs, and EYUDAs.

**Attribute**
**Description**

**NAMETYPE**
Type of data

**1** CVDAS

**2** CVDAT

**3** EYUDA

**VALUE**
Numeric value of CVDA or EYUDA

**NAME** Name of CVDA or EYUDA

**DESCRIPTION**
Description of CVDA or EYUDA

## METAPARM

The METAPARM resource table provides information about a parameter for an action.

**Attribute**
**Description**

**TABLE**
Table name

**ACTION**
Action name

**NAME** Parameter name as it appears in the API PARM string

**ID** Parameter number

**GROUP_ID**
Multiple parameters may be related to each other in the sense that only one of a group may be specified. Parameters that are related in this way will have the same group ID.

**REQUIRED**
Indicates whether or not the parameter is required

**Y** The parameter is required

**N** The parameter is not required

**WORKLOAD**
Indicates whether or not the parameter is a workload name

**Y** The parameter is a workload name

**N** The parameter is not a workload name

**WRKLOWNER**
Indicates whether or not the parameter is the name of a workload owner

**Y** The parameter is the name of a workload owner

**N** The parameter is not the name of a workload owner

**VALUE**
Parameter value

**MODE** Method by which parameter is applied

**1** Copy from base table

**2** Array of values

**3** Bit setting

**4** Keyword in API parameter string:

**5** Filter string

**6** API keyword with value

**7** Base table field with existence bit

**8** API modification string

Modes 3 and 4 appear in the API parameter string as stand-alone keywords. Modes 2, 5, 6 and 8 appear in the API parameter string as keywords with a value. Modes 1 and 7 do not appear in the API parameter string.

**DESCRIPTION**
Description

**CICSVALAVAIL**
Indicates whether or not CICS validity data is available

**Y** CICS validity data is available

**N** CICS validity data is not available

**VALCICSESA**
First byte of flags indicating whether or not the parameter is valid in different versions of CICS/ESA:

```
1... ....    X'80'      CICS/MVS 2.1.2
.1.. ....    X'40'      CICS/ESA 3.3.0
..1. ....    X'20'      CICS/ESA 4.1.0
...1 ....    X'10'      CICS Transaction Server for
                        OS/390 Release 1
.... 1...    X'08'      CICS Transaction server for
                        OS/390 Release 2
.... .1..    X'04'      CICS Transaction Server for
                        OS/390 Release 3
.... ..1.    X'02'      CICS Transaction Server for
                        z/OS Version 2 Release 1
.... ...1    X'01'      CICS Transaction Server for z/OS, Version 2 Release 2
```

The parameter is not valid in the version of CICS if the bit is set on. The second byte of flags is contained in VALCICSES2.

**VALCICSVSE**
Flags indicating whether or not the parameter is valid in different versions of CICS/VSE:

```
1... ....    X'80'      CICS/VSE 2.2.0
.1.. ....    X'40'      CICS/VSE 2.3.0
..1. ....    X'20'      CICS/VSE 4.1.0
...1 1111               Reserved
```

The parameter is not valid in the version of CICS if the bit is set on.

**VALCICSOS2**
Flags indicating whether or not the parameter is valid in different versions of CICS OS/2:

```
1... ....    X'80'      CICS OS/2 2.0.1
.1.. ....    X'40'      CICS OS/2 3.0.0
..1. ....    X'20'      CICS OS/2 3.1.0
...1 1111               Reserved
```

The parameter is not valid in the version of CICS if the bit is set on.

**VALCICSWNT**
Flags indicating whether or not the parameter is valid in different versions of TXSeries:

```
1... ....    X'80'      CICS for TXSeries 4.3.0
.1.. ....    X'40'      CICS for TXSeries 5.0.0
..11 1111               Reserved
```

The parameter is not valid in the version of CICS if the bit is set on.

**VALCICSES2**
Second byte of flags indicating whether or not the parameter is valid in different versions of CICS/ESA:

```
1... ....    X'80'      CICS Transaction Server for z/OS, Version 2 Release 3
.1.. ....    X'40'      CICS Transaction Server for z/OS, Version 3 Release 1
..11 1111               Reserved
```

The parameter is not valid in the version of CICS if the bit is set on. The first byte of flags is contained in VALCICSESA.

# OBJACT

The OBJACT resource table provides action information for a specific managed object.

**Attribute**
**Description**

**OBJECT**
The name of the managed object to which the specific action applies.

**TABLEVER**
The version of the table identified by the OBJECT attribute.

**ACTION**
The name of the action. 1 to 12 characters in length.

**VALCICSESA**
First byte of flags indicating whether or not the action is valid in different versions of CICS/ESA:

```
1... ....    X'80'      CICS/MVS 2.1.2
.1.. ....    X'40'      CICS/ESA 3.3.0
..1. ....    X'20'      CICS/ESA 4.1.0
...1 ....    X'10'      CICS Transaction Server for
                        OS/390 Release 1
.... 1...    X'08'      CICS Transaction server for
                        OS/390 Release 2
.... .1..    X'04'      CICS Transaction Server for
                        OS/390 Release 3
.... ..1.    X'02'      CICS Transaction Server for
                        z/OS Version 2 Release 1
.... ...1    X'01'      CICS Transaction Server for z/OS, Version 2 Release 2
```

The action is not valid in the version of CICS if the bit is set on. The second byte of flags is contained in VALCICSES2.

**VALCICSVSE**
Flags indicating whether or not the action is valid in different versions of CICS/VSE:

```
1... ....    X'80'      CICS/VSE 2.2.0
.1.. ....    X'40'      CICS/VSE 2.3.0
..1. ....    X'20'      CICS/VSE 4.1.0
...1 1111               Reserved
```

The action is not valid in the version of CICS if the bit is set on.

**VALCICSOS2**
Flags indicating whether or not the action is valid in different versions of CICS OS/2:

```
1... ....    X'80'      CICS OS/2 2.0.1
.1.. ....    X'40'      CICS OS/2 3.0.0
..1. ....    X'20'      CICS OS/2 3.1.0
...1 1111               Reserved
```

The action is not valid in the version of CICS if the bit is set on.

**VALCICSWNT**
Flags indicating whether or not the action is valid in different versions of TXSeries:

```
1... ....    X'80'      CICS for TXSeries 4.3.0
.1.. ....    X'40'      CICS for TXSeries 5.0.0
..11 1111               Reserved
```

The action is not valid in the version of CICS if the bit is set on.

**VALCICSES2**
Second byte of flags indicating whether or not the action is valid in different versions of CICS/ESA:

```
1... ....    X'80'      CICS Transaction Server for z/OS, Version 2 Release 3
.1.. ....    X'40'      CICS Transaction Server for z/OS, Version 3 Release 1
..11 1111               Reserved
```

The action is not valid in the version of CICS if the bit is set on.The first byte of flags is contained in VALCICSESA.

**DESCRIPTION**
The description of the action

**ID** The number of the action

**PARMCOUNT**
The number of parameters for this action

**APIPERFORM**
Indicates whether or not an action is valid for EXEC CPSM PERFORM, GET, SET, CREATE, UPDATE, and REMOVE.

**N** The action is not valid.

**Y** The action is valid.

## OBJECT

The OBJECT resource table provides detailed information for a specific managed object.

**Attribute**
**Description**

**NAME** The name of the managed object. 1 to 8 characters in length.

**ID** The numeric resource table ID.

**NUMTBLVER**
The number of different versions of the managed object which are known to exist.

**HIGHTBLVER**
The number of the highest version of the managed object.

**RELTBLVER**
The version of the managed object at the current CPSM release.

**OWNERNAME**
The name of the component which owns the managed object.

**CREATREL**
CPSM release at which the managed object was introduced.

**QUERYREL**
CPSM release of the querying CMAS.

**OBJTYPE**
The object type of the managed object:

**C** CICS Resource

**M** Monitored CICS Resource

**D** CPSM Definition

**V** CPSM Resource

**O** CPSM Metadata

**N** CPSM Notification

**R** CICS Resource Definition

**L** CPSM Configuration Definition

**CURTBLVER**
Version of the managed object at the current CONNECT version

**CURNUMATTR**
Number of attributes in the managed object at the current CONNECT version

**CURSTGSIZE**
External length of the managed object at the current CONNECT version

**CURCPSMREL**
CPSM release when the version of the managed object at the current CONNECT version was created

**CURVALRTA**
Whether or not the managed object is valid for use with RTA: Y or N.

**CURVALUTL**
Whether or not the managed object is valid for use with the batch utility: Y or N.

**CURGETVAL**
Whether or not the managed object is valid for GET requests: Y or N.

**CURSETVAL**
Whether or not the managed object is valid for SET requests: Y or N.

**CURCREVAL**
Whether or not the managed object is valid for CREATE requests: Y or N.

**CURUPDVAL**
Whether or not the managed object is valid for UPDATE requests: Y or N.

**CURREMVAL**
Whether or not the managed object is valid for REMOVE requests: Y or N.

**CURACTVAL**
Whether or not the managed object has actions defined: Y or N.

Use the OBJACT resource table to obtain action information.

**CURVALESA**
First byte of flags indicating whether or not the managed object is valid in different versions of CICS/ESA:

```
1... ....    X'80'      CICS/MVS 2.1.2
.1.. ....    X'40'      CICS/ESA 3.3.0
..1. ....    X'20'      CICS/ESA 4.1.0
...1 ....    X'10'      CICS Transaction Server for
                        OS/390 Release 1
.... 1...    X'08'      CICS Transaction Server for
                        OS/390 Release 2
.... .1..    X'04'      CICS Transaction Server for
                        OS/390 Release 3
.... ..1.    X'02'      CICS Transaction Server for
                        z/OS Version 2 Release 1
.... ...1    X'01'      CICS Transaction Server for
                        z/OS Version 2 Release 2
```

The object is not valid in the version of CICS if the bit is set on.The second byte of flags is contained in CURVALES2.

**CURVALVSE**
Flags indicating whether or not the managed object is valid in different versions of CICS/VSE:

```
1... ....    X'80'      CICS/VSE 2.2.0
.1.. ....    X'40'      CICS/VSE 2.3.0
..1. ....    X'20'      CICS/VSE 4.1.0
...1 1111               Reserved
```

The object is not valid in the version of CICS if the bit is set on.

**CURVALOS2**
Flags indicating whether or not the managed object is valid in different versions of CICS OS/2:

```
1... ....    X'80'      CICS OS/2 2.0.1
.1.. ....    X'40'      CICS OS/2 3.0.0
..1. ....    X'20'      CICS OS/2 3.1.0
...1 1111               Reserved
```

The object is not valid in the version of CICS if the bit is set on.

**CURVALWNT**
Flags indicating whether or not the managed object is valid in different versions of TXSeries:

```
1... ....    X'80'      CICS for TXSeries 4.3.0
.1.. ....    X'40'      CICS for TXSeries 5.0.0
..11 1111               Reserved
```

The object is not valid in the version of CICS if the bit is set on.

**CURVALES2**
Second byte of flags indicating whether or not the managed object is valid in different versions of CICS/ESA:

```
1... ....    X'80'      CICS Transaction Server for z/OS, Version 2 Release 3
.1.. ....    X'40'      CICS Transaction Server for z/OS, Version 3 Release 1
..11 1111               Reserved
```

The action is not valid in the version of CICS if the bit is set on. The first byte of flags is contained in CURVALESA.

**DESC** The description of the managed object.

**VIEWMOD**
Eligibility of the managed object for view support

**Y** The managed object is eligible for view support

**N** The managed object is not eligible for view support

**APIPREFIX**
Indicates whether or not an API prefix is required

**Y** An API prefix is required

**N** An API prefix is not required

**SCOPESORT**

**Y** The API sorts by scope

**N** The API does not sort by scope

**SCOPEREQ**

**Y** Scope must be specified

**N** Scope need not be specified

## PARMAVA

The PARMAVA resource table provides information about the values that may be specified for a parameter.

**Attribute**
**Description**

**PARMIDN**
Parameter number

**PARMAVAIDN**
AVA number for parameter

**LITERAL**
Parameter literal

**VALUE**
Parameter value in numeric form

**VALUENAME**
Parameter value as a character string

# Using CRESxxxx resource tables

The CRESxxxx base tables are externalized versions of the topology resource maps, and are usually updated when a resource is installed, added, discarded or removed. This information is captured via the CICS XRSINDI global user exit.

In addition, a small number of CRESxxxx base tables are also updated when the characteristics of an existing resource is modified. Those CRESxxxx base tables that are updated regularly at MAS heartbeat time are:

- CRESDSNM data set
- CRESFECO FEPI connection
- CRESGLUE global user exit

- CRESSDMP system dump code
- CRESTDMP transaction dump code
- CRESTRUE task-related user exit

An application program that needs to be informed when any of the CRESxxxx topology resource maps is changed can use the API LISTEN command to register an interest in the corresponding ERMCxxxx CPSM notification resource table.

## Querying the CICSPlex SM API exit

In a CICS LMAS environment the CICSPlex SM API function is implemented via a task related user exit. CICS application programs can use the EXEC CICS INQUIRE EXITPROGRAM command to retrieve information about the CICSPlex SM API task related user exit:

```
EXEC CICS INQUIRE EXITPROGRAM(EYU9XLAP)
                  CONNECTST(cvda)
                  QUALIFIER(data-area)
```

In CICS systems that support the CONNECTST and QUALIFIER keywords of the INQUIRE EXITPROGRAM command, CONNECTST returns a CVDA indicating the status of the CICSPlex SM API task related user exit (see the *CICS System Programming Reference* for more details on INQUIRE EXITPROGRAM), and QUALIFIER returns the name of the CICSplex to which the LMAS is connected.

# Chapter 3. Writing an EXEC CPSM program

This chapter describes how to use the CICSPlex SM command-level interface to write an API program. It describes the language-specific copy books supplied for each CICSPlex SM resource table. It also describes the translation process and the compile, link-edit, and run-time considerations for each environment.

## Using the resource table copy books

The CICSPlex SM API accepts and returns resource data in the form of records that contain the resource attributes. For example, if you issue a FETCH command against a result set containing LOCTRAN resource table records, the API returns all the attributes for a given transaction in a single record. Your program must identify an area of storage to receive the resource table records.

**Note:** This method of returning data differs from the EXEC CICS system programming interface, where you must fetch each attribute of a resource individually.

To simplify the use of these resource table records, CICSPlex SM provides a set of copy books for each resource table that you can access from an API program. By including these copy books in your program, you can access the resource table data in the appropriate structure and format for the language you are using.

## How to access the copy books

The copy books are installed as part of the CICSPlex SM installation process. They are placed into the following libraries:

Assembler
: CICSTS31.CPSM.SEYUMAC

COBOL
: CICSTS31.CPSM.SEYUCOB

PL1
: CICSTS31.CPSM.SEYUPL1

C
: CICSTS31.CPSM.SEYUC370.

If you want to include the copy books in your program, make sure the appropriate library is available to the assemble or compile step.

**Note:** The CICSPlex SM API uses variable names that begin with EYU. Make sure your program does not define variables or structures with variable names that are the same as variable names generated by the translator or declared in the resource table copy books. Also be careful that your program does not implicitly generate such variable names.

## Copybook names and aliases

Each CICSPlex SM resource table has a name that is unique within the product. In addition, a unique name is created for each copy book version of the resource table in each language. The copy book names take the form:

```
EYUtnnnn
```

where:

**t**
: Identifies which language the copy book supports, as one of the following:

  **A** Assembler

**P** PL/I

**L** COBOL

**C** C

**nnnn** Is a 4-character numeric resource table identifier.

For example:

**EYUA0001**
Is the Assembler DSECT for the CICSRGN resource table.

**EYUC2451**
Is the C structured data type for the CMAS resource table.

To make the copy books easy to reference in your program, CICSPlex SM provides alias support for the copy book names. The appropriate data set contains the following two entries for each resource table:

**EYUtnnnn**
The resource table copy book name.

**formname**
The format name alias, which is the resource table name as shown in *CICSPlex System Manager Resource Tables Reference*.

So, using the previous example, the Assembler DSECT for the CICSRGN resource table could be referred to as either EYUA0001 or its alias, CICSRGN.

## Copybook format

Each copy book contains a prologue that describes the resource table and its characteristics, including:

- Valid API operations
- Any parameters that are required for an operation
- Valid API actions
- CICS releases that do not support the resource table, if any.

A description is provided for each attribute of the resource table. In addition, the following information is provided for an attribute, if appropriate:

- Whether the attribute can be modified by a SET command
- CICS releases that do not support the attribute, if any
- CICS releases that do not allow the attribute to be modified, if any.

## Copybook data characteristics

Each resource table that can be processed by an API program contains data values for each of its attributes. The attribute values are presented in an internal format that is appropriate for the data type and the environment in which the program is running:

- Standard System/390 data formats are used. No translation or formatting operations are performed on the attribute values.
- For programs written in C, variable-length character fields do not contain the zero-byte ending delimiter.
- The lengths of all resource table records are a multiple of 8 bytes. Each copy book contains a definition of the resource table length.

- System/390 boundary alignments are observed for all data types. That means all resource table records are maintained internally starting on doubleword-aligned storage locations. Alignment fields are automatically generated in each copy book. These alignment fields, which contain binary zeros, have names like:

  ```
  EYU_RSVnnnn
  ```

  Make sure the data areas your program uses to send and receive resource table records have proper boundary alignment.

# Supplied copy books

This section provides detailed information about the resource table copy books supplied for each language.

## Assembler copy books

**Distributed in:**
CICSTS31.CPSM.SEYUMAC

**Distributed as:**
DSECTs

**Copybook names:**
EYUAnnnn

Note the following as you use the Assembler copy books:

- DSECT and DS statements are used to describe the resource table.
- The DSECT name is the resource table format name (such as, EMASSTRT).
- The attribute names are a concatenation of the resource table format name and the attribute name, connected by an underscore (such as, EMASSTRT_CMASNAME).
- EQU statements are used to describe the setting of indicator fields for bit, binary, and character values.
- The table length field is a concatenation of the resource table format name and the constant TBL_LEN, connected by an underscore (such as, EMASSTRT_TBL_LEN).

The resource table data types are defined using the data definition operands of the DS statement. The following data type definitions are used:

```
DS   X  - Bit, binary values greater than 8 bytes
        - Odd number binary values less than 8 bytes
        - Mixed character and binary data

DS   H  - 2-byte binary numeric values

DS   F  - 4-byte binary numeric values
        - 4-byte intervals

DS   D  - Time stamps and 8-byte intervals
        - 8-byte numeric values

DS   P  - Packed decimal data

DS   C  - Character data
```

Figure 13 is a representative extract of an Assembler resource table copy book:

```
*-------------------------------------------------------------------*
*   Name = EYUA2400                                                 *
*   Format Name = EMASSTRT                                          *
*   Version = 0001                                                  *
*   Status = CPSMREL(0310)                                          *
*   Function = Base Table Structure generator                       *
*   Format definition for this element = EMASSTRT                   *
*   Valid Operations = None                                         *
*   Valid Actions = None                                            *
*-------------------------------------------------------------------*
EMASSTRT               DSECT       Notify CICS System Start Event
EMASSTRT_CMASNAME      DS  CL0008  CMAS Name
EMASSTRT_PLEXNAME      DS  CL0008  CICSPlex Name
EMASSTRT_CSYSNAME      DS  CL0008  CICS System Name
EMASSTRT_MON_SPEC      DS  CL0008  Monitor Spec Name
EMASSTRT_RTA_SPEC      DS  CL0008  Real Time Analysis Spec Name
EMASSTRT_WLM_SPEC      DS  CL0008  Work Load Manager Spec Name
EMASSTRT_STATUS        DS  XL0001  Status
EMASSTRT_STATUS_LOCAL          EQU 128      Local MAS
EMASSTRT_STATUS_REMOTE         EQU 64       Remote MAS
EMASSTRT_DYNROUTE      DS  XL0001  Dynamic Routing Mode
EMASSTRT_DYNROUTE_ACTIVE       EQU 1        Routing ACTIVE
EMASSTRT_DYNROUTE_SUSPEND      EQU 2        Routing SUSPENDED
EMASSTRT_DYNTYPE       DS  CL0003  Dynamic Routing Type
EMASSTRT_DYNTYPE_WLMTOR        EQU C'TOR'   Routing TOR
EMASSTRT_DYNTYPE_WLMAOR        EQU C'AOR'   Routing AOR
EMASSTRT_DESC          DS  CL0030  Description
EMASSTRT_CSYSAPPL      DS  CL0008  CICS System VTAM APPLID
EMASSTRT_EYU_RSV0015   DS  XL0005  Alignment Padding
EMASSTRT_MASSTART      DS  D       MAS Start STCK Value
EMASSTRT_TMEZONEO      DS  XL0001  Time Zone Offset
EMASSTRT_TMEZONE       DS  CL0001  Time Zone
EMASSTRT_EYU_RSV0019   DS  XL0002  Alignment Padding
EMASSTRT_DAYLGHTSV     DS  F       DayLight savings in effect
EMASSTRT_SYSID         DS  CL0004  MAS System Id
EMASSTRT_OPSYSREL      DS  CL0004  MAS Op Sys Release
EMASSTRT_MVSNAME       DS  CL0004  MVS System Name
EMASSTRT_JOBNAME       DS  CL0008  MAS Job Name
EMASSTRT_CECNAME       DS  CL0008  CEC Name
EMASSTRT_SYSPLEX       DS  CL0008  SYSPlex Name
EMASSTRT_EYU_RSV0257   DS  XL0004  Alignment Padding
EMASSTRT_TBL_LEN       EQU 152     Current Table size
```

*Figure 13. Sample Assembler copy book*

## PL/I copy books

**Distributed in:**
CICSTS31.CPSM.SEYUPL1

**Distributed as:**
Based structures

**Copybook names:**
EYUPnnnn

Note the following as you use the PL/I copy books:

- The variable EYUPTPTR must be explicitly declared as follows:

  ```
  DCL EYUPTPTR POINTER;
  ```

- The structure level 1 name is the resource table format name (such as, EMASSTRT).
- The attribute names are used as subordinate level names.

- For attributes that describe bit indicators, subordinate structure levels are used. Each bit indicator is assigned a unique name.
- All other indicator attributes result in constant declarations being generated at the end of the resource table. These constants can be used for assignment or evaluation of the attribute. The constant name is a concatenation of the resource table name, the attribute name, and the indicator name, connected by underscores (such as, EMASSTRT_DYNROUTE_ACTIVE).
- The table length field is a concatenation of the resource table format name and the constant TBL_LEN, connected by an underscore (such as, EMASSTRT_TBL_LEN).

The resource table data types are mapped into the valid set of PL/I data types. However, exact mapping is not always possible. The resource table data types are mapped as follows:

```
     BIT(8) ALIGNED  - 1-byte binary numeric values

      FIXED BIN(15)  - 2-byte binary numeric values

      FIXED BIN(31)  - 4-byte binary numeric values
                     - 4-byte intervals

  (2) FIXED BIN(31)  - Time stamps and 8-byte intervals
                     - 8-byte binary numeric values
                       (an array of two fullwords)

       FIXED DEC(n)  - Packed decimal data

         CHAR(nnnn)  - Character data
                     - Binary values greater than 8 bytes
                     - Odd number binary values less than 8 bytes
```

Figure 14 is a representative extract of a PL/I resource table copy book:

```
/*----------------------------------------------------------------*/
/* Name = EYUP2400                                                */
/* Format Name = EMASSTRT                                         */
/* Version = 0001                                                 */
/* Status = CPSMREL(0310)                                         */
/* Function = Base Table Structure generator                      */
/* Format definition for this element = EMASSTRT                  */
/* Valid Operations = None                                        */
/* Valid Actions = None                                           */
/*----------------------------------------------------------------*/
```

*Figure 14. Sample PL/I copy book (Part 1 of 5)*

```
DCL 01 EMASSTRT BASED(EYUPTPTR), /* Notify CICS System Start Event*/
       02 CMASNAME        CHAR(0008),
          /* CMAS Name                                           */
       02 PLEXNAME        CHAR(0008),
          /* CICSPlex Name                                       */
       02 CSYSNAME        CHAR(0008),
          /* CICS System Name                                    */
       02 MON_SPEC        CHAR(0008),
          /* Monitor Spec Name                                   */
       02 RTA_SPEC        CHAR(0008),
          /* Real Time Analysis Spec Name                        */
       02 WLM_SPEC        CHAR(0008),
          /* Work Load Manager Spec Name                         */
```

*Figure 14. Sample PL/I copy book (Part 2 of 5)*

```
02 STATUS,
   /* Status                                          */
   03 LOCAL        BIT(1) UNALIGNED,
   /* Local MAS                                       */
   03 REMOTE       BIT(1) UNALIGNED,
   /* Remote MAS                                      */
   03 RSVD0003     BIT(1) UNALIGNED,
   /* Reserved                                        */
   03 RSVD0004     BIT(1) UNALIGNED,
   /* Reserved                                        */
   03 RSVD0005     BIT(1) UNALIGNED,
   /* Reserved                                        */
   03 RSVD0006     BIT(1) UNALIGNED,
   /* Reserved                                        */
   03 RSVD0007     BIT(1) UNALIGNED,
   /* Reserved                                        */
   03 RSVD0008     BIT(1) UNALIGNED,
   /* Reserved                                        */
```

*Figure 14. Sample PL/I copy book (Part 3 of 5)*

```
02 DYNROUTE        BIT(8) ALIGNED,
   /* Dynamic Routing Mode                            */
02 DYNTYPE         CHAR(0003),
   /* Dynamic Routing Type                            */
02 DESC            CHAR(0030),
   /* Description                                     */
02 CSYSAPPL        CHAR(0008),
   /* CICS System VTAM APPLID                         */
02 EYU_RSV0015     CHAR(0005),
   /* Alignment Padding                               */
02 MASSTART(2)     FIXED BIN(31),
   /* MAS Start STCK Value                            */
02 TMEZONEO        BIT(8) ALIGNED,
   /* Time Zone Offset                                */
02 TMEZONE         CHAR(0001),
   /* Time Zone                                       */
02 EYU_RSV0019     CHAR(0002),
   /* Alignment Padding                               */
02 DAYLGHTSV       FIXED BIN(31),
   /* DayLight savings in effect                      */
02 SYSID           CHAR(0004),
   /* MAS System Id                                   */
02 OPSYSREL        CHAR(0004),
   /* MAS Op Sys Release                              */
02 MVSNAME         CHAR(0004),
   /* MVS System Name                                 */
02 JOBNAME         CHAR(0008),
   /* MAS Job Name                                    */
02 CECNAME         CHAR(0008),
   /* CEC Name                                        */
02 SYSPLEX         CHAR(0008),
   /* SYSPlex Name                                    */
02 EYU_RSV0257     CHAR(0004);
   /* Alignment Padding                               */
```

*Figure 14. Sample PL/I copy book (Part 4 of 5)*

```
/*----------------------------------------------------------------*/
/*                                                                */
/* EMASSTRT Constants for Table                                   */
/*                                                                */
/*----------------------------------------------------------------*/
DCL EMASSTRT_DYNROUTE_ACTIVE BIT(8) ALIGNED STATIC INIT('01'BX);
          /* Routing ACTIVE                                       */
DCL EMASSTRT_DYNROUTE_SUSPEND BIT(8) ALIGNED STATIC INIT('02'BX);
          /* Routing SUSPENDED                                    */
DCL EMASSTRT_DYNTYPE_WLMTOR CHAR(3) STATIC INIT('TOR');
          /* Routing TOR                                          */
DCL EMASSTRT_DYNTYPE_WLMAOR CHAR(3) STATIC INIT('AOR');
          /* Routing AOR                                          */
DCL EMASSTRT_TBL_LEN FIXED BIN(15) STATIC INIT(152);
```

*Figure 14. Sample PL/I copy book (Part 5 of 5)*

## COBOL copy books

**Distributed in:**
CICSTS31.CPSM.SEYUCOB

**Distributed as:**
Structures

**Copybook names:**
EYULnnnn

Note the following as you use the COBOL copy books:

- The structure level 1 name is the resource table format name (such as, EMASSTRT).
- The attribute names are used as subordinate level names.
- For attributes that describe indicators, subordinate 88 levels are used. Each indicator is assigned a unique name. Hexadecimal literals are used to describe the content of the indicator setting.
- By default, CICSPlex SM attribute names are formed with a connecting underscore character, as in WLM_SPEC. However, since COBOL syntax does not support underscores, all attribute names that contain underscores are converted in the copy books to use hyphens, as in WLM-SPEC. When attribute names are passed to the API, they must contain the underscore character, not the hyphen used by COBOL.
- All the resource tables use apostrophe characters as literal delimiters. When you translate or compile your program with a supplied copy book, you must specify the APOST option. Otherwise, you will receive COBOL warning messages.
- COBOL reserves many words for its own use. Some of the CICSPlex SM resource table and attribute names conflict with these reserved words. To prevent such a conflict, any CICSPlex SM name that conflicts with a COBOL reserved word is modified by adding a suffix of -R. For example, the name of the CONNECT resource table becomes CONNECT-R and the name of the STATUS attribute becomes STATUS-R. The comment area for a name that would conflict with COBOL shows the description “-- RESERVED WORD --”. When resource table or attribute names are passed to the API, they must not include the -R suffix.
- COBOL does not support duplicate names at different levels in the same data structure. Some of the CICSPlex SM attribute names are the same as resource table names. To prevent a duplicate name problem, any attribute name that is the same as a resource table name is modified by adding a suffix of -A. For

example, the name of the DSNAME attribute becomes DSNAME-A. The name of the DSNAME resource table remains unchanged. The comment area for an attribute that has the same name as a resource table shows the description "-- RESERVED WORD --". When attribute names are passed to the API, they must not include the -A suffix.

- The table length field is a concatenation of the resource table format name and the constant TBL-LEN, connected by a hyphen (such as, EMASSTRT-TBL-LEN).

The resource table data types are mapped into the valid set of COBOL data types. However, exact mapping is not always possible. The resource table data types are mapped as follows:

```
         PIC S9(0004) USAGE BINARY  - 2-byte binary numeric values

         PIC S9(0008) USAGE BINARY  - 4-byte binary numeric values
                                    - 4-byte intervals

         PIC S9(0016) USAGE BINARY  - Time stamps and 8-byte intervals
                                    - 8-byte binary numeric values

  PIC S9(nnnn) USAGE PACKED-DECIMAL  - Packed decimal data

                       PIC X(0001)  - 1-byte binary and bit indicators

                       PIC X(nnnn)  - Character data
                                    - Binary values greater than 8 bytes
                                    - Odd number binary values less than
                                      8 bytes
```

Figure 15 is a representative extract of a COBOL resource table copy book:

```
* ------------------------------------------------------------*
*   Name = EYUL2400                                           *
*   Format Name = EMASSTRT                                    *
*   Version = 0001                                            *
*   Status = CPSMREL(0310)                                    *
*   Function = Base Table Structure generator                 *
*   Format definition for this element = EMASSTRT             *
*   Valid Operations = None                                   *
*   Valid Actions = None                                      *
* ------------------------------------------------------------*
 01 EMASSTRT.
* Notify CICS System Start Event
   02 CMASNAME       PIC X(0008).
* CMAS Name
   02 PLEXNAME       PIC X(0008).
* CICSPlex Name
   02 CSYSNAME       PIC X(0008).
* CICS System Name
   02 MON-SPEC       PIC X(0008).
* Monitor Spec Name
   02 RTA-SPEC       PIC X(0008).
* Real Time Analysis Spec Name
   02 WLM-SPEC       PIC X(0008).
* Work Load Manager Spec Name
   02 STATUS-R       PIC X(0001).
* Status                            -- RESERVED WORD --
     88 LOCAL        VALUE X'80'.
* Local MAS
     88 REMOTE       VALUE X'40'.
* Remote MAS
   02 DYNROUTE       PIC X(0001).
* Dynamic Routing Mode
     88 ACTIVE       VALUE X'01'.
* Routing ACTIVE
     88 SUSPEND      VALUE X'02'.
* Routing SUSPENDED
   02 DYNTYPE        PIC X(0003).
* Dynamic Routing Type
     88 WLMTOR       VALUE 'TOR'.
* Routing TOR
     88 WLMAOR       VALUE 'AOR'.
* Routing AOR
   02 DESC           PIC X(0030).
* Description
   02 CSYSAPPL       PIC X(0008).
* CICS System VTAM APPLID
   02 EYU-RSV0015    PIC X(0005).
```

*Figure 15. Sample COBOL copy book (Part 1 of 2)*

```
* Alignment Padding
   02 MASSTART       PIC S9(0016) USAGE BINARY.
* MAS Start STCK Value
   02 TMEZONEO       PIC X(0001).
* Time Zone Offset
   02 TMEZONE        PIC X(0001).
* Time Zone
   02 EYU-RSV0019    PIC X(0002).
* Alignment Padding
   02 DAYLGHTSV      PIC S9(0008) USAGE BINARY.
* DayLight savings in effect
   02 SYSID          PIC X(0004).
* MAS System Id
   02 OPSYSREL       PIC X(0004).
* MAS Op Sys Release
   02 MVSNAME        PIC X(0004).
* MVS System Name
   02 JOBNAME        PIC X(0008).
* MAS Job Name
   02 CECNAME        PIC X(0008).
* CEC Name
   02 SYSPLEX        PIC X(0008).
* SYSPlex Name
     02 EYU-RSV0257  PIC X(0004).
* Alignment Padding
* ------------------------------------------------------------*
*                                                             *
*   EMASSTRT Constants for Table                              *
*                                                             *
* ------------------------------------------------------------*
 01 EMASSTRT-TBL-LEN PIC S9(4) USAGE BINARY VALUE 152.
```

*Figure 15. Sample COBOL copy book (Part 2 of 2)*

## C copy books

**Distributed in:**
CICSTS31.CPSM.SEYUC370

**Distributed as:**
Structured data types

**Copybook names:**
EYUCnnnn

Note the following as you use the C copy books:

- Typedef statements are used to describe the resource table.
- The structure name is the resource table format name (such as, EMASSTRT).
- The attribute names are used as subordinate names.
- For attributes that describe bit indicators, #define statements are generated at the end of the resource table Each #define statement identifies a single indicator value. These constants can be used for assignment or evaluation of the attribute. The constant name is a concatenation of the resource table name, the attribute name, and the indicator name, connected by underscores (such as, EMASSTRT_DYNROUTE_ACTIVE).
- The copy books use trigraphs, which are multi-character combinations, to represent square brackets.
- Any variable-length data that you send to the API must be padded with blanks to the end of the field. The API does not insert the zero-byte ending delimiter.

- The table length field is a concatenation of the resource table format name and the constant TBL_LEN, connected by an underscore (such as, EMASSTRT_TBL_LEN).

The resource table data types are mapped into the valid set of C data types. However, exact mapping is not always possible. The resource table data types are mapped as follows:

```
          char  - 1-byte binary numeric values

     short int  - 2-byte binary numeric values

          long  - 4-byte binary numeric values
                - 4-byte intervals

        long 2  - Time stamps and 8-byte intervals
                  8- byte binary numeric values
                   (an array of two fullwords)

      char nnnn - Packed decimal data

      char nnnn - Character data
                - Binary values greater than 8 bytes
                - Odd number binary values less than 8 bytes
```

Figure 16 is a representative extract of a C resource table copy book:

```
/*-------------------------------------------------------------------*
 * Name = EYUC2400                                                   *
 * Format Name = EMASSTRT                                            *
 * Version = 0001                                                    *
 * Status = CPSMREL(0310)                                            *
 * Function = Base Table Structure generator                         *
 * Format definition for this element = EMASSTRT                     *
 * Valid Operations = None                                           *
 * Valid Actions = None                                              *
 *-------------------------------------------------------------------*/
typedef struct EMASSTRT  {
 char      CMASNAME??(8??);       /* CMAS Name                      */
 char      PLEXNAME??(8??);       /* CICSPlex Name                  */
 char      CSYSNAME??(8??);       /* CICS System Name               */
 char      MON_SPEC??(8??);       /* Monitor Spec Name              */
 char      RTA_SPEC??(8??);       /* Real Time Analysis Spec Name   */
 char      WLM_SPEC??(8??);       /* Work Load Manager Spec Name    */
 char      STATUS;                /* Status                         */
 char      DYNROUTE;              /* Dynamic Routing Mode           */
 char      DYNTYPE??(3??);        /* Dynamic Routing Type           */
 char      DESC??(30??);          /* Description                    */
 char      CSYSAPPL??(8??);       /* CICS System VTAM APPLID        */
 char      EYU_RSV0015??(5??);    /* Alignment Padding              */
 long      MASSTART??(2??);       /* MAS Start STCK Value           */
 char      TMEZONEO;              /* Time Zone Offset               */
 char      TMEZONE;               /* Time Zone                      */
 char      EYU_RSV0019??(2??);    /* Alignment Padding              */
 long      DAYLGHTSV;             /* DayLight savings in effect     */
 char      SYSID??(4??);          /* MAS System Id                  */
 char      OPSYSREL??(4??);       /* MAS Op Sys Release             */
 char      MVSNAME??(4??);        /* MVS System Name                */
 char      JOBNAME??(8??);        /* MAS Job Name                   */
 char      CECNAME??(8??);        /* CEC Name                       */
 char      SYSPLEX??(8??);        /* SYSPlex Name                   */
 char      EYU_RSV0257??(4??);    /* Alignment Padding              */
  } EMASSTRT;
```

*Figure 16. Sample C copy book (Part 1 of 2)*

```
/*-------------------------------------------------------------------*
 *                                                                   *
 * EMASSTRT Defines for Table                                        *
 *                                                                   *
 *-------------------------------------------------------------------*/
 #define  EMASSTRT_STATUS_LOCAL          128
 #define  EMASSTRT_STATUS_REMOTE         64
 #define  EMASSTRT_DYNROUTE_ACTIVE       1
 #define  EMASSTRT_DYNROUTE_SUSPEND      2
 #define  EMASSTRT_DYNTYPE_WLMTOR        "TOR"
 #define  EMASSTRT_DYNTYPE_WLMAOR        "AOR"
 #define  EMASSTRT_TBL_LEN               152
```

*Figure 16. Sample C copy book (Part 2 of 2)*

## Language and environment considerations

This section describes various language and environment considerations that you should keep in mind when writing a CICSPlex SM API program. Note that all of the usual language considerations that apply to the various environments (CICS, MVS batch, TSO, and NetView) also apply to CICSPlex SM programs written to run in those environments.

## Assembler considerations

For Assembler programs that run in an MVS batch, TSO, or NetView environment, you need to be aware of the following special considerations:

- Since the program does not execute in CICS, do not use the DFHEIENT or DFHEIRET macros. Instead, use the CICS translator options NOEPILOG, NOPROLOG, and NOSYSEIB.
- You must explicitly code the DFHEISTG and DFHEIEND macros to provide the required work areas for EXEC CPSM commands. Your program is responsible for acquiring storage for the DFHEISTG area and setting up any necessary base registers prior to making any EXEC CPSM calls. This storage can be acquired dynamically using local GETMAIN services or, if the program is nonreentrant, the storage can be defined directly in the program area. Reentrant programs are recommended if there is any possibility of the program being used concurrently in the same address space.
- You must make the appropriate CICS macro library available in the SYSLIB concatenation for the Assembler step. The DFHEISTG, DFHEIEND, and DFHSCALL macros are fetched from this library.

## PL/I considerations

For PL/I programs, you need to be aware of the following special considerations:

- The variable EYUPTPTR must be explicitly declared as follows:

```
DCL EYUPTPTR POINTER;
```

## NetView considerations

If you plan to run C programs under NetView, you need to be aware of the following special considerations:

- Depending on which resource tables you access, you may encounter some name conflicts between the CICSPlex SM #define statements for resource table attributes and the standard NetView #define statements. For example, the NetView statement `#include "dsic.h"` generates the following define statement:

  ```
  #define COMMAND "COMMAND "
  ```

  Some of the CICSPlex SM resource tables use COMMAND as an attribute name. If you use `#include "dsic.h"` as supplied by NetView, the resource table attribute names are converted and cannot be processed by CICSPlex SM.

  One way of handling any potential conflicts is to undefine the COMMAND value, like this:

  ```
  #include "dsic.h"
  #undef COMMAND
  #include "feedback.h"
        .
        .
        .
  ```

  If you want to, you can also redefine the COMMAND value using a new name that does not conflict with any resource table attribute name, like this:

  ```
  #include "dsic.h"
  #undef COMMAND
  #define XCOMMAND "COMMAND "
  #include "feedback.h"
        .
        .
        .
  ```

## User-replaceable programs

The CICSPlex SM API cannot be used from within the user-replaceable programs EYU9XESV and EYU9WRAM.

## CICS Global User exit programs

The CICSPlex SM API may be used from within the CICS XICEREQ Global User Exit program. You must avoid recursion within the CICSPlex SM API program and the exit should not delay any requests issued by CPSM related tasks.

The use of the CICSPlex SM API from within other CICS Global User Exit points is not recommended as the results are unpredictable.

## Status programs

The CICSPlex SM API cannot be used from within a program that is invoked through the STATDEF view. Where access to the API is required, you must start another task and invoke the API from the new task.

# Translating your program

This section describes **separate translation** which is the process of converting programs into executable code that the compiler (or assembler) can understand.

Some compilers allow you to use the **integrated CICS translator** approach, where the compiler interfaces with CICS at compile time to interpret CICS commands and convert them automatically to calls to CICS service routines. If you use the integrated CICS translator approach many of the translation tasks are done for you. For details of the integrated CICS translator see the *CICS Application Programming Guide*.

For programs written using the command-level interface, you must use a language translator to interpret the source program for the API. Any external program that contains EXEC CPSM commands must be processed by the appropriate version of the CICS/ESA command level translator.

The following versions of the CICS translator support EXEC CPSM commands:

- CICS/ESA 4.1 with APAR PN73812
- CICS TS

**Notes:**

1. If you are using the CICS/ESA 4.1 version of the translator, make sure the appropriate APAR has been applied before you attempt to translate your program.
2. If you are using Business Application Services (BAS) to create CICS resource definitions, be sure to use the appropriate version of the translator for the definitions you are creating. That is, if you want to create CICS TS for OS/390 resource definitions, you must use the translator that is distributed with that version of CICS.

## Specifying the CPSM translator option

Because CICSPlex SM uses the CICS/ESA translator, you can use your CICS translate JCL as a model for translating CICSPlex SM API programs. You must specify one additional translator option, called CPSM, in order to translate

CICSPlex SM programs. The CPSM option can be specified by using either the PARM operand of the EXEC statement or a language-specific XOPTS options statement.

If your program also contains EXEC CICS commands, those commands are processed in the same translation step. The CICS translator inserts the necessary variable and invocation definitions required for proper execution of the program.

When using the CPSM API in a non-CICS environment, be sure to remove any CICS or SP translator options, and only specify the CPSM translator option.

As a result of the translation process, EXEC CPSM statements are replaced with language specific calls to an EXEC interface stub program.

### Sample Assembler translation

To specify the CPSM translator option, use either the PARM operand of the EXEC statement, like this:

```
//TRANSLAT EXEC PGM=DFHEAP1$,PARM='CPSM',REGION=4096K
```

or an XOPTS options statement, like this:

```
*ASM XOPTS(...CPSM)
```

### Sample PL/I translation

To specify the CPSM translator option, use either the PARM operand of the EXEC statement, like this:

```
//TRANSLAT EXEC PGM=DFHEPP1$,PARM='CPSM',REGION=4096K
```

or an XOPTS options statement, like this:

```
*PROCESS XOPTS(...CPSM)
```

### Sample COBOL translation

To specify the CPSM translator option, use either the PARM operand of the EXEC statement, like this:

```
//TRANSLAT EXEC PGM=DFHECP1$,PARM='COBOL3,CPSM',REGION=4096K
```

or ( for the separate translator) an XOPTS options statement, like this:

```
PROCESS XOPTS(...CPSM)
```

or (for the integrated translator) a CICS compiler option like this:

```
CICS('opt1 opt2 optn ...')
```

Note that when you translate a COBOL program, you must specify both the CPSM and the COBOL3 translator options.

### Sample C translation

To specify the CPSM translator option, use either the PARM operand of the EXEC statement, like this:

```
//TRANSLAT EXEC PGM=DFHEDP1$,PARM='CPSM',REGION=4096K
```

or an XOPTS options statement, like this:

```
#pragma XOPTS(...CPSM)
```

# Compiling your program

Compiling a CICSPlex SM API program is similar to compiling a CICS program. You can use your CICS compile JCL as a model and then make the following modifications according to the language you are using.

## Assembler considerations

See the *CICS Transaction Server for z/OS Release Guide* for details of supported assemblers.

To assemble CICSPlex SM programs, you must include a SYSLIB statement for the CICSTS31.CPSM.SEYUMAC macro library in your compile JCL, like this:

```
//ASM     EXEC PGM=ASMA90,REGION=4096K
                          .
                          .
                          .
//SYSLIB   DD DSN=CICSTS31.CPSM.SEYUMAC,DISP=SHR
                          .
                          .
                          .
```

## PL/I considerations

See the *CICS Transaction Server for z/OS Release Guide*; for details of supported PL/I compilers.

To compile CICSPlex SM programs, you must include a SYSLIB statement for the CICSTS31.CPSM.SEYUPL1 macro library in your compile JCL, like this:

```
//COMPILE EXEC PGM=IEL0AA,REGION=1000K,
//       PARM='OBJECT,MACRO,LIST'
                    .
                    .
                    .
//SYSLIB   DD DSN=CICSTS31.CPSM.SEYUPL1,DISP=SHR
                    .
                    .
                    .
```

## COBOL considerations

See the *CICS Transaction Server for z/OS Release Guide*; for details of supported COBOL compilers.

To compile CICSPlex SM programs, you must include a SYSLIB statement for the CICSTS31.CPSM.SEYUCOB macro library in your compile JCL, like this:

```
//COMPILE EXEC PGM=IGYCRCTL,REGION=4096K
                          .
                          .
                          .
//SYSLIB   DD DSN=CICSTS31.CPSM.SEYUCOB,DISP=SHR
                          .
                          .
                          .
```

## C considerations

See the *CICS Transaction Server for z/OS Release Guide*; for details of supported C compilers.

To compile CICSPlex SM programs, you must include a SYSLIB statement for the CICSTS31.CPSM.SEYUC370 macro library in your compile JCL, like this:

```
//COMPILE EXEC PGM=EDCCOMP,REGION=4096K
                          .
                          .
                          .
//SYSLIB   DD DSN=CICSTS31.CPSM.SEYUC370,DISP=SHR
                          .
                          .
                          .
```

## Link editing your program

The CICS/ESA translator inserts a call to the CICSPlex SM EXEC interface stub program. The stub entry name is not the name of an object or load module. Since CICSPlex SM API programs can run in a variety of environments, the stub reference must be resolved to a module consistent with the intended usage. This resolution is performed at link-edit time using the INCLUDE linkage editor control statement.

You must link edit all program load modules with the correct CICSPlex SM stub module for the environment where the program will run. To do this, specify one of the following stub modules in the INCLUDE statement:

**EYU9AMSI**
: For CICS/ESA and CICS TS programs. EYU9AMSI is supplied in the CICSTS31.CPSM.SEYULOAD library.

**EYU9ABSI**
: For batch, TSO, or NetView programs. EYU9ABSI is supplied in the CICSTS31.CPSM.SEYUAUTH library.

Each of these stub modules contains the appropriate entrypoint identifier. The services provided by the entrypoint are unique to the type of execution environment.

**Note:** You should not attempt to run a program identified as a CICS program in a batch environment. Likewise, batch programs are not suitable for running under CICS.

You can use your CICS link-edit JCL as a model for link editing CICSPlex SM programs. Be sure to review the language-specific considerations in the remainder of this section and modify your JCL accordingly.

In addition, if your program contains EXEC CICS commands, you should review the link-edit considerations in the *Application Programming Guide* for your version of CICS. Likewise, if your program runs under NetView, you should refer to the NetView customization book for your programming language, either *Customization: Using Assembler*, or *Customization: Using PL/I and C*.

## Assembler considerations

Assembler load modules can reside in 24- or 31-bit storage and can be entered in either addressing mode.

To link edit an Assembler module to run with a CICSPlex SM program, you must include a SYSLIB statement for the SEYULOAD load library in your link-edit step. This allows you to include the appropriate CICSPlex SM stub module when link editing. For example:

```
//LKED     EXEC  PGM IEWL,
//    PARM='XREF,LET,LIST,AMODE=ANY,RMODE=31',
//    REGION=4096K,COND=(7,LT,ASM)
                                .
                                .
                                .
//SYSLIB   DD DSN=CICSTS31.CPSM.SEYULOAD,DISP=SHR
                                .
                                .
                                .
  INCLUDE  SYSLIB(userprog)
  INCLUDE  SYSLIB(EYU9AMSI)
  NAME     LMODNAME(R)
```

## PL/I, COBOL, and C considerations

PL/I, COBOL, and C load modules can reside in 24- or 31-bit storage and can be entered in either addressing mode.

To link edit a module to run with a CICSPlex SM program, you must include a SYSLIB statement for the SEYULOAD load library in your link-edit step. This allows you to include the appropriate CICSPlex SM stub module when link editing. For example:

```
//LKED     EXEC  PGM=IEWL,
//    PARM='XREF,LET,LIST,AMODE=ANY,RMODE=31',
//    REGION=4096K,COND=(8,LE,COMPILE)
                      .
                      .
                      .
//SYSLIB   DD DSN=CICSTS31.CPSM.SEYULOAD,DISP=SHR
                      .
                      .
                      .
  INCLUDE  SYSLIB(userprog)
  INCLUDE  SYSLIB(EYU9AMSI)
  NAME     LMODNAME(R)
```

## Run-time considerations

- The run-time version of a CICSPlex SM API program is equal to the level of the CMAS to which it connects:
  - For a program written to run as a CICS application, the run-time version is that of the CMAS to which the MAS is connected.
  - For a program written to run as a batch job or under NetView or TSO, the version is determined by the version of the CICSPlex SM run-time module (EYU9AB00).

    EYU9AB00 is distributed in CICSTS22.CPSM. At run time, CICSPlex SM must find EYU9AB00 in the STEPLIB, MVS linklist, or LPA library concatenation.
- The run-time version of a program must be greater than or equal to:
  - The version of the stub routine module (EYU9AxSI) with which the program was link edited.
  - The value specified on the VERSION option of the CONNECT command.
- For programs written in PL/I, COBOL, or C, a set of run-time libraries is shipped with the language compiler. To run a CICSPlex SM program written in one of these languages, you must modify your environment startup procedure to reference the appropriate run-time libraries for the language.

- Before running any CICSPlex SM program under CICS, make sure the program and its associated transaction are defined to CEDA. The program may be defined with an EXECKEY value of either User or CICS. The associated transaction may be defined with a TASKDATAKEY value of either User or CICS.

# Chapter 4. Dealing with exception conditions

This chapter describes the tools and techniques that are available for dealing with error conditions in a CICSPlex SM API program.

**Note:** For information on additional diagnostic data that is available for an API program, refer to *CICSPlex System Manager Problem Determination*.

## Default CICSPlex SM exception handling

The CICSPlex SM API writes an exception trace, in the form of a user trace record, to the CICS trace data set. Resources available via the CICSPlex SM API are not recoverable, and, therefore, resources updated prior to the exception are neither recovered nor are they available for backout by the application using EXEC CICS SYNCPOINT and EXEC CICS SYNCPOINT ROLLBACK commands.

## Using the RESPONSE and REASON options

The RESPONSE and REASON options are required on each API command. You should specify these options as user-defined variables to receive the numeric response and reason values returned by a command. You can then convert the numeric values into more meaningful character equivalents. In general, RESPONSE describes the result of command processing and REASON further qualifies the response to certain commands.

**Note:** The TBUILD and TPARSE commands, which can be used only with the REXX run-time interface, do not use the RESPONSE and REASON options. The result of these REXX-specific processes is returned by their STATUS option. For more information, see Chapter 6, “REXX error handling,” on page 109.

## Types of responses

There are three types of responses that an API command can return:

- Normal
- Warning
- Error.

The character equivalents of the RESPONSE and REASON values that can be returned are given in the description of each command. For a summary of RESPONSE and REASON character values by command, see *CICSPlex System Manager Application Programming Reference*. For a list of RESPONSE and REASON character values and their numeric equivalents, also see *CICSPlex System Manager Application Programming Reference*.

### Normal responses

A normal response indicates the API command completed processing successfully. The following values represent a normal response:

**OK** The command was successfully processed and control was returned to the program. There are no reasons associated with a response of OK.

**SCHEDULED**
A command that was issued with the NOWAIT option has been scheduled for processing. The actual result of command processing is returned by the

RECEIVE command in an ASYNCREQ resource table record. There are no reasons associated with a response of SCHEDULED.

## Warning responses

A warning response indicates the API command was successfully processed, but a condition occurred that should be investigated. A REASON value is also returned that describes the condition. The following values represent a warning response:

**NODATA**
A command that normally results in data being returned to the program was processed successfully, but there was no data to return. The reasons for a NODATA response are given with the commands that return it.

**WARNING**
A command that normally results in data being returned to the program was processed successfully, but not all of the available data was returned. A typical reason for this response might be that the output area provided by the program was not large enough to hold all the data. The actual reasons for a WARNING response are given with the commands that return it.

## Error responses

An error response indicates the API command was not successful. One or more REASON values are also returned that describe the error.

**Note:** Note that, except for the FAILED error response, these response codes usually indicate either an error in the user's API program (for example, failing to discard resources when they are no longer required), or an error with the CICSPlex SM environment (for example, a CMAS or MAS is not available).

The following values represent an error response:

**BUSY** A resource referred to by the command is currently being processed by another command. This situation can occur when a command that was previously issued with the NOWAIT option is processing a resource that is required by the current command. The reasons for a BUSY response are given with the commands that return it.

**DUPE** A resource referred to by the command already exists. The reasons for a DUPE response are given with the commands that return it.

**ENVIRONERROR**
An environmental condition (such as short on storage) prevented the command from being processed. The reasons for an ENVIRONERROR response are given with the commands that return it.

**FAILED**
An unexpected problem occurred during command processing. The reasons for a FAILED response are given with the commands that return it.

In the case of a FAILED EXCEPTION response, you should check the following sources for information related to the condition:

- EYULOG
- Job log
- AUXTRACE data set

**INCOMPATIBLE**
Two or more resources referred to by the command are incompatible. The reasons for an INCOMPATIBLE response are given with the commands that return it.

**INUSE**
A resource referred to by the command is in use and, therefore, cannot be discarded. The reasons for an INUSE response are given with the commands that return it.

**INVALIDATA**
The command parameter list contains invalid data. The reason for an INVALIDATA response is always the name of the parameter that contains invalid data. The reasons are given with the commands that return this response.

**INVALIDCMD**
The command is invalid as indicated by the reason code:

**Filter** The filter that is being built is too large or complex.

**Length**
The total length of all the inputs used in the command exceeds the maximum limit.

**N_A** The command is invalid. Check which version of the CICS translator was used to translate the API command. Also check that the command being used is available on the CICSPlex SM release that the program is using.

**INVALIDPARM**
The command parameter list is invalid. There are a variety of situations that could result in an INVALIDPARM response. For example:

**Syntax error**
The syntax of an input parameter is incorrect (for example, a resource table name begins with a numeric character).

**Null parameter address**
An input parameter could not be found because the generated address for that parameter is 0.

The reason for an INVALIDPARM response is always the name of the parameter that is invalid. The reasons are given with the commands that return this response.

**NOTAVAILABLE**
A required CMAS or MAS resource is not available. The reasons for a NOTAVAILABLE response are given with the commands that return it.

**NOTFOUND**
A resource referred to by the command could not be found. The reasons for a NOTFOUND response are given with the commands that return it.

**NOTPERMIT**
The API request is not permitted by the external security manager (ESM) at your enterprise. The reasons for a NOTPERMIT response are given with the commands that return it.

**SERVERGONE**
The CMAS to which the processing thread was connected is no longer active. There are no reasons associated with a response of SERVERGONE.

**TABLEERROR**
An error was detected in a resource table record (either a result set record

or a CICSPlex SM definition record). The reasons for a TABLEERROR response are given with the commands that return it.

**VERSIONINVL**
An invalid version of CICSPlex SM was detected. The reasons for a VERSIONINVL response are given with the commands that return it.

# Testing for RESPONSE and REASON

To evaluate the results of an API command, you simply code the RESPONSE and REASON options on the command and follow the command immediately with a test of the returned values. The RESPONSE and REASON options return numeric values. Different built-in functions are provided for converting and testing the numeric response and reason values in the command-level interface and the REXX run-time interface.

## Using the command-level interface

When you are using the CICSPlex SM command-level interface, you can use the EYUVALUE built-in function to convert and test the numeric RESPONSE and REASON values returned by an API command.

As an example, consider this API command:

```
EXEC CPSM CONNECT
          CONTEXT(WCONTEXT)
          SCOPE(WSCOPE)
          VERSION('0310')
          THREAD(WTHREAD)
          RESPONSE(WRESPONSE)
          REASON(WREASON)
   .
   .
```

To test for the RESPONSE value in each of the supported languages, you could code:

**COBOL or PL/I:**

```
IF WRESPONSE NOT = EYUVALUE(OK) GO TO NOCONNECT.
```

**C:**

```
if (WRESPONSE ¬= EYUVALUE(OK)) { goto NOCONNECT; }
```

**Assembler language:**

```
CLC   WRESPONSE,EYUVALUE(OK)
BNE   NOCONNECT
```

which the built-in function changes to:

```
CLC   WRESPONSE,=F'1024'
```

You can use EYUVALUE in the same way to test for the REASON value, if the RESPONSE is one that returns a reason.

## Using the REXX run-time interface

When you are using the REXX run-time interface, you can use the EYURESP and EYUREAS built-in functions to convert and test the numeric RESPONSE and REASON values returned by an API command.

As an example, consider this API command:

```
var = EYUAPI('CONNECT'             ,
             'CONTEXT('WCONTEXT')' ,
             'SCOPE('WSCOPE')'     ,
```

```
                'VERSION(0310)'      ,
                'THREAD(WTHREAD)'      ,
                'RESPONSE(WRESPONSE)'  ,
                'REASON(WREASON)')
      .
      .
```

To test for the RESPONSE value, you could code:

```
If WRESPONSE <> EYURESP(OK) Then Signal NOCONNECT
```

to compare the numeric RESPONSE value returned in WRESPONSE with the numeric equivalent of OK.

Alternatively, you could code:

```
If EYURESP(WRESPONSE) <> "OK" Then Signal NOCONNECT
```

to convert the numeric RESPONSE value to its character equivalent first.

**Note:** The RESPONSE and REASON options report only run-time errors. Errors in interpreting an API command are reported in either the REXX RC variable or the variable assigned to a REXX function.

## Retrieving FEEDBACK records

In addition to the specific values returned by a command's RESPONSE and REASON options, CICSPlex SM also provides diagnostic data in the form of FEEDBACK resource table records. This data can help you evaluate the results of an API command, especially if the command did not complete successfully.

## Using the FEEDBACK command

You can retrieve diagnostic data about a previously issued API command by issuing the FEEDBACK command. The type of command for which you want diagnostic data affects how you specify the FEEDBACK command and where the data is placed:

**A command that processed a result set**

Use the RESULT option of the FEEDBACK command to retrieve data about the last command that processed a specific result set.

If the command that processed the result set returned a RESPONSE other than OK, a FEEDBACK resource table record is appended to the end of each resource table record in the result set that had an error associated with it. You can use the FIRST, NEXT, and COUNT options of the FEEDBACK command to retrieve multiple FEEDBACK records.

The diagnostic data in a result set is available to the FEEDBACK command until another command processes the same result set. At that point, the data is replaced with FEEDBACK records for the subsequent command.

**Note:** No FEEDBACK records are produced if the command that processed the result set returned a RESPONSE of OK.

**A command that did not process a result set**

Use the FEEDBACK command without the RESULT option to retrieve data about the command issued immediately before FEEDBACK.

The FEEDBACK resource table records are returned in a separate feedback area. The records in that feedback area are cleared and refreshed

for each command that is not result set-oriented. So for commands that place their diagnostic data in the feedback area rather than in a result set, FEEDBACK can retrieve data only for the most recently issued command.

Once you have issued the FEEDBACK command to retrieve diagnostic data for a command, the feedback record or area is cleared. You cannot request the same FEEDBACK resource table records more than once.

## Evaluating a FEEDBACK record

The diagnostic data for a CICSPlex SM API command is presented in a FEEDBACK resource table record. The attributes of that resource table provide a variety of information about the completion status of an API command.

**Note:** This section provides general information about FEEDBACK records. The FEEDBACK resource table copy book that is supplied by CICSPlex SM provides a detailed description of the contents and structure of a FEEDBACK record. You should refer to the *CICSPlex System Manager Resource Tables Reference* or the supplied copy book when writing a program that uses the FEEDBACK command.

To identify which API operation the FEEDBACK record applies to, check the values in these fields:

**COMMAND**
A numeric code that identifies the command to which this FEEDBACK record applies. The API commands and their numeric equivalents are given in Table 8.

**OBJECT**
The CICSPlex SM object that the command was issued against.

**OBJECT_ACT**
The action that was being performed against the CICSPlex SM object.

**RSLTRECID**
If the FEEDBACK record applies to a result set, the numeric ID of the result set record associated with this FEEDBACK record.

*Table 8. Numeric codes and API commands*

| Numeric code | Mnemonic | Command |
|---|---|---|
| 02 | CANCEL | Cancel |
| 03 | CONNECT | Connect |
| 04 | COPY | Copy |
| 05 | CREATE | Create |
| 06 | DELETE | Delete |
| 07 | DISCARD | Discard |
| 08 | DISCONN | Disconnect |
| 09 | FETCH | Fetch |
| 10 | GET | Get |
| 11 | LOCATE | Locate |
| 12 | MARK | Mark |
| 13 | ORDER | Order |
| 14 | PERFSET | Perform Set |

*Table 8. Numeric codes and API commands (continued)*

| Numeric code | Mnemonic | Command |
|---|---|---|
| 15 | PERFOBJ | Perform Object |
| 16 | QUALIFY | Qualify |
| 17 | QUERY | Query |
| 18 | RECEIVE | Receive |
| 19 | REMOVE | Remove |
| 20 | FILTER | Specify Filter |
| 21 | UNMARK | Unmark |
| 22 | ADDRESS | Address |
| 23 | GETDEF | Getdef |
| 24 | LISTEN | Listen |
| 25 | REFRESH | Refresh |
| 26 | SET | Set |
| 27 | VIEW | Specify View |
| 28 | TERM | Terminate |
| 29 | TRANS | Translate |
| 30 | GROUP | Group by |
| 31 | UPDATE | Update |

To determine what type of problem the FEEDBACK record describes, check the values in these fields:

**ATTRDATAVAL**

Indicates whether attribute data is available for the command. Attribute data is included only if the command itself did not complete successfully.

If the ATTRDATAVAL value is Y, the FEEDBACK record identifies as many as five attributes (ATTR_NM1 through ATTR_NM5) that contributed to the error. Each attribute is identified by its name and its offset and relative number within the resource table record. The data type and length of each attribute is also included.

If the ATTRDATAVAL value is N, you can ignore the ATTR_ fields.

**CEIBDATAVAL**

Indicates whether CICS EIB data is available for the command. EIB data is included only if the command encountered a CICS error.

If the CEIBDATAVAL value is Y, the FEEDBACK record includes the EIBFN, RESP, and RESP2 values as provided by CICS. Note that if the RESP value indicates a NOTAUTH condition that was raised due to CICSPlex SM simulated security, EIBFN is not set.

If the CEIBDATAVAL value is N, you can ignore the CEIBFN, CEIBRESP, and CEIBRESP1 fields.

**ERRCODEVAL**

Indicates whether a CICSPlex SM error code is available for the command. An error code is included only if the command itself did not complete successfully.

If the ERRCODEVAL value is Y, the FEEDBACK record includes a numeric ERROR_CODE value. Each resource table copy book includes a list of the error codes for that object and their meanings.

If the ERRCODEVAL value is N, you can ignore the ERROR_CODE field, as well as the RESPONSE and REASON fields.

For some API operations that affect BAS resources, the FEEDBACK record may point to additional diagnostic data in an error result set. For more information about using the diagnostic data in error result sets, see “Additional processing for BAS” on page 95.

## Availability of FEEDBACK records

In general, FEEDBACK records are produced for all API commands, whether they are successful or not. However, for some API commands and in some situations, FEEDBACK records are not produced because they would not provide useful diagnostic data.

FEEDBACK records are not available for these commands:

**DISCONNECT and TERMINATE**
When you disconnect an API processing thread from CICSPlex SM, any remaining diagnostic data is discarded.

**FEEDBACK**
The FEEDBACK command cannot report on its own processing.

**TBUILD and TPARSE**
These REXX-specific commands issue a series of API commands internally and reuse the same feedback area. Therefore, the feedback area cannot represent the entire sequence of events.

FEEDBACK records are also not available in these situations:

- A command processes a result set and completes with a RESPONSE value of OK, and no additional information was returned by CICS in the EIBRESP2 field.
- A command is processed asynchronously (that is, you specify the NOWAIT option). The diagnostic data for asynchronous requests is returned in the ASYNCREQ notification resource table.

## An example of FEEDBACK for a result set

As an example of how you can use FEEDBACK data, Figure 17 on page 95 illustrates the results of issuing a SET command. In this case, SET was issued to modify the service status of CONNECT records in the result set referenced by TOKENC.

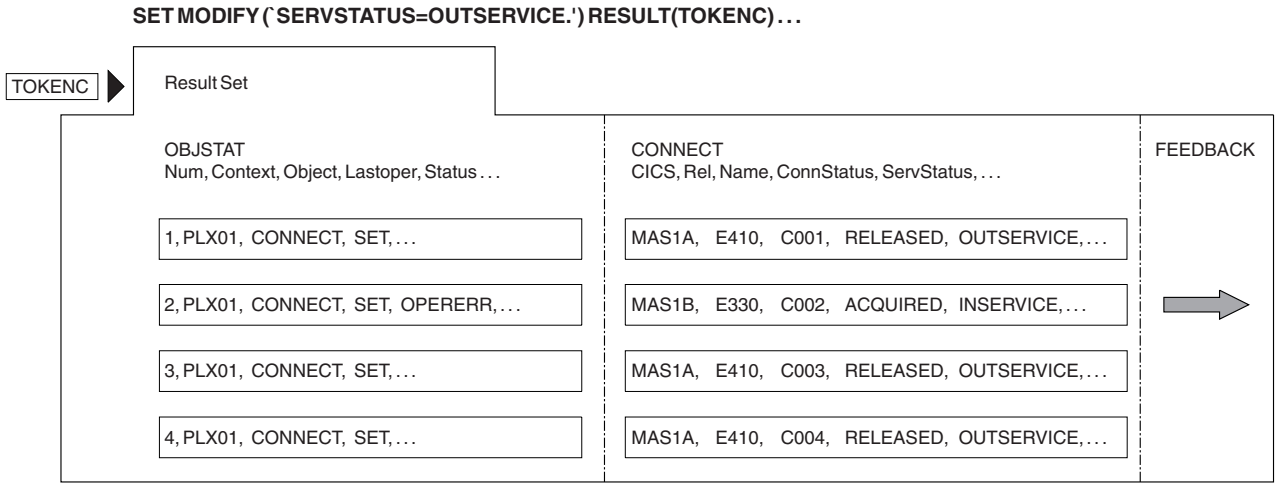



*Figure 17. Using SET to modify result set records*

One of the connections (C002 in MAS1B) was not successfully taken out of service by the SET command. The ServStatus field is still set to INSERVICE and there is a pointer to FEEDBACK data.

Figure 18 shows how you can use the FEEDBACK command to retrieve the FEEDBACK records associated with the result set referenced by TOKENC. The FEEDBACK record shown in Figure 18 reveals the cause of the problem.

```
FEEDBACK RESULT(TOKENC) INTO(AREA5) . . .

SET, N, Y, N, TABLEERROR, DATAERROR, . . ., 16, 2, . . ., CONNECT, . . .
```

*Figure 18. Using FEEDBACK to retrieve diagnostic data for a result set*

CICSPlex SM returned RESPONSE and REASON values of TABLEERROR DATAERROR, which means the value associated with one or more resource table attributes is invalid. Furthermore, CICS responded to the SET request for this connection with RESP(16) RESP2(2). A check of the CICS response codes indicates that the attempt to take the connection out of service was invalid because the connection is currently acquired.

**Note:** The LASTOPER and STATUS attributes of the OBJSTAT resource table and some of the FEEDBACK attributes are actually binary fields (that is, they are represented by a bit being set on or off). For detailed information about the attribute values for a given resource table, refer to the *CICSPlex System Manager Resource Tables Reference* or the supplied copy books.

## Additional processing for BAS

For API operations that affect BAS resources, the diagnostic data in a FEEDBACK record may not be enough to fully describe an error condition. In these cases, the FEEDBACK record points to an error result set. An error result set is identified by the following fields:

**ERR_RESULT**
A 4-byte token identifying an error result set.

**ERR_COUNT**
The number of records in the error result set referenced by ERR_RESULT.

**ERR_OBJECT**
The type of records in the error result set referenced by ERR_RESULT. This value is the 1- to 8-character name of a CICSPlex SM resource table, and may be BINSTERR, BINCONRS, BINCONSC, or FEEDBACK.

**Note:** For details of the BINSTERR, BINCONRS, and BINCONSC resource tables, see the *CICSPlex System Manager Resource Tables Reference*.

## Evaluating error result set records

If the ERR_OBJECT field of the FEEDBACK record contains FEEDBACK, the error result set contains errors that arose when CICSPlex SM attempted to update CICS resources. In response to the API command:

```
UPDATE RESULT(token) MODIFY(string)
```

CICSPlex SM tries to update multiple CICS definition records in a result set according to the supplied modification string. For each CICS definition that could not be modified, an error record is created in the error result set. The RESPONSE and REASON values returned are TABLEERROR and DATAERROR.

The records are standard FEEDBACK records. To access the error result records, use the FEEDBACK command to retrieve diagnostic data about each of the CICS definitions in the ERR_RESULT result set. The ERR_COUNT value in the original FEEDBACK record for the UPDATE command indicates how many records are in the ERR_RESULT result set and therefore the number of times you should issue the FEEDBACK command against the ERR_RESULT result set.

## Evaluating BINSTERR resource table records

If the ERR_OBJECT field of the FEEDBACK record contains BINSTERR, errors were encountered while CICS resources were being installed. In response to one of the following API commands:

```
PERFORM OBJECT ACTION(INSTALL)
PERFORM SET ACTION(INSTALL)
```

CICSPlex SM tries to install CICS resources in one or more active systems running CICS/ESA 4.1 or later. A BINSTERR record is created for each CICS resource that cannot be installed. The RESPONSE and REASON values returned are TABLEERROR and DATAERROR.

The BINSTERR records that you receive contain the following information:

**CMASNAME**
The 1- to 8-character name of a CMAS that manages the specified CICSplex.

**PLEXNAME**
The 1- to 8-character name of the CICSplex to which the specified CICS system belongs.

**CICSNAME**
The 1- to 8-character name of the CICS system into which the resource could not be installed.

**RESNAME**
The name of the CICS resource that could not be installed.

**RESVER**
The version of the CICS definition that represents the resource being installed.

**ERRCODE**
A numeric CICSPlex SM error code. See “BINSTERR” on page 114. The BINSTERR resource table copy book also contains a list of the error codes and their meanings.

**CRESP1**
The RESP value as returned by CICS.

**CRESP2**
The RESP2 value as returned by CICS.

**CEIBFN**
The EIBFN value as returned by CICS.

To access the error result set records, use the FETCH command to retrieve the BINSTERR records from the ERR_RESULT result set. The ERR_COUNT value in the FEEDBACK record for the PERFORM command indicates how many records are in the ERR_RESULT result set and therefore the number of times you should issue the FETCH command against the ERR_RESULT result set.

## Evaluating BINCONRS resource table records

If the ERR_OBJECT field of the FEEDBACK record contains BINCONRS, inconsistent resource set errors were encountered when attempting to update or create the specified definition. In response to one of the following API commands:

```
CREATE OBJECT(basdef)
UPDATE OBJECT(basdef)
```

CICSPlex SM tries to create or update one of the following Business Application Services definitions:

- RASGNDEF (resource assignment)
- RASINDSC (resource assignment in resource description)
- RESDESC (resource description)
- RESGROUP (resource group)
- RESINDSC (resource group in resource description)

A BINCONRS resource table record is created for each CICS definition that would cause an inconsistent set error. The RESPONSE and REASON values returned are TABLEERROR and DATAERROR.

The BINCONRS records that you receive contain the following information:

**CMASNAME**
The 1- to 8-character name of a CMAS that manages the specified CICSplex.

**PLEXNAME**
The 1- to 8-character name of the CICSplex to which the specified CICS system belongs.

**CICSNAME**
The 1- to 8-character name of the CICS system that experienced inconsistent resource set errors.

**RESTYPE**
The type of CICS resource.

**ERROP**
A numeric value that identifies the operation being performed when the error occurred (such as updating a RASGNDEF). See “BINCONRS” on page 113. The BINCONRS resource table copy book also contains a list of the ERROP values and their meanings.

**CANDNAME**
The name of the candidate resource

**CANDVER**
The version of the candidate resource

**CANDRGRP**
The group of the candidate resource

**CANDRASG**
The assignment of the candidate resource

**CANDRDSC**
The description of the candidate resource

**CANDUSAGE**
The candidate assignment usage

**CANDSGRP**
The candidate system group

**CANDTYPE**
The candidate system type

**CANDASGOVR**
The candidate assignment override

**EXISTNAME**
The name of the existing resource

**EXISTVER**
The version of the existing resource

**EXISTRGRP**
The group of the existing resource

**EXISTRASG**
The assignment of the existing resource

**EXISTRDSC**
The description of the existing resource

**EXISTUSAGE**
The existing assignment usage

**EXISTSGRP**
The existing system group

**EXISTTYPE**
The existing system type

**EXISTASGOVR**
The existing assignment override

To access the error result records, use the FETCH command to retrieve the BINCONRS records from the ERR_RESULT result set. The ERR_COUNT value in

the FEEDBACK record for the CREATE or UPDATE command indicates how many records are in the ERR_RESULT result set and therefore the number of times you should issue the FETCH command against the ERR_RESULT result set.

## Evaluating BINCONSC resource table records

If the ERR_OBJECT field contains BINCONSC, inconsistent scope errors were encountered while attempting to update or create the specified definition. In response to one of the following API commands:

```
CREATE OBJECT(basdef)
UPDATE OBJECT(basdef)
```

CICSPlex SM tries to create or update one of the following Business Application Services definitions:

- RASGNDEF (resource assignment)
- RASINDSC (resource assignment in resource description)
- RESDESC (resource description)
- RESGROUP (resource group)
- RESINDSC (resource group in resource description)

A BINCONSC resource table record is created for each CICS definition that would cause an inconsistent scope error. The RESPONSE and REASON values returned are TABLEERROR and DATAERROR.

BINCONSC records contain the following information:

**CMASNAME**
The 1- to 8-character name of a CMAS that manages the specified CICSplex.

**PLEXNAME**
The 1- to 8-character name of the CICSplex to which the specified CICS system belongs.

**CICSNAME**
The 1- to 8-character name of the CICS system that experienced inconsistent scope errors.

**ERROP**
A numeric value that identifies the operation being performed when the error occurred (such as updating a RASGNDEF). See “BINCONSC” on page 113. The BINCONSC resource table copy book also contains a list of the ERROP values and their meanings.

**ERRCODE**
A numeric CICSPlex SM error code. See “BINCONSC” on page 113. The BINCONSC resource table copy book contains a list of the error codes and their meanings.

**TARGSCOPE**
The name of the target scope

**TARGRASG**
The assignment for the target scope

**TARGRDSC**
The description for the target

**RELSCOPE**
The name of the related scope

**RELRASG**
The assignment for the related scope

**RELRDSC**
The description for the related scope

To access the error result records, use the FETCH command to retrieve the BINCONSC records from the ERR_RESULT result set. The ERR_COUNT value in the FEEDBACK record for the CREATE or UPDATE command indicates how many records are in the ERR_RESULT result set and therefore the number of times you should issue the FETCH command against the ERR_RESULT result set.

## An example of a BAS error result set

As an example of how you can the FEEDBACK data to obtain BAS error result set information, Figure 19 illustrates the results of issuing a PERFORM OBJECT command. In this case, PERFORM OBJECT ACTION(INSTALL) was issued to install the CONNDEF definitions in the result set referenced by TOKENC.

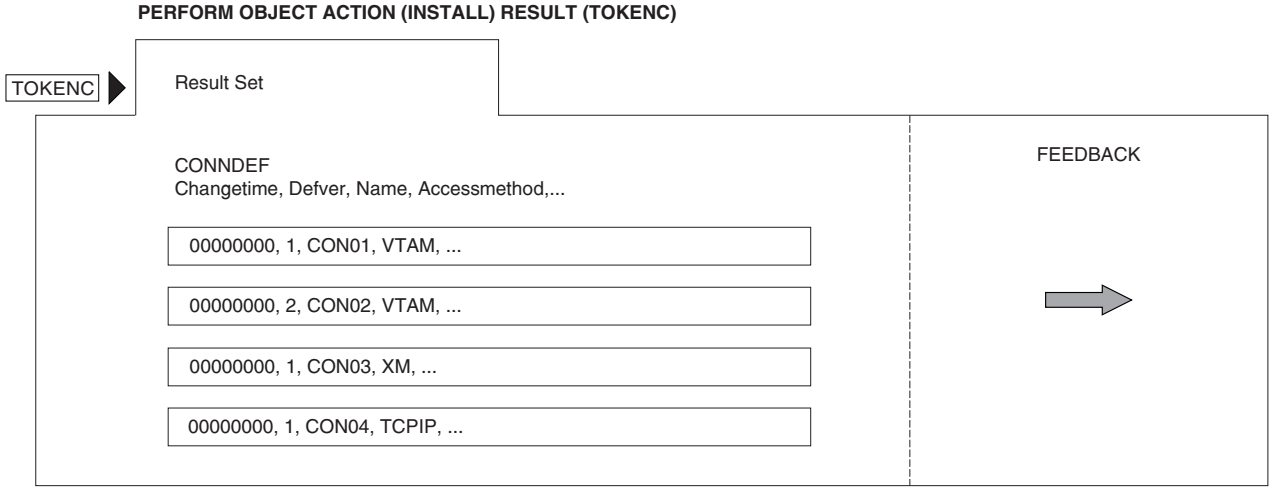



*Figure 19. Using PERFORM OBJECT to install BAS definitions*

One of the connection definitions (CON02,VTAM®) was not successfully installed by the PERFORM OBJECT command. There is a pointer to the FEEDBACK data.

Figure 20 shows how you can use the FEEDBACK command to retrieve the FEEDBACK records associated with the result set referenced by TOKENC.

```
FEEDBACK RESULT (TOKENC) INTO (AREA5) ...

PERFORM OBJECT, N,  Y,  N,  TABLEERROR,  DATAERROR, ..., ⟹ , 1, BINSTERR
```

*Figure 20. Using FEEDBACK to retrieve diagnostic data for a result set*

The FEEDBACK data shown in Figure 20 reveals the cause of the problem. CICSPlex SM returned RESPONSE and REASON values of TABLEERROR DATAERROR, which means that one or more connection definitions did not install successfully. Furthermore, the ERR_RESULT attribute points to an error result set which contains a single BINSTERR resource table record.

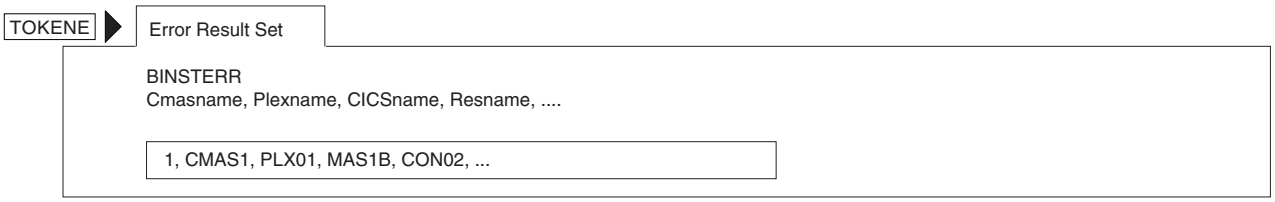



*Figure 21. BINSTERR error result set*

The BINSTERR error result set referenced by TOKENE, and shown in Figure 21 is accessed using a FETCH command.

```
FETCH RESULT (TOKENE) INTO (AREA6) ...
```

```
CMAS1, PLX01, MAS1B, CONO2, 2, FORCENO, ...
```

*Figure 22. Using FETCH to retrieve BINSTERR records*

Figure 22 shows how you can use the FETCH command to retrieve the BINSTERR records associated with the error result set referenced by TOKENE.

# Chapter 5. Writing a REXX program

This chapter describes how to use the REXX run-time interface to write an API program. It describes how to access the API through the REXX function package that is supplied with CICSPlex SM, how to specify an API command, and how to process the data in a resource table record.

## Accessing the API environment

The REXX run-time interface does not require any translation of API commands. The commands are interpreted by a REXX function package that is supplied by CICSPlex SM.

**Note:** For instructions on installing the REXX function package, see *CICS Transaction Server for z/OS Installation Guide*.

The REXX run-time interface consists of a single load module containing two entry points:

**EYU9AR00**
The function package

**EYU9AR01**
The host subcommand

The function package contains these functions:

**EYUAPI()**
Passes an API command to CICSPlex SM.

**EYUINIT()**
Initializes the CICSPlex SM API environment and allocates the necessary REXX resources.

**EYUREAS()**
Translates the value returned by the REASON option of an API command.

**EYURESP()**
Translates the value returned by the RESPONSE option of an API command.

**EYUTERM()**
Terminates the CICSPlex SM API environment and releases any allocated REXX resources.

For complete descriptions of these functions, see *CICSPlex System Manager Application Programming Reference*.

In general, you access the CICSPlex SM API environment through the supplied function package. The first call to CICSPlex SM in your program must be an EYUINIT or EYUAPI function. EYUINIT is the primary means of initializing the API environment. However, if EYUINIT is not issued first, the EYUAPI function initializes the environment.

For example, sample program EYU#API1, which is distributed in the CICSTS22.CPSM.SEYUSAMP library, begins like this:

```
Say 'Initializing API...'
XX = EYUINIT()
If XX <> 0 Then Signal UNEXPECTED
```

```
Say 'Establishing connection...'
XX = EYUAPI('CONNECT' ,
            'CONTEXT('W_CONTEXT')' ,
            'SCOPE('W_SCOPE')' ,
            'VERSION(0310)' ,
            'THREAD(W_THREAD)' ,
            'RESPONSE(W_RESPONSE)' ,
            'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
```

In this example, the EYUINIT function is issued first to initialize the API environment. Then an EYUAPI function is used to issue the API CONNECT command.

Once you have issued an EYUINIT or EYUAPI function, you can:

- Issue any other CICSPlex SM function.
- Access the host subcommand environment by issuing the REXX ADDRESS command.

Once the API environment is initialized, it exists until it is terminated, either by your program or by REXX. Therefore, the final call to CICSPlex SM in your program should always be an EYUTERM function. If you do not issue EYUTERM, some REXX resources, such as storage, may remain allocated and REXX becomes responsible for releasing them.

For example, sample program EYU#API1 ends like this:

```
XX = EYUAPI('TERMINATE RESPONSE(W_RESPONSE) REASON(W_REASON)')
XX = EYUTERM()
```

In this example, the EYUAPI function is used to issue an API TERMINATE command. Then EYUTERM is issued to terminate the API environment and release its allocated resources.

Using the EYUTERM function is always a good idea. However, if the CICSPlex SM host subcommand environment is actually installed at your enterprise (as opposed to being called from the function package), you may not need to use the EYUTERM function at the end of every program. Depending on the programming guidelines at your enterprise, the REXX resources that remain allocated can be reused by the next CICSPlex SM API program that accesses the host subcommand environment.

## Specifying an API command

When you write a program in REXX, you pass a character image of the command to be issued to the REXX function package supplied by CICSPlex SM. The command string can include imbedded REXX variables, as appropriate. You can specify the command in one of two ways:

- Invoke the EYUAPI function with the name of the command as its parameter.
- Use the REXX ADDRESS command to pass subsequent statements to the function package.

**Note:** You can also use the REXX PARSE VALUE command to pass API commands to the function package. However, the processing overhead of PARSE VALUE is quite high. Furthermore, the EYUAPI function returns only a single character (0 or 1), so there is no need to parse its results. For these reasons, using PARSE VALUE is not recommended.

The following example shows a partial GET command as it would be issued using the EYUAPI function:

```
var = EYUAPI('GET OBJECT(LOCTRAN)...')
```

`var` is the variable assigned to receive the return code from the EYUAPI function.

The next example shows the same GET command being issued by the REXX ADDRESS command:

```
ADDRESS CPSM 'GET OBJECT(LOCTRAN)...'
```

When the data in a REXX variable is to be passed to the function package the text portion of the API command must be terminated, the REXX variable provided, and the rest of the API command completed. The following is an example of a complete GET command that demonstrates the imbedded use of REXX variables:

```
var = EYUAPI('GET OBJECT(LOCTRAN)' ,
            'RESULT(setvar) THREAD(THRD1)' ,
            'RESPONSE(rspvar) REASON(reavar)')
```

In this example, the result set to receive the LOCTRAN objects, and the RESPONSE and REASON options are all specified as REXX variables.

Because of the way REXX handles variable substitution, you must keep in mind whether a variable is being used to send data to the API, receive data from the API, or both. The next example shows a CONNECT command where the USER and VERSION options send data to the API. The THREAD, RESPONSE, and REASON options all name variables to receive data from the API. Note that names of variables that receive data are specified as part of the command.

```
var = EYUAPI('CONNECT USER('userid') VERSION(0310)' ,
            'THREAD(thdtkn) RESPONSE(rspvar) REASON(reavar)')
```

In those cases where you want to access a resource table, special processing is required. An example of this is a FETCH command, which requires an INTO option to define where the resource table data should be placed for processing by your program. In REXX, you must specify the INTO option as the prefix of a stem variable to receive one or more resource table records. The zero entry of the stem variable indicates the number of records returned.

## Accessing resource table data

Because of the way CICSPlex SM supplies resource table data to REXX, two additional commands are provided as part of the REXX function package:

**TPARSE**
: Extracts individual resource table attributes from a record and places them into standard REXX variables. The resource table record itself can be supplied in any valid REXX variable, including a stem variable.

  You can use TPARSE to break down and access the attribute data in a resource table record.

**TBUILD**
: Builds a CPSM Definition or CICS Definition resource table record from a set of variables that you supply. Each variable must contain an individual resource table attribute.

  You can use TBUILD to build the resource table record for a definition that you want to create, update, or remove in the CICSPlex SM data repository.

**Note:** TBUILD only uses attributes that you specify; it does not assume any default values for optional attributes. If you do not supply a variable for an attribute that is optional, the corresponding field in the resource table record is initialized according to its data type (that is, character fields are set to blanks, binary data and EYUDA values are set to zeroes).

The variables that represent the resource table attributes are created either by CICSPlex SM, in the case of TPARSE, or by you, in the case of TBUILD. The variable names are formed by adding a prefix to the attribute name, like this:

```
prefix_fieldname
```

where:

**prefix** Is a text string that you supply. The maximum allowable length for a prefix is determined by REXX and the environment in which the program runs.

**fieldname**
Is the name of an attribute in the resource table.

An underscore character (_) must be inserted between the prefix and the attribute name.

When a program written in REXX passes resource table records to the API, the format and layout of the record must be exactly as it is defined by CICSPlex SM.

For complete descriptions of the TBUILD and TPARSE commands, see *CICSPlex System Manager Application Programming Reference*.

## Translating attribute values

The TBUILD and TPARSE commands use the TRANSLATE API command when processing certain resource table attributes. For example, EYUDA and CVDA values are maintained in a resource table record in their numeric form. By default, the TPARSE command converts these values into a displayable character form. TBUILD, on the other hand, converts any EYUDA or CVDA character values that you supply into their numeric equivalents.

However, if you use the ASIS option on these commands, attribute values are not converted. If you specify ASIS on the TPARSE command, you must also specify ASIS on the TBUILD command when you rebuild the record so that the API does not try to reconvert the values.

If you specify ASIS on the TPARSE command and then decide you want to convert the attribute values, you can use the TRANSLATE API command.

## Processing CHANGETIME and CREATETIME attributes

The first 8 bytes of every CPSM Definition and CICS Definition resource table record contain an attribute called CHANGETIME, which reflects the date and time at which the record was last modified. CICS Definition records also include a CREATETIME attribute, which is the date and time at which the definition was created. The CHANGETIME and CREATETIME attributes are maintained internally by CICSPlex SM; you should not attempt to modify these attribute values. When you update a resource table record, the CHANGETIME and CREATETIME values you pass to the TBUILD command must be the same values you received from TPARSE.

By default, the TPARSE command translates the CHANGETIME and CREATETIME values into displayable, character values. However, the character forms of these values cannot be passed back to TBUILD. So, if you plan to update a definition and then rebuild the resource table record, you should use the ASIS option on the TPARSE and TBUILD commands. When you use ASIS, the CHANGETIME and CREATETIME values appear as 16-byte hexadecimal values.

## Processing FEEDBACK attributes

Having used a TPARSE command to extract the individual resource table attributes additional processing may be required before the data can be used in subsequent API commands.

The ERR_RESULT error result set token is returned in decimal format and must be converted to character format before it can be used in a RESULT() option. To do this you can use the D2X() and X2C() REXX built-in functions, for example:

```
var = X2C(RIGHT(D2X(FEEDBACK_ERR_RESULT),8,'0'))
```

# Chapter 6. REXX error handling

## Translation errors

Errors that occur while REXX is trying to interpret a CICSPlex SM API command result in a REXX return code. If REXX cannot process a command string or function, the run-time interface sets the REXX return code in one of two places:

**RC variable**
When the ADDRESS CPSM command is used.

The return code value is one of the following:

**0** The command was successfully processed.

**8** The command contained syntax errors that prevented REXX from processing it. EYUARnnnn messages that describe the error are written to the destination defined on your system for IRXSAY WRITEERR output.

**16** The command could not be processed because of some system failure (such as a lack of storage). REXX messages that describe the error may be produced.

**–3** The CICSPlex SM API environment is not available. This condition can occur if the function package is not properly installed. If the function package is installed, it could mean that you did not issue at least one EYUxxxx REXX function before invoking the ADDRESS CPSM command.

**Function variable**
When an EYUxxxx REXX function is used.

For most EYUxxxx functions, the return code value is one of the following:

**0** The function was successfully processed.

**1** The function failed. EYUARnnnn messages that describe the error are written to the destination defined on your system for IRXSAY WRITEERR output.

For the EYURESP and EYUREAS functions, the return code is either the numeric equivalent of the value being translated or –1, if the translation failed.

In general, if the REXX return code is anything other than:

**0** From EYUAPI, EYUINIT, or EYUTERM

**A valid RESPONSE or REASON value**
From EYURESP or EYUREAS

the API command was not successfully interpreted by REXX and, therefore, was not passed to CICSPlex SM for processing. If a command is not processed, the RESPONSE and REASON values are not set and you do not need to check them.

If the return code is 0, the API command was interpreted by REXX and passed to CICSPlex SM. Note that a return code of 0 does not indicate whether the command was successfully processed by CICSPlex SM. To determine the results of an API command, refer to the RESPONSE and REASON values returned by the command.

## Run-time errors

Errors that occur while CICSPlex SM is trying to process an API command are reported by the RESPONSE and REASON values for the command. For more information, see “Using the RESPONSE and REASON options” on page 87.

## TPARSE and TBUILD errors

The results of a TPARSE or TBUILD command are returned by the STATUS option, which is a required option on those commands. The STATUS option serves a similar purpose to the RESPONSE and REASON options on other API commands.

The STATUS option returns the REXX status value in character form as one of the following:

**OK** The command completed processing successfully.

**SYNTAX ERROR**
The command could not be processed because of a syntax error. EYUARnnnn messages that describe the error are written to the destination defined on your system for IRXSAY WRITEERR output.

**FAILURE**
The command failed because some of the data it was attempting to process is invalid. Trace data is written to a REXX stem variable called EYUTRACE. EYUARnnnn messages that describe the failure may also be written to the destination defined on your system for IRXSAY WRITEERR output.

For more information about the EYUTRACE stem variable, see “EYU_TRACE data.”

## Messages

Many of the error conditions you might encounter when using the REXX run-time interface are accompanied by messages that describe the error. These messages, which begin with the prefix EYUARnnnn, are written to the destination defined on your system for IRXSAY WRITEERR output. By default, such output goes to one of the following places:

- For a program running in TSO foreground, the output goes to the terminal.
- For a program running in background, the output goes to the SYSTSPRT DD destination.

## EYU_TRACE data

The run-time interface creates a REXX stem variable called EYU_TRACE anytime an error occurs that warrants tracing. Such conditions include:

- A STATUS of FAILURE from a TBUILD or TPARSE command
- A return code other than 0 from an EYUxxxx function.

The zero entry of the stem array indicates the number of trace records that were produced. Entries 1 through n contain the actual trace records.

If you are having problems with a REXX program or the run-time interface, IBM support may request the trace records from EYU_TRACE. CICSPlex SM distributes a REXX EXEC that IBM support will ask you to include in your REXX program to format and print the EYU_TRACE records. The formatting routine is called EYU#TRCF and is distributed in the SEYUCLIB library. EYU#TRCF should be used

only at the request of IBM support.

# Appendix A. BINCONRS, BINCONSC, and BINSTERR error codes

This appendix contains the error codes in the BINCONRS, BINCONSC, and BINSTERR copy books. See “Retrieving FEEDBACK records” on page 91 for information on interpreting feedback error result sets containing these error codes.

## BINCONRS

*Table 9. BINCONRS error codes–ERROP field*

| Value | Code | Reason |
|---|---|---|
| 01 | ADDSYS | Add System to System Group |
| 02 | ADDTOGRP | Add Definition to Group |
| 03 | UPDINGRP | Update Definition in Group |
| 04 | ADDRASI | Add RASINDSC |
| 05 | ADDRESI | Add RESINDSC |
| 06 | UPDRASG | Update RASGNDEF |
| 07 | UPDRASI | Update RASINDSC |
| 08 | UPDRESI | Update RESDESC Install Scope |
| 09 | UPDRDSC | Update RESDESC |

## BINCONSC

*Table 10. BINCONSC error codes–ERROP field*

| Value | Code | Reason |
|---|---|---|
| 01 | ADDRASI | Add RASINDSC |
| 02 | UPDRASG | Update RASGNDEF |
| 03 | UPDRASI | Update RASINDSC |
| 04 | UPDRDSC | Update RESDESC |
| 05 | ADDSYS | Add System to Group |

*Table 11. BINCONSC error codes–ERRCODE field*

| Value | Code | Reason |
|---|---|---|
| 01 | SAMESCP | Target/Related scopes are same |
| 02 | TRGINREL | Target Scope is in Related |
| 03 | RELINTRG | Related Scope is in Target |
| 04 | SYSNBOTH | CICSNAME in Target and Related |
| 05 | MULTREL | Multiple Systems in Related |
| 06 | RELNOSYS | Related System has no SYSID |

# BINSTERR

*Table 12. BINSTERR error codes–ERRCODE field*

| Value | Code | Reason |
|---|---|---|
| 01 | SYSSTATE | System inactive/not create capable |
| 02 | INSTNAUT | Install not authorized |
| 03 | DSCDNAUT | Discard not authorized |
| 04 | INSTFAIL | Install failure |
| 05 | DSCDFAIL | Install discard failure |
| 06 | INSTCPFL | Install Complete failure |
| 07 | INSTNCON | Install Connection failure |
| 08 | INSTSTAT | Install status failure |
| 09 | INSTNSUP | Install not supported |
| 10 | FORCENO | Resource Install negated |
| 11 | DSCRDERR | Discard failure |
| 12 | METHFAIL | MAS method failure |
| 13 | NOCREATE | System not create capable |

# Appendix B. Sample program listings

This appendix provides listings for the sample programs that are distributed with CICSPlex SM. Each sample program is shown here in one of the languages in which it is distributed. For a list of the sample programs provided in each language and the libraries where they are distributed, see Table 6 on page 11.

**Note:** Additional sample CICSPlex SM API programs are available via the IBM CICS SupportPacs system at:

`http://www.ibm.com/software/htp/cics/downloads`

## Sample program EYU#API1

Program EYU#API1 is written in REXX for the TSO environment.

**EYUxAPI1**

This program does the following:

- Establishes a connection to the API.
- Creates a result set containing all PROGRAM resource table records that do not begin with DFH, EYU, or IBM.
- Retrieves each record in the result set.
- Translates any CICS CVDA attributes into meaningful character values.
- Displays each record on the terminal, showing the program name, language, enable status, and CEDF status.
- Terminates the API connection.

**Commands Used:** CONNECT, FETCH, GET, TERMINATE, TRANSLATE

## sample program EYU#API1

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* MODULE NAME = EYU#API1                                            */
/*                                                                   */
/* DESCRIPTIVE NAME = CPSM Sample API Program 1                      */
/*                    (Sample REXX Version)                          */
/*                                                                   */
/*    5695-081                                                       */
/*    COPYRIGHT = NONE                                               */
/*                                                                   */
/* STATUS = %CP00                                                    */
/*                                                                   */
/* FUNCTION =                                                        */
/*                                                                   */
/*  To provide an example of the use of the following EXEC CPSM      */
/*  commands: CONNECT, GET, FETCH, TRANSLATE, TERMINATE.             */
/*                                                                   */
/*  When invoked, the program depends upon the values held in the    */
/*  W_CONTEXT and W_SCOPE declarations when establishing a           */
/*  connection with CICSPlex SM. They must take the following        */
/*  values:                                                          */
/*                                                                   */
/*  W_CONTEXT  = The name of a CMAS or CICSplex. Refer to the        */
/*               description of the EXEC CPSM CONNECT command        */
/*               for further information regarding the CONTEXT       */
/*               option.                                             */
/*                                                                   */
/*  W_SCOPE    = The name of a CICSplex, CICS system, or CICS        */
/*               system group within the CICSplex. Refer to the      */
/*               description of the EXEC CPSM CONNECT command        */
/*               for further information regarding the SCOPE         */
/*               option.                                             */
/*                                                                   */
/*  This sample requires no parameters at invocation time.           */
/*                                                                   */
/*  The sample establishes an API connection and issues a GET        */
/*  command to create a result set containing program resource       */
/*  table records which match the criteria.                          */
/*                                                                   */
/*  Using the FETCH command each record in the result set is         */
/*  retrieved. Once retrieved the TRANSLATE command is used to       */
/*  convert those attributes of each record which are EYUDA or       */
/*  CVDA values into meaningful character representations. A          */
/*  record is then displayed on the terminal showing the program     */
/*  name, language, program status, and CEDF status.                 */
/*                                                                   */
/*  Finally, the API connection is terminated.                       */
/*                                                                   */
/*-------------------------------------------------------------------*/
/*NOTES :                                                            */
/*  DEPENDENCIES = S/390, TSO                                        */
/*  RESTRICTIONS = None                                              */
/*  REGISTER CONVENTIONS =                                           */
/*  MODULE TYPE  = Executable                                        */
/*  PROCESSOR    = REXX                                              */
/*  ATTRIBUTES   = Read only, Serially Reusable                      */
/*                                                                   */
```

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/*ENTRY POINT = EYU#API1                                             */
/*                                                                   */
/*   PURPOSE = All Functions                                         */
/*                                                                   */
/*   LINKAGE = From TSO as a REXX EXEC.                              */
/*                                                                   */
/*   INPUT   = None.                                                 */
/*                                                                   */
/*-------------------------------------------------------------------*/
/*                                                                   */
Address 'TSO'
Parse Value 0 0 With W_RESPONSE W_REASON .
/*-------------------------------------------------------------------*/
/*    CHANGE W_CONTEXT AND W_SCOPE TO MATCH YOUR INSTALLATION       */
/*-------------------------------------------------------------------*/
W_CONTEXT = 'RTGA'
W_SCOPE = 'RTGA'
/*-------------------------------------------------------------------*/
/*    OBTAIN A CPSM API CONNECTION.                                  */
/*                                                                   */
/*    THE API WILL RETURN A TOKEN IDENTIFYING THE THREAD IN          */
/*    VARIABLE W_THREAD.                                             */
/*-------------------------------------------------------------------*/
Say 'Initializing API...'
XX = EYUINIT()
If XX <> 0 Then Signal UNEXPECTED
Say 'Establishing connection...'
XX = EYUAPI('CONNECT' ,
            'CONTEXT('W_CONTEXT')' ,
            'SCOPE('W_SCOPE')' ,
            'VERSION(0310)' ,
            'THREAD(W_THREAD)' ,
            'RESPONSE(W_RESPONSE)' ,
            'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE <> EYURESP(OK) Then Signal NO_CONNECT
/*-------------------------------------------------------------------*/
/*    GET THE PROGRAM RESOURCE TABLE.                                */
/*                                                                   */
/*    CREATE A RESULT SET CONTAINING ENTRIES FOR ALL PROGRAMS        */
/*    WITH NAMES NOT BEGINNING DFH, EYU or IBM.                      */
/*    THE NUMBER OF ENTRIES MEETING THE CRITERIA IS RETURNED IN      */
/*    VARIABLE W_RECCNT.                                             */
/*-------------------------------------------------------------------*/
Say 'Get the PROGRAM resource table...'
W_CRITERIA = 'NOT (PROGRAM=DFH* OR PROGRAM=EYU* OR PROGRAM=IBM*).'
W_CRITERIALEN = 'LENGTH'(W_CRITERIA)
XX = EYUAPI('GET OBJECT(PROGRAM)' ,
                'CRITERIA(W_CRITERIA)' ,
                'LENGTH('W_CRITERIALEN')' ,
                'COUNT(W_RECCNT)' ,
                'RESULT(W_RESULT)' ,
                'THREAD(W_THREAD)' ,
                'RESPONSE(W_RESPONSE)' ,
                'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE <> EYURESP(OK) Then Signal NO_GET
```

```
/*-------------------------------------------------------------------*/
/*    RETRIEVE INFORMATION ABOUT EACH PROGRAM.                       */
/*                                                                   */
/*    FETCH EACH ENTRY AND USE TPARSE TO OBTAIN EACH ATTRIBUTE.      */
/*    DISPLAY DETAILS OF EACH PROGRAM TO THE USER.                   */
/*-------------------------------------------------------------------*/
Say 'Fetching' W_RECCNT 'PROGRAM entries...'
Say 'Program  Language     Status       CEDF Status'
W_INTO_OBJECTLEN = 136                   /* LENGTH OF PROGRAM TABLE */
Do III = 1 To W_RECCNT
  XX = EYUAPI('FETCH INTO(W_INTO_OBJECT)' ,
                    'LENGTH(W_INTO_OBJECTLEN)' ,
                    'RESULT(W_RESULT)' ,
                    'THREAD(W_THREAD)' ,
                    'RESPONSE(W_RESPONSE)' ,
                    'REASON(W_REASON)')
  If XX <> 0 Then Signal UNEXPECTED
  If W_RESPONSE <> EYURESP(OK) Then Signal NO_FETCH
  XX = EYUAPI('TPARSE OBJECT(PROGRAM)' ,
                     'PREFIX(PGM)' ,
                     'STATUS(W_RESPONSE)' ,
                     'VAR(W_INTO_OBJECT.1)' ,
                     'THREAD(W_THREAD)')
  If W_RESPONSE <> 'OK' Then Signal UNEXPECTED
  W_TEXT = PGM_PROGRAM
  W_TEXT = 'OVERLAY'(PGM_LANGUAGE,W_TEXT,10)
  W_TEXT = 'OVERLAY'(PGM_STATUS,W_TEXT,23)
  W_TEXT = 'OVERLAY'(PGM_CEDFSTATUS,W_TEXT,36)
  Say W_TEXT
End III
Signal ENDIT
/*-------------------------------------------------------------------*/
/*    PROCESSING FOR API FAILURES.                                   */
/*-------------------------------------------------------------------*/
UNEXPECTED:
      W_MSG_TEXT = 'UNEXPECTED ERROR.'
      Signal SCRNLOG
NO_CONNECT:
      W_MSG_TEXT = 'ERROR CONNECTING TO API.'
      Signal SCRNLOG
NO_GET:
      W_MSG_TEXT = 'ERROR GETTING RESOURCE TABLE.'
      Signal SCRNLOG
NO_FETCH:
      W_MSG_TEXT = 'ERROR FETCHING RESULT SET.'
      Signal SCRNLOG
SCRNLOG:
      Say W_MSG_TEXT
      Say 'RESPONSE='||W_RESPONSE ,
          'REASON='||W_REASON 'RESULT='XX
ENDIT:
/*-------------------------------------------------------------------*/
/*    TERMINATE API CONNECTION.                                      */
/*-------------------------------------------------------------------*/
XX = EYUAPI('TERMINATE RESPONSE(W_RESPONSE) REASON(W_REASON)')
XX = EYUTERM()
Exit
```

The C/370™, COBOL and PL/1 versions of EYUxAPI1 are written for the CICS environment and can be converted to run in the MVS/ESA batch environment by commenting the EXEC CICS SEND commands, and uncommenting the preceeding language specific output statements.

# Sample program EYUCAPI2

Program EYUCAPI2 is written in C for the CICS ENVIRONMENT.

**EYUxAPI2**

This program does the following:

- Establishes a connection to the API.
- Defines a filter to identify PROGRAM resource table records with a language attribute of Assembler.
- Creates a result set containing all PROGRAM resource table records that do not begin with DFH, EYU, or IBM.
- Marks those records in the result set that match the specified filter (LANGUAGE=ASSEMBLER).
- Copies the marked records to a new result set.
- Deletes the marked records from the original result set.
- For each result set (LANGUAGE=ASSEMBLER and LANGUAGE≠ASSEMBLER):
  - Retrieves each record.
  - Translates any CICS CVDA attributes.
  - Displays each record on the terminal.
- Terminates the API connection.

**Commands Used:** CONNECT, COPY, DELETE, FETCH, GET, LOCATE, MARK, SPECIFY FILTER, TERMINATE, TRANSLATE

## sample program EYUCAPI2

```
/*****************************************************************/
/*                                                               */
/* MODULE NAME = EYUCAPI2                                        */
/*                                                               */
/* DESCRIPTIVE NAME = CPSM Sample API Program 2                  */
/*                    (Sample C Version)                         */
/*                                                               */
/*    5695-081                                                   */
/*    COPYRIGHT = NONE                                           */
/*                                                               */
/* STATUS = %CP00                                                */
/*                                                               */
/* FUNCTION =                                                    */
/*                                                               */
/*  To provide an example of the use of the following EXEC CPSM  */
/*  commands: CONNECT, SPECIFY FILTER, GET, MARK, COPY, DELETE,  */
/*  LOCATE, FETCH, TRANSLATE, TERMINATE.                         */
/*                                                               */
/*  When invoked, the program depends upon the values held in the*/
/*  W_CONTEXT and W_SCOPE declarations when establishing a       */
/*  connection with CICSPlex SM. They must take the following    */
/*  values:                                                      */
/*                                                               */
/*  W_CONTEXT  = The name of a CMAS or CICSplex. Refer to the    */
/*               description of the EXEC CPSM CONNECT command    */
/*               for further information regarding the CONTEXT   */
/*               option.                                         */
/*                                                               */
/*  W_SCOPE    = The name of a CICSplex, CICS system, or CICS     */
/*               system group within the CICSplex. Refer to the  */
/*               description of the EXEC CPSM CONNECT command    */
/*               for further information regarding the SCOPE     */
/*               option.                                         */
/*                                                               */
/*  This sample requires no parameters at invocation time.       */
/*                                                               */
/*  The sample establishes an API connection and issues a SPECIFY*/
/*  FILTER command to create a filter which will match only      */
/*  specific program resource table records. The filter is used  */
/*  later in the program by the MARK command.                    */
/*                                                               */
/*  A GET command is issued to create a result set containing    */
/*  program resource table records which match the criteria. The */
/*  result set is then used by the MARK command to flag records  */
/*  meeting the previous filter specification. The marked records*/
/*  are then COPYed to a new result set, and then DELETEd from   */
/*  the original result set. After this sequence of commands we  */
/*  have two results sets; one containing records which did not  */
/*  meet the filter specification (that is, records where the    */
/*  LANGUAGE is not ASSEMBLER), and one containing records       */
/*  which did match the filter (that is, records where the       */
/*  LANGUAGE is ASSEMBLER).                                      */
/*                                                               */
/*  Taking each of the two results sets in turn a LOCATE command */
/*  is used to ensure we start at the top of the result set      */
/*  before a FETCH command is used to retrieve each record in    */
/*  the result set. Once retrieved the TRANSLATE command is used */
/*  to convert those attributes of each record which are EYUDA   */
/*  or CVDA values into meaningful character representations. A  */
/*  record is then displayed on the terminal showing the program */
/*  name, language, program status, and CEDF status.             */
/*                                                               */
/*  Finally, the API connection is terminated.                   */
/*                                                               */
```

```
* -------------------------------------------------------------------*
 /*NOTES :                                                           */
 /*  DEPENDENCIES = S/390, CICS                                      */
 /*  RESTRICTIONS = None                                             */
 /*  REGISTER CONVENTIONS =                                          */
 /*  MODULE TYPE  = Executable                                       */
 /*  PROCESSOR    = C                                                */
 /*  ATTRIBUTES   = Read only, Serially Reusable                     */
 /*                                                                  */
 /*------------------------------------------------------------------*/
 /*                                                                  */
 /*ENTRY POINT = EYUCAPI2                                            */
 /*                                                                  */
 /*   PURPOSE = All Functions                                        */
 /*                                                                  */
 /*   LINKAGE = From CICS either with EXEC CICS LINK or as a CICS    */
 /*             transaction.                                         */
 /*                                                                  */
 /*   INPUT   = None.                                                */
 /*                                                                  */
 /*------------------------------------------------------------------*/
 /*                                                                  */
#include <PROGRAM>
void main()
{
/*-------------------------------------------------------------------*/
/*    CHANGE W_CONTEXT AND W_SCOPE TO MATCH YOUR INSTALLATION       */
/*-------------------------------------------------------------------*/
char    *W_CONTEXT       = "RTGA    ";
char    *W_SCOPE         = "RTGA    ";
int      W_RESPONSE;
int      W_REASON;
int      W_THREAD;
char    *W_CRITERIA;
int      W_CRITERIALEN;
int      W_FILTER_TOKEN;
int      W_RESULT        = 0;
int      W_COUNT;
int      W_RESULT2       = 0;
int      W_COUNT2;
int      III;
int      JJJ;
int      W_RESULT_TOK;
int      W_RECCNT;
PROGRAM  W_INTO_OBJECT;
int      W_INTO_OBJECTLEN;
char     W_TRANSCVDA??(12??);
char     W_TEXT??(81??);
char     W_MSG_TEXT??(81??);
W_TEXT??(80??) = 0x13;
W_MSG_TEXT??(80??) = 0x13;
/*-------------------------------------------------------------------*/
/*    OBTAIN A CPSM API CONNECTION.                                 */
/*                                                                  */
/*    THE API WILL RETURN A TOKEN IDENTIFYING THE THREAD IN         */
/*    VARIABLE W_THREAD.                                            */
/*-------------------------------------------------------------------*/
strcpy(W_TEXT,"Establishing connection...");
/* printf("Establishing connection...\n"); */
EXEC CICS SEND FROM(W_TEXT) LENGTH(81) ERASE;
EXEC CPSM CONNECT
          CONTEXT(W_CONTEXT)
          SCOPE(W_SCOPE)
          VERSION("0310")
          THREAD(W_THREAD)
          RESPONSE(W_RESPONSE)
          REASON(W_REASON) ;
if (W_RESPONSE != EYUVALUE(OK)) { goto NO_CONNECT; }
```

```
/*-------------------------------------------------------------------*/
/*    CREATE A FILTER.                                               */
/*                                                                   */
/*    CREATE A FILTER WHICH WILL MATCH ONLY THOSE PROGRAMS WITH      */
/*    A LANGUAGE OF ASSEMBLER.                                       */
/*    THE FILTER WILL BE USED IN A SUBSEQUENT MARK COMMAND.          */
/*-------------------------------------------------------------------*/
strcpy(W_TEXT,"Create a filter...        ");
/* printf("Create a filter...\n"); */
EXEC CICS SEND FROM(W_TEXT) LENGTH(81) WAIT;
W_CRITERIA = "LANGUAGE=ASSEMBLER.";
W_CRITERIALEN = strlen(W_CRITERIA);
EXEC CPSM SPECIFY FILTER(W_FILTER_TOKEN)
          CRITERIA(W_CRITERIA)
          LENGTH(W_CRITERIALEN)
          OBJECT("PROGRAM ")
          THREAD(W_THREAD)
          RESPONSE(W_RESPONSE)
          REASON(W_REASON) ;
if (W_RESPONSE != EYUVALUE(OK)) { goto NO_FILTER; }


/*-------------------------------------------------------------------*/
/*    GET THE PROGRAM RESOURCE TABLE.                                */
/*                                                                   */
/*    CREATE A RESULT SET CONTAINING ENTRIES FOR ALL PROGRAMS        */
/*    WITH NAMES NOT BEGINNING DFH, EYU OR IBM.                      */
/*    THE NUMBER OF ENTRIES MEETING THE CRITERIA IS RETURNED IN      */
/*    VARIABLE W_COUNT.                                              */
/*-------------------------------------------------------------------*/
strcpy(W_TEXT,"Get the PROGRAM resource table...");
/* printf("Get the PROGRAM resource table...\n"); */
EXEC CICS SEND FROM(W_TEXT) LENGTH(81) WAIT;
W_CRITERIA = "NOT (PROGRAM=DFH* OR PROGRAM=EYU* OR PROGRAM=IBM*).";
W_CRITERIALEN = strlen(W_CRITERIA);
EXEC CPSM GET OBJECT("PROGRAM ")
          CRITERIA(W_CRITERIA)
          LENGTH(W_CRITERIALEN)
          COUNT(W_COUNT)
          RESULT(W_RESULT)
          THREAD(W_THREAD)
          RESPONSE(W_RESPONSE)
          REASON(W_REASON) ;
if (W_RESPONSE != EYUVALUE(OK)) { goto NO_GET; }
sprintf(W_TEXT,"Total number of entries: %d", W_COUNT);
/* printf(W_TEXT); */
EXEC CICS SEND FROM(W_TEXT) LENGTH(81) WAIT;
/*-------------------------------------------------------------------*/
/*    MARK SELECTED PROGRAM ENTRIES.                                 */
/*                                                                   */
/*    USING THE FILTER WE MARK THOSE ENTRIES IN THE RESULT SET       */
/*    WHICH MEET THE FILTER SPECIFICATION IE. THOSE ENTRIES WITH     */
/*    A LANGUAGE OF ASSEMBLER.                                       */
/*-------------------------------------------------------------------*/
strcpy(W_TEXT,"Mark LANGUAGE=ASSEMBLER entries...");
/* printf("Mark LANGUAGE=ASSEMBLER entries...\n"); */
EXEC CICS SEND FROM(W_TEXT) LENGTH(81) WAIT;
EXEC CPSM MARK FILTER(W_FILTER_TOKEN)
          RESULT(W_RESULT)
          THREAD(W_THREAD)
          RESPONSE(W_RESPONSE)
          REASON(W_REASON) ;
if (W_RESPONSE != EYUVALUE(OK)) { goto NO_MARK; }
```

```
/*-------------------------------------------------------------------*/
/*    COPY MARKED ENTRIES TO ANOTHER RESULT SET.                     */
/*                                                                   */
/*    HAVING MARKED ENTRIES IN THE RESULT SET WE CAN COPY THEM       */
/*    TO A NEW RESULT SET.                                           */
/*    AFTER THIS COMMAND WE WILL HAVE TWO RESULT SETS. ONE           */
/*    CONTAINING ALL THE PROGRAM ENTRIES, AND THE OTHER CONTAINING   */
/*    JUST THOSE ENTRIES WITH A LANGUAGE OF ASSEMBLER.               */
/*-------------------------------------------------------------------*/
strcpy(W_TEXT,"Copy marked entries...       ");
/* printf("Copy marked entries...\n"); */
EXEC CICS SEND FROM(W_TEXT) LENGTH(81) WAIT;
EXEC CPSM COPY FROM(W_RESULT)
          TO(W_RESULT2)
          MARKED
          COUNT(W_COUNT2)
          THREAD(W_THREAD)
          RESPONSE(W_RESPONSE)
          REASON(W_REASON) ;
if (W_RESPONSE != EYUVALUE(OK)) { goto NO_COPY; }
sprintf(W_TEXT,"Number of entries copied: %d", W_COUNT2);
/* printf(W_TEXT); */
EXEC CICS SEND FROM(W_TEXT) LENGTH(81) WAIT;
/*-------------------------------------------------------------------*/
/*    DELETE MARKED ENTRIES FROM RESULT SET.                         */
/*                                                                   */
/*    WE CAN NOW DELETE THE MARKED ENTRIES FROM THE ORIGINAL         */
/*    RESULT SET.                                                    */
/*    AFTER THIS COMMAND WE HAVE TWO RESULT SETS. ONE RESULT SET     */
/*    CONTAINING ENTRIES WITH LANGUAGE NOT ASSEMBLER, AND THE        */
/*    OTHER CONTAINING ENTRIES WITH A LANGUAGE OF ASSEMBLER.         */
/*-------------------------------------------------------------------*/
strcpy(W_TEXT,"Delete marked entries...     ");
/* printf("Delete marked entries...\n"); */
EXEC CICS SEND FROM(W_TEXT) LENGTH(81) WAIT;
EXEC CPSM DELETE MARKED
          COUNT(W_COUNT)
          RESULT(W_RESULT)
          THREAD(W_THREAD)
          RESPONSE(W_RESPONSE)
          REASON(W_REASON) ;
if (W_RESPONSE != EYUVALUE(OK)) { goto NO_DELETE; }
sprintf(W_TEXT,"Number of entries remaining: %d", W_COUNT);
/* printf(W_TEXT); */
EXEC CICS SEND FROM(W_TEXT) LENGTH(81) WAIT;
```

```
/*-------------------------------------------------------------------*/
/*    RETRIEVE INFORMATION ABOUT EACH PROGRAM.                       */
/*                                                                   */
/*    FETCH EACH ENTRY, USE INCLUDED STRUCTURE TO OBTAIN EACH        */
/*    ATTRIBUTE AND USE TRANSLATE TO CONVERT CICS CVDAS.             */
/*    DISPLAY DETAILS OF EACH PROGRAM TO THE USER.                   */
/*-------------------------------------------------------------------*/
W_INTO_OBJECTLEN = PROGRAM_TBL_LEN;
for (JJJ = 1; JJJ <= 2; JJJ++)
 {
   if (JJJ == 1)
    {
      sprintf(W_TEXT,"Fetching %d non-ASSEMBLER PROGRAM entries...\n",
              W_COUNT);
      W_RESULT_TOK = W_RESULT;
      W_RECCNT = W_COUNT;
    }
   else
     {
      sprintf(W_TEXT,"Fetching %d ASSEMBLER PROGRAM entries...\n",
              W_COUNT2);
      W_RESULT_TOK = W_RESULT2;
      W_RECCNT = W_COUNT2;
     }
   /* printf(W_TEXT); */
   EXEC CICS SEND FROM(W_TEXT) LENGTH(81) WAIT;
   EXEC CPSM LOCATE TOP
             RESULT(W_RESULT_TOK)
             THREAD(W_THREAD)
             RESPONSE(W_RESPONSE)
             REASON(W_REASON) ;
   if (W_RESPONSE != EYUVALUE(OK)) { goto NO_LOCATE; }
   strcpy(W_TEXT,"Program  Language     Status       CEDF Status");
   /* printf("Program  Language     Status       CEDF Status\n"); */
   EXEC CICS SEND FROM(W_TEXT) LENGTH(81) WAIT;
   for (III = 1; III <= W_RECCNT; III++)
    {
      EXEC CPSM FETCH INTO(&W_INTO_OBJECT)
                LENGTH(W_INTO_OBJECTLEN)
                RESULT(W_RESULT_TOK)
                THREAD(W_THREAD)
                RESPONSE(W_RESPONSE)
                REASON(W_REASON) ;
      if (W_RESPONSE != EYUVALUE(OK)) { goto NO_FETCH; }
      memcpy(W_TEXT,W_INTO_OBJECT.PROGRAM,8);
      EXEC CPSM TRANSLATE OBJECT("PROGRAM ")
                ATTRIBUTE("LANGUAGE    ")
                FROMCV(W_INTO_OBJECT.LANGUAGE)
                TOCHAR(W_TRANSCVDA)
                THREAD(W_THREAD)
                RESPONSE(W_RESPONSE)
                REASON(W_REASON) ;
      if (W_RESPONSE != EYUVALUE(OK)) { goto NO_TRANSLATE; }
      memcpy(W_TEXT+9,W_TRANSCVDA,12);
      EXEC CPSM TRANSLATE OBJECT("PROGRAM ")
                ATTRIBUTE("STATUS      ")
                FROMCV(W_INTO_OBJECT.STATUS)
                TOCHAR(W_TRANSCVDA)
                THREAD(W_THREAD)
                RESPONSE(W_RESPONSE)
                REASON(W_REASON) ;
```

```
      if (W_RESPONSE != EYUVALUE(OK)) { goto NO_TRANSLATE; }
      memcpy(W_TEXT+22,W_TRANSCVDA,12);
      EXEC CPSM TRANSLATE OBJECT("PROGRAM ")
                ATTRIBUTE("CEDFSTATUS  ")
                FROMCV(W_INTO_OBJECT.CEDFSTATUS)
                TOCHAR(W_TRANSCVDA)
                THREAD(W_THREAD)
                RESPONSE(W_RESPONSE)
                REASON(W_REASON) ;
      if (W_RESPONSE != EYUVALUE(OK)) { goto NO_TRANSLATE; }
      memcpy(W_TEXT+35,W_TRANSCVDA,12);
      /* printf("%s\n",W_TEXT); */
      EXEC CICS SEND FROM(W_TEXT) LENGTH(81) WAIT;
    }
 }
goto ENDIT;
/*-------------------------------------------------------------------*/
/*    PROCESSING FOR API FAILURES.                                   */
/*-------------------------------------------------------------------*/
NO_CONNECT:
   strcpy(W_MSG_TEXT,"ERROR CONNECTING TO API.\n");
   goto SCRNLOG;
NO_FILTER:
   strcpy(W_MSG_TEXT,"ERROR CREATING FILTER.\n");
   goto SCRNLOG;
NO_GET:
   strcpy(W_MSG_TEXT,"ERROR GETTING RESOURCE TABLE.\n");
   goto SCRNLOG;
NO_MARK:
   strcpy(W_MSG_TEXT,"ERROR MARKING RESULT SET.\n");
   goto SCRNLOG;
NO_COPY:
   strcpy(W_MSG_TEXT,"ERROR COPYING RESULT SET.\n");
   goto SCRNLOG;
NO_DELETE:
   strcpy(W_MSG_TEXT,"ERROR DELETING FROM RESULT SET.\n");
   goto SCRNLOG;
NO_LOCATE:
   strcpy(W_MSG_TEXT,"ERROR LOCATING TO TOP OF RESULT SET.\n");
   goto SCRNLOG;
NO_FETCH:
   strcpy(W_MSG_TEXT,"ERROR FETCHING RESULT SET.\n");
   goto SCRNLOG;
NO_TRANSLATE:
   strcpy(W_MSG_TEXT,"ERROR TRANSLATING ATTRIBUTE\n");
   goto SCRNLOG;
SCRNLOG:
   /* printf(W_MSG_TEXT); */
   EXEC CICS SEND FROM(W_MSG_TEXT) LENGTH(81) WAIT;
   sprintf(W_MSG_TEXT,"RESPONSE=%d REASON=%d\n",W_RESPONSE,W_REASON);
   /* printf(W_MSG_TEXT); */
   EXEC CICS SEND FROM(W_MSG_TEXT) LENGTH(81) WAIT;
ENDIT:
/*-------------------------------------------------------------------*/
/*    TERMINATE API CONNECTION.                                      */
/*-------------------------------------------------------------------*/
EXEC CPSM TERMINATE RESPONSE(W_RESPONSE) REASON(W_REASON);
EXEC CICS RETURN;
}
```

The C/370, COBOL and PL/1 versions of EYUxAPI2 are written for the CICS environment and can be converted to run in the MVS/ESA batch environment by commenting the EXEC CICS SEND commands, and uncommenting the preceeding language specific output statements.

# Sample program EYUAAPI3

Program EYUAAPI3 is written in Assembler for the MVS/ESA batch environment.

**EYUxAPI3**

This program does the following:

- Establishes a connection to the API with the context set to an existing CICSplex.
- Verifies that a proposed new CICSplex name is not already defined to CICSPlex SM as a CICSplex, CMAS, CICS system, or CICS system group.
- Creates a result set containing the CPLEXDEF resource table record for the existing CICSplex definition and retrieves that record.
- Creates a new CPLEXDEF resource table record using the existing record as a model.
- Creates a result set containing the CICSPLEX resource table records associated with the existing CICSplex and retrieves those records.
- Creates new CICSPLEX resource table records using the existing records as models.
- Sequentially retrieves all the resource table records associated with the existing CICSplex, including CICS systems, CICS system groups, workload management definitions, real-time analysis definitions, and resource monitoring definitions.
- Creates all the necessary resource table records for the new CICSplex using the existing records as models.
- If an error occurs before all the necessary resource table records are created, removes the new CICSplex definition.
- Disconnects the API processing thread.

**Commands Used:** CONNECT, CREATE, DISCARD, DISCONNECT, FETCH, GET, PERFORM OBJECT, QUALIFY, QUERY, REMOVE

```
*
EYUAAPI3 TITLE 'EYUAAPI3 - CPSM SAMPLE API PROGRAM 3 - ASSEMBLER'
*********************************************************************
*                                                                   *
* MODULE NAME = EYUAAPI3                                            *
*                                                                   *
* DESCRIPTIVE NAME = API sample program 3 ASSEMBLER Version         *
*                                                                   *
*        5695-081                                                   *
*        COPYRIGHT = NONE                                           *
*                                                                   *
* STATUS = %CP00                                                    *
*                                                                   *
* FUNCTION =                                                        *
*                                                                   *
*    To mirror an existing PLEX to a new PLEX.                      *
*                                                                   *
*    When invoked, the program depends upon the values held in the  *
*    OLDPLEX, NEWPLEX, and MPCMAS variables.  They must be set to   *
*    the following values:                                          *
*                                                                   *
*    OLDPLEX    = The name of an existing PLEX that will be mirrored. *
*                                                                   *
*    NEWPLEX    = The name that will be given to the new PLEX.      *
*                                                                   *
*    MPCMAS     = The maintenance point CMAS of the OLDPLEX.  This  *
*                 will also be the MP for the NEWPLEX.              *
*                                                                   *
*    This sample requires no parameters at invocation time.         *
*                                                                   *
*    The sample processes as follows:                               *
*                                                                   *
*    -  a CONNECTion is established to CPSM, with the CONTEXT and   *
*       SCOPE of the OLDPLEX.                                       *
*                                                                   *
*    -  since a PLEX can be either a CONTEXT or SCOPE, we verify    *
*       that the NEWPLEX is not already a valid CONTEXT (i.e, an    *
*       existing CICSplex or CMAS) or SCOPE in the OLDPLEX (i.e,    *
*       an existing CICS system or CICS system group).              *
*                                                                   *
*    -  we GET the CPLEXDEF record for the OLDPLEX, and use this as *
*       a module to CREATE the NEWPLEX.                             *
*                                                                   *
*    -  we GET the CICSPLEX records for the OLDPLEX, and use these  *
*       to add the CMASs in the OLDPLEX to the NEWPLEX.             *
*                                                                   *
*    -  using a list that contains all possible CICSplex definitions, *
*       we GET and FETCH the records from the OLDPLEX, and CREATE   *
*       them in the NEWPLEX.                                        *
*                                                                   *
*    -  we then DISCONNECT from CPSM.                               *
*                                                                   *
```

```
* ------------------------------------------------------------------*
*                                                                   *
* NOTES :                                                           *
*    DEPENDENCIES = S/370                                           *
*    RESTRICTIONS = None                                            *
*    REGISTER CONVENTIONS =                                         *
*        R0              Workarea / external call parameter pointer *
*        R1              Workarea / external call parameter pointer *
*        R2              Resource Table record pointer              *
*        R3              Loop counter                               *
*        R4              List pointer                               *
*        R5              Loop counter                               *
*        R6              Unused                                     *
*        R7              Unused                                     *
*        R8              Unused                                     *
*        R9              Subroutine linkage                         *
*        R10             Subroutine linkage                         *
*        R11             Base register                              *
*        R12             Base register                              *
*        R13             Workarea pointer                           *
*        R14             External call linkage                      *
*        R15             External call linkage                      *
*                                                                   *
*    MODULE TYPE  = Executable                                      *
*    PROCESSOR    = Assembler                                       *
*    ATTRIBUTES   = Read only, Serially Reusable                    *
*                   AMODE(31), RMODE(ANY)                           *
*                                                                   *
* ------------------------------------------------------------------*
*                                                                   *
* ENTRY POINT = EYUAAPI3                                            *
*                                                                   *
*     PURPOSE = All Functions                                       *
*                                                                   *
*     LINKAGE = Executed as a batch program.                        *
*                                                                   *
*     INPUT   = None                                                *
*                                                                   *
*     OUTPUT  = File for messages.                                  *
*               DDNAME  = SYSPRINT                                  *
*               DSORG   = PS                                        *
*               RECFM   = FB                                        *
*               LRECL   = 80                                        *
*               BLKSIZE = as desired (a multiple of 80)             *
*                                                                   *
* ------------------------------------------------------------------*
         EJECT
EYUAAPI3 CSECT
         STM   R14,R12,12(R13)
         LR    R12,R15
         USING EYUAAPI3,R12
*-------------------------------------------------------------------*
*        GETMAIN working storage and set up SA chain.               *
*-------------------------------------------------------------------*
         GETMAIN R,LV=WORKLEN
         ST    R13,4(,1)
         ST    R1,8(,R13)
         L     R1,24(,R13)
         L     R13,8(,R13)
         USING SAVEAREA,R13
```

```
*--------------------------------------------------------------------*
*        Preset return code to error - will change to 0 if all ok.   *
*--------------------------------------------------------------------*
         MVC   RETCODE,=F'8'
*--------------------------------------------------------------------*
*        OPEN file for error messages.                               *
*--------------------------------------------------------------------*
         OPEN  (SYSPRINT,OUTPUT)
*--------------------------------------------------------------------*
*        Specify variables: OLDPLEX, NEWPLEX, MPCMAS                 *
*                                                                    *
*        Insure that the values specified are valid NAME type (i.e,  *
*        valid member name) or following code will fail.             *
*--------------------------------------------------------------------*
         MVC   OLDPLEX,=CL8'plexold'    *** SPECIFY AS DESIRED ***
         MVC   NEWPLEX,=CL8'plexnew'    *** SPECIFY AS DESIRED ***
         MVC   MPCMAS,=CL8'mpcmas'      *** SPECIFY AS DESIRED ***
*--------------------------------------------------------------------*
*        Connect to CPSM API via OLDPLEX.                            *
*--------------------------------------------------------------------*
         MVC   CONTEXT,OLDPLEX
         EXEC  CPSM CONNECT                                          X
                    CONTEXT(CONTEXT)                                 X
                    VERSION(=CL4'0130')                              X
                    THREAD(THREAD)                                   X
                    RESPONSE(RESPONSE)                               X
                    REASON(REASON)
         CLC   RESPONSE,EYUVALUE(OK)    RESPONSE OK?
         BNE   ERRCON                   No - msgs and out
*--------------------------------------------------------------------*
*        Verify that the desired NEWPLEX name is not already a       *
*        PLEX or CMAS.  We do this by trying to set the CONTEXT      *
*        to the NEWPLEX name.  If successful (NEWPLEX already exists *
*        as a CONTEXT) issue messages and get out.                   *
*--------------------------------------------------------------------*
         EXEC  CPSM QUALIFY                                          X
                    CONTEXT(NEWPLEX)                                 X
                    THREAD(THREAD)                                   X
                    RESPONSE(RESPONSE)                               X
                    REASON(REASON)
         CLC   RESPONSE,EYUVALUE(OK)    RESPONSE OK?
         BE    ERRNISPC                 Yes - already a CONTEXT
```

```
*-------------------------------------------------------------------*
*        Verify that the desired NEWPLEX name is not already a      *
*        CSYSDEF or CSYSGRP in the old, soon to be new, CICSplex.   *
*                                                                   *
*        Here we will start issuing EXEC CPSM GET requests, to      *
*        get result sets of different Resource Tables.  We make     *
*        the call through the GETOBJ subroutine.  Variable OBJECT   *
*        must be set with the Resource Table name.  If we only want *
*        a subset of the records for a given Resource Table, we also*
*        set variable CRITERIA with a selection criteria string.    *
*        This string can contain references to any fields in the    *
*        Resource Table, connected by logical operators, and must   *
*        end with a period - . -.  Variable CRITLEN must be loaded  *
*        with the length of the criteria string.                    *
*                                                                   *
*        We will check the RESPONSE from GET calls inline, instead  *
*        of in the subroutine.  The reason for this is that sometimes*
*        a RESPONSE of OK will mean that we have a problem (e.g.,   *
*        the NEWPLEX name already exists as a CICS System name).    *
*-------------------------------------------------------------------*
*
*        Ask for a CSYSSYS record equal to the NEWPLEX name.
*
         MVC   OBJECT,=CL8'CSYSDEF'
         MVC   CRITERIA(5),=CL5'NAME='
         MVC   CRITERIA+5(8),NEWPLEX
         MVI   CRITERIA+13,C'.'
         MVC   CRITLEN,=F'14'
         BAS   R10,GETOBJ               Build result set
         CLC   RESPONSE,EYUVALUE(OK)    RESPONSE OK?
         BE    ERRNISC                  Yes - already a CICS system
         CLC   RESPONSE,EYUVALUE(NODATA) No CSYSDEF with NEWPLEX name?
         BE    NOTCSYS                  Yes - continue
         B     ERRGETO                  No - some error - msgs and out
NOTCSYS  DS    0H
*
*        Ask for a CSYSGRP record equal to the NEWPLEX name.
*
         MVC   OBJECT,=CL8'CSYSGRP'
         MVC   CRITERIA(6),=CL6'GROUP='
         MVC   CRITERIA+6(8),NEWPLEX
         MVI   CRITERIA+14,C'.'
         MVC   CRITLEN,=F'15'
         BAS   R10,GETOBJ               Build the result set
         CLC   RESPONSE,EYUVALUE(OK)    RESPONSE OK?
         BE    ERRNISS                  Yes - already a system group
         CLC   RESPONSE,EYUVALUE(NODATA) No CSYSGRP with NEWPLEX name?
         BE    NOTCGRP                  Yes - continue
         B     ERRGETO                  No - some error - msgs and out
NOTCGRP  DS    0H
*-------------------------------------------------------------------*
*        If we have gotten this far, we know that NEWPLEX is not    *
*        already the name of a CICSplex, CMAS, CICS System, or      *
*        CICS System group - so we can start building the NEWPLEX.  *
*                                                                   *
*        Switch CONTEXT to MPCMAS to build NEWPLEX and add CMASs.   *
*-------------------------------------------------------------------*
         MVC   CONTEXT,MPCMAS
```

```
*--------------------------------------------------------------------*
*       Build new plex using OLDPLEX as a model.                     *
*                                                                    *
*       The record that defines a CICSplex is the CPLEXDEF Resource  *
*       Table.  We will GET the OLDPLEX CPLEXDEF record, modify      *
*       it as needed, and then CREATE the NEWPLEX CPLEXDEF records.  *
*       This creates the NEWPLEX.                                    *
*--------------------------------------------------------------------*
        MVI   PLEXBLT,C'N'            Indicate NEWPLEX not built yet
*
*       First GET CPLEXDEF record for the OLDPLEX.
*
        MVC   OBJECT,=CL8'CPLEXDEF'
        MVC   CRITERIA(9),=CL9'CICSPLEX='
        MVC   CRITERIA+9(8),OLDPLEX
        MVI   CRITERIA+17,C'.'
        MVC   CRITLEN,=F'18'
        BAS   R10,GETOBJ              Build result set
        CLC   RESPONSE,EYUVALUE(OK)   RESPONSE OK?
        BNE   ERRGETO                 No - msgs and out
*
*       Here we start using the GETBUF subroutine.  This subroutine
*       GETMAINs a buffer into which we can FETCH the records of the
*       result set that we last issued a GET for.
*
        BAS   R10,GETBUF              Get storage to receive recs
*
*       Here we start using the FETCH subroutine.  This subroutine
*       reads all the records from the result set into the buffer.
*       On return to mainline, R2 points to the first record in
*       the buffer.
*
        BAS   R10,FETCH               Sets R2 to fetched record
*
*       Change the OLDPLEX CPLEXDEF record into the NEWPLEX
*       CPLEXDEF record.
*
        USING CPLEXDEF,R2             Map the record
        MVC   CPLEXDEF_CICSPLEX,NEWPLEX                              X
                                      Set CICSplex name to NEWPLEX
        MVC   CPLEXDEF_DESC,=CL30'API cloned from'                   X
                                      Modify CICSPlex ....
        MVC   CPLEXDEF_DESC+16(8),OLDPLEX                            X
                                      .... description
        MVC   NEWPLXD(CPLEXDEF_TBL_LEN),0(R2)                        X
                                      Save NEWPLEX def and len ....
        MVC   NEWPLXDL,=A(CPLEXDEF_TBL_LEN)                          X
                                      .... for possible later REMOVE
```

```
*
*       Here we start using the CREATE subroutine.  This subroutine
*       will cause a CPSM Resource Table record to be built.  Variable
*       OBJECT needs to be preset to the Resource Table name, the
*       Resource Table record to be built must be pointed to by R2
*       and must be filled out prior to called CREATE.
*
        BAS   R10,CREATE              CREATE NEWPLEX
        MVI   PLEXBLT,C'Y'            Indicate NEWPLEX now built
*
*       Here we start using the FREEBUF subroutine.  This subroutine
*       FREEMAINs the buffer into which we FETCHed the records.
*
        BAS   R10,FREEBUF             Free record storage
*
*       When a result set is built (in our program by either GET or
*       PERFORM) an id is associated with the result set and placed
*       into the variable pointed to by keyword RESULT (for GET we
*       are using variable RESULT - for PERFORM, RESULT2).  This is
*       done so that subsequent calls can reference the result set
*       built (e.g, FETCH can retrieve records for GET).  When we
*       are done using a result set, we must DISCARD it, so that
*       CPSM frees us resources allocated for the result set.
*       Note that we have not done this with the 2 previous GETs
*       we did since the object of them was to NOT get a result set.
*       If any of the previous GETs caused a result set to get built,
*       we DISCONNECT from CPSM - which causes all our resources to
*       be released - and exit.
*
        MVC   RESULTD,RESULT          Copy GET result set id for     X
                                      DISCARD
        BAS   R10,DISCARD             Discard the GET result set
        DROP  R2                      Drop mapping to CPLEXDEF rec
*-------------------------------------------------------------------*
*       Add CMASs in OLDPLEX to NEWPLEX.                            *
*                                                                   *
*       There is a CICSPLEX Resource Table record for each CMAS     *
*       that participates in the management of a plex.  We first    *
*       ask for all the CICSPLEX records for OLDPLEX, and use       *
*       this info to add the CMASs to the NEWPLEX.                  *
*-------------------------------------------------------------------*
*       Ask for the CICSPLEX records from the OLDPLEX.
*
        MVC   OBJECT,=CL8'CICSPLEX'
        MVC   CRITERIA(9),=CL9'PLEXNAME='
        MVC   CRITERIA+9(8),OLDPLEX
        MVI   CRITERIA+17,C'.'
        MVC   CRITLEN,=F'18'
        BAS   R10,GETOBJ              Build result set
        CLC   RESPONSE,EYUVALUE(OK)   RESPONSE OK?
        BNE   ERRGETO                 no - msgs and out
        BAS   R10,GETBUF              Get storage for records
        BAS   R10,FETCH               Points R2 to first record
        USING CICSPLEX,R2             Map the Resource Table
        L     R5,COUNT                Will loop the number of       X
                                      returned CMASs
*       The MP CMAS is added to the CICSplex when the CPLEXDEF
*       record was CREATEd.  To add any other CMASs to the CICSplex
*       we issue a PERFORM against the CPLEXDEF record for NEWPLEX,
*       with a parm = CICSPLEX(newplex) CMAS(cmasname).
*
        MVC   ADDCPARM(ADDCLEN),ADDC  Build most of parm
        MVC   PARMLEN,=A(ADDCLEN)     Set its length
        MVC   ADDCPLEX,NEWPLEX        Add CICSplex name to parm
        MVC   OBJECT,=CL8'CPLEXDEF'   PERFORM against CPLEXDEF
ADDCMAS DS    0H
        CLC   CICSPLEX_CMASNAME,MPCMAS CMAS = MPCMAS?
        BE    NOADDMP                 Yes - don't add it then
        MVC   ADDCCMAS,CICSPLEX_CMASNAME                          X
                                      Add CMAS name to PARM       X
                                      This comes from the CICSPLEX  X
                                      records.
```

```
*
*        Note that we already have the CICSPLEX result set active,
*        with the id in RESULT.  So here we will use RESULT2 for
*        result set that is built for each PERFORM.
*
         MVC   RESULT2,=F'0'            Always build new result set
         EXEC  CPSM PERFORM                                        X
                    OBJECT(OBJECT)                                 X
                    ACTION(=CL12'ASSIGN')                          X
                    PARM(ADDCPARM)                                 X
                    PARMLEN(PARMLEN)                               X
                    RESULT(RESULT2)                                X
                    CONTEXT(CONTEXT)                               X
                    THREAD(THREAD)                                 X
                    RESPONSE(RESPONSE)                             X
                    REASON(REASON)
         CLC   RESPONSE,EYUVALUE(OK)    RESPONSE OK?
         BNE   ERRPERF                  no - msgs and out
         MVC   RESULTD,RESULT2          Copy PERFORM result set id for X
                                        DISCARD
         BAS   R10,DISCARD              Discard the PERFORM result set
NOADDMP  DS    0H
*
*        We need to get to the next CICSPLEX record for the next CMAS.
*        The GETBUF subroutine places into variable RECLEN the length
*        of the Resource Table record.  We now add this to the address
*        of the current record to point to the next record.
*
         A     R2,RECLEN
         BCT   R5,ADDCMAS               Add the next CMAS
*
*        No more CICSPLEX records - discard the CICSPLEX result set
*        and continue on.
*
         BAS   R10,FREEBUF              Free FETCHed record storage
         MVC   OBJECT,=CL8'CICSPLEX'    For possible DISCARD error msg
         MVC   RESULTD,RESULT           Copy GET result set id for     X
                                        DISCARD
         BAS   R10,DISCARD              Discard the GET result set
         DROP  R2                       Drop mapping to CICSPLEX rec
```

```
*-------------------------------------------------------------------*
*        Take all defs in OLDPLEX and put into NEWPLEX.             *
*                                                                   *
*        We have a list of all CICSplex Resource Table names.  We   *
*        loop through this list, getting all the records for a      *
*        specific Resource Table from the OLDPLEX and adding them   *
*        to the NEWPLEX.                                            *
*-------------------------------------------------------------------*
         MVC   CRITLEN,=F'0'           Want all records from each     X
                                       Resource Table - so we don't   X
                                       want a CRITERIA for GET.
         LA    R3,DEFNUM               Get number of Resource Tables
         LA    R4,DEFLIST              Point R4 to first Resource     X
                                       Table in list
BLDLOOP  DS    0H
         MVC   OBJECT,0(R4)            Move in Resource Table name
*
*        Get old data -  set CONTEXT to OLDPLEX.
*
         MVC   CONTEXT,OLDPLEX
         MVC   SCOPE,OLDPLEX
         BAS   R10,GETOBJ              Build result set
         CLC   RESPONSE,EYUVALUE(OK)   RESPONSE OK?
         BE    GOTDEFS                 Yes - FETCH and add
         CLC   RESPONSE,EYUVALUE(NODATA) No records returned?
         BE    NODATA                  Yes - on to next Resource Tab
         B     ERRGETO                 GET error - msgs and out
GOTDEFS  DS    0H
         BAS   R10,GETBUF              Get storage for records
         BAS   R10,FETCH               Point R2 to first record
         L     R5,COUNT                Load number of records for loop
*
*        Add new data - set CONTEXT to NEWPLEX.
*
         MVC   CONTEXT,NEWPLEX
CRELOOP  DS    0H
*
*        We need to check if the object being created is a RTAINAPS
*        table.  If it is, we need to check if the SCOPE is the
*        OLDPLEX name - and if so, change it to the NEWPLEX name.
*        The RTAINAPS table is the only resource table in our list
*        that may have the OLDPLEX specified as a SCOPE.
*
         CLC   OBJECT,=CL8'RTAINAPS'   Creating an RTAINAPS?
         BNE   CRELOOP2                No, just CREATE it
         USING RTAINAPS,R2             May to the record
         CLC   RTAINAPS_SCOPE,OLDPLEX  Is SCOPE equal to OLDPLEX?
         BNE   CRELOOP2                No, don't change record
         MVC   RTAINAPS_SCOPE,NEWPLEX  Alter SCOPE to NEWPLEX
         DROP  R2                      Drop mapping to RTAINAPS rec
CRELOOP2 DS    0H
         BAS   R10,CREATE              CREATE record in NEWPLEX
         A     R2,RECLEN               Point to next record
         BCT   R5,CRELOOP              Loop
         BAS   R10,FREEBUF             Release record storage
         MVC   RESULTD,RESULT          Copy GET result set id for     X
                                       DISCARD
         BAS   R10,DISCARD             Discard the GET result set
NODATA   DS    0H
         LA    R4,8(,R4)               Point to next Resource Table
         BCT   R3,BLDLOOP              Do next Resource Table
*
*        We have gone through all the Resource Tables ok.  Set
*        the return code to 0.
*
         MVC   RETCODE,=F'0'
```

```
*-------------------------------------------------------------------*
*        Disconnect the connection and exit the program.            *
*-------------------------------------------------------------------*
EXITDISC DS    0H
         EXEC  CPSM DISCONNECT                                     X
                    THREAD(THREAD)                                 X
                    RESPONSE(RESPONSE)                             X
                    REASON(REASON)
EXIT     DS    0H
         CLOSE (SYSPRINT)
*-------------------------------------------------------------------*
*        Unchain save area, FREEMAIN working storage, and restore   *
*        registers.                                                 *
*-------------------------------------------------------------------*
         L     R2,RETCODE              Retrieve return code
         L     R13,4(,R13)
         L     R1,8(,R13)
         FREEMAIN R,A=(R1),LV=WORKLEN
         L     R14,12(,R13)
         LR    R15,R2
         LM    R0,R12,20(R13)
         LA    R15,0
         BR    R14
```

```
*-------------------------------------------------------------------*
*        Error routines.                                            *
*-------------------------------------------------------------------*
ERRCON   DS    0H
         MVC   OUTLINE,=CL80'Error: Connecting to the API'
         BAS   R9,PUTMSG
         BAS   R10,DORR                Format and msg RESPONSE/REASON
         B     EXIT                    Exit
ERRNISPC DS    0H
         MVC   OUTLINE,=CL80'Error: NEWPLEX is already defined as a CICX
               Splex or CMAS'
         BAS   R9,PUTMSG
         B     EXITDISC                DISCONNECT and exit
ERRNISC  DS    0H
         MVC   OUTLINE,=CL80'Error: NEWPLEX is already defined as a CICX
               S system in the OLDPLEX'
         BAS   R9,PUTMSG
         B     EXITDISC                DISCONNECT and exit
ERRNISS  DS    0H
         MVC   OUTLINE,=CL80'Error: NEWPLEX is already defined as a CICX
               S system group in the OLDPLEX'
         BAS   R9,PUTMSG
         B     EXITDISC                DISCONNECT and exit
ERRPERF  DS    0H
         MVC   OUTLINE,=CL80'Error: Adding a CMAS to the NEWPLEX'
         BAS   R9,PUTMSG
         MVC   OUTLINE,=CL80' '
         MVC   OUTTXT1,=CL10'CMASNAME:'
         MVC   OUTDAT1,ADDCCMAS
         BAS   R9,PUTMSG
         BAS   R10,DORR                Format and msg RESPONSE/REASON
         B     EXITERR
ERRGETO  DS    0H
         MVC   OUTLINE,=CL80'Error: GETting an object'
         BAS   R9,PUTMSG
         B     DOOBJMSG
ERRQUERY DS    0H
         MVC   OUTLINE,=CL80'Error: QUERYing a record size.'
         BAS   R9,PUTMSG
         B     DOOBJMSG
ERRFETCH DS    0H
         MVC   OUTLINE,=CL80'Error: FETCHing an object.'
         BAS   R9,PUTMSG
         B     DOOBJMSG
ERRCREAT DS    0H
         MVC   OUTLINE,=CL80'Error: CREATEing an object.'
         BAS   R9,PUTMSG
         B     DOOBJMSG
ERRDISCA DS    0H
         MVC   OUTLINE,=CL80'Error: DISCARDing object.'
         BAS   R9,PUTMSG
DOOBJMSG DS    0H
         MVC   OUTLINE,=CL80' '
         MVC   OUTTXT1,=CL10'OBJECT:'
         MVC   OUTDAT1,OBJECT
         BAS   R9,PUTMSG
         BAS   R10,DORR
EXITERR  DS    0H
         CLI   PLEXBLT,C'Y'            Did we CREATE the NEWPLEX?
         BNE   EXITDISC                No - just DISCONNECT and exit
```

```
*
*        We had already CREATEd the NEWPLEX when an error occurred
*        so we want to delete the NEWPLEX before ending our program.
*
         EXEC  CPSM REMOVE                                          X
                    OBJECT(=CL8'CPLEXDEF')                          X
                    FROM(NEWPLXD)                                   X
                    LENGTH(NEWPLXDL)                                X
                    CONTEXT(MPCMAS)                                 X
                    THREAD(THREAD)                                  X
                    RESPONSE(RESPONSE)                              X
                    REASON(REASON)
         CLC   RESPONSE,EYUVALUE(OK)    RESPONSE OK?
         BE    EXITDISC                 Yes - DISCONNECT and exit
         MVC   OUTLINE,=CL80'Error: REMOVEing NEWPLEX.'
         BAS   R9,PUTMSG
         BAS   R10,DORR
         B     EXITDISC                 DISCONNECT and exit
*-------------------------------------------------------------------*
*        End of error routines.                                     *
*-------------------------------------------------------------------*
*        Subroutines.                                               *
*-------------------------------------------------------------------*
PUTMSG   DS    0H
         PUT   SYSPRINT,OUTLINE
         BR    R9
DORR     DS    0H
*-------------------------------------------------------------------*
*        Subroutine: DORR                                           *
*        Entry:      Via BAS R10,DORR                               *
*        Function:   Put out error messages indicating what function  *
*                    failed and the RESPONSE and REASON from that     *
*                    function.                                      *
*        Processing: -  Format the EXEC CPSM RESPONSE and move to the *
*                       OUTLINE.                                    *
*                    -  Format the EXEC CPSM REASON and move to the   *
*                       OUTLINE.                                    *
*                    -  Call the PUTMSG subroutine to send the        *
*                       RESPONSE/REASON data to SYSPRINT.           *
*                    -  Return to caller.                           *
*-------------------------------------------------------------------*
         MVC   OUTLINE,=CL80' '         clear format area
         MVC   OUTTXT1,=CL10'RESPONSE:' move in ....
         L     R3,RESPONSE              load up the RESPONSE
         CVD   R3,DOUBLE                convert to decimal
         MVC   OUTDAT1(6),=XL6'402020202120' move in EDIT pattern
         ED    OUTDAT1(6),DOUBLE+5      EDIT RESPONSE to format area
         MVC   OUTTXT2,=CL10'REASON:'   .... constant data
         L     R3,REASON                load up the REASON
         CVD   R3,DOUBLE                convert to decimal
         MVC   OUTDAT2(6),=XL6'402020202120' move in EDIT pattern
         ED    OUTDAT2(6),DOUBLE+5      EDIT REASON to format area
         BAS   R9,PUTMSG                SEND it
         MVC   OUTLINE,=CL80' '         clear out OUTLINE again
         BAS   R9,PUTMSG                put out blank line
         BR    R10                      return to caller
```

```
GETOBJ   DS    0H
*-------------------------------------------------------------------*
*        Subroutine: GETOBJ                                         *
*        Entry:      Via BAS R10,GETOBJ                             *
*        Function:   Issue the EXEC CPSM GET command to create a    *
*                    result set for a specific object.  Note that   *
*                    all operands for GET must be preset in         *
*                    mainline code - except for RESULT.             *
*        Processing: -  Clear out the result set id - RESULT - so   *
*                       that a new result set is always built.  It  *
*                       is the responsibility of mainline to DISCARD*
*                       any previous result set for GET.            *
*                    -  Determine if the GET request has a CRITERIA *
*                       and use the proper EXEC CPSM GET call.      *
*                    -  Note that GETOBJ does not check the RESPONSE*
*                       from CPSM - this is done in mainline.       *
*                    -  Return to caller.                           *
*-------------------------------------------------------------------*
         MVC   RESULT,=F'0'            Always get new result set
         CLC   CRITLEN,=F'0'
         BE    GETNOCRT
         EXEC  CPSM GET                                             X
                    OBJECT(OBJECT)                                  X
                    CRITERIA(CRITERIA)                              X
                    LENGTH(CRITLEN)                                 X
                    COUNT(COUNT)                                    X
                    RESULT(RESULT)                                  X
                    THREAD(THREAD)                                  X
                    CONTEXT(CONTEXT)                                X
                    RESPONSE(RESPONSE)                              X
                    REASON(REASON)
         BR    R10
GETNOCRT DS    0H
         EXEC  CPSM GET                                             X
                    OBJECT(OBJECT)                                  X
                    COUNT(COUNT)                                    X
                    RESULT(RESULT)                                  X
                    THREAD(THREAD)                                  X
                    CONTEXT(CONTEXT)                                X
                    RESPONSE(RESPONSE)                              X
                    REASON(REASON)
         BR    R10
```

```
GETBUF   DS    0H
*-------------------------------------------------------------------*
*        Subroutine: GETBUF                                         *
*        Entry:      Via BAS R10,GETBUF                             *
*        Function:   Get a buffer to hold all the records contained *
*                    in the last result set we build though GET.    *
*        Processing: -  Issue EXEC CPSM QUERY to get the length of  *
*                       the Resource Table record.  We use the same *
*                       OBJECT and RESULT from the GET.  Variable   *
*                       RECLEN gets the record length.              *
*                    -  Check the RESPONSE from QUERY and issue msgs*
*                       and EXIT if not OK.                         *
*                    -  Multiple the RECLEN times the COUNT (returned *
*                       from last GET) to determine the buffer size *
*                       required and GETMAIN it.                    *
*                    -  Save the buffer length (BUFLEN) and buffer  *
*                       address (BUFFER) for the FREEMAIN call in   *
*                       the FREEBUF subroutine.                     *
*                    -  Return to caller.                           *
*-------------------------------------------------------------------*
         EXEC  CPSM QUERY                                          X
                    OBJECT(OBJECT)                                 X
                    DATALENGTH(RECLEN)                             X
                    RESULT(RESULT)                                 X
                    THREAD(THREAD)                                 X
                    RESPONSE(RESPONSE)                             X
                    REASON(REASON)
         CLC   RESPONSE,EYUVALUE(OK)    RESPONSE OK?
         BNE   ERRQUERY                 No - msgs and out
         L     R0,RECLEN
         L     R1,COUNT
         MR    R0,R0
         GETMAIN R,LV=(R1)
         ST    R0,BUFLEN
         ST    R1,BUFFER
         BR    R10
FREEBUF  DS    0H
*-------------------------------------------------------------------*
*        Subroutine: FREEBUF                                        *
*        Entry:      Via BAS R10,FREEBUF                            *
*        Function:   To FREEMAIN the buffer created to hold the     *
*                    records from the last result set we built .    *
*                    through GET.                                   *
*        Processing: -  Use BUFLEN and BUFFER from GETBUF, FREEMAIN  *
*                       the buffer area.                            *
*                    -  Return to caller.                           *
*-------------------------------------------------------------------*
         L     R0,BUFLEN
         L     R1,BUFFER
         FREEMAIN R,A=(R1),LV=(R0)
         BR    R10
```

```
FETCH    DS    0H
*-------------------------------------------------------------------*
*        Subroutine: FETCH                                          *
*        Entry:      Via BAS R10,FETCH                              *
*        Function:   Issue the EXEC CPSM FETCH command to retrieve  *
*                    the result set created by the last GET.        *
*                    mainline code - except for RESULT.             *
*        Processing: -  For FETCH we must provide a receiving area  *
*                       and length.  We put in the area length into *
*                       R2 and the area length in variable LENGTH.  *
*                       Note that we got both the area and length   *
*                       in the GETBUF routine.                      *
*                    -  Issue the FETCH request using the result set*
*                       id - RESULT - from the last GET.            *
*                    -  Check the RESPONSE - if not OK, issue msgs  *
*                       and exit.                                   *
*                    -  Return to caller.                           *
*-------------------------------------------------------------------*
         L     R2,BUFFER
         MVC   LENGTH,BUFLEN
         EXEC  CPSM FETCH                                          X
                    ALL                                            X
                    INTO(0(,R2))                                   X
                    LENGTH(LENGTH)                                 X
                    COUNT(COUNT)                                   X
                    RESULT(RESULT)                                 X
                    THREAD(THREAD)                                 X
                    RESPONSE(RESPONSE)                             X
                    REASON(REASON)
         CLC   RESPONSE,EYUVALUE(OK)
         BNE   ERRFETCH
         BR    R10
```

```
CREATE   DS    0H
*-------------------------------------------------------------------*
*        Subroutine: CREATE                                         *
*        Entry:      Via BAS R10,CREATE                             *
*        Function:   Issue the EXEC CPSM CREATE to build a Resource *
*                    Table record.                                  *
*        Processing: -  Place the length of the record to be build  *
*                       (RECLEN from GETBUF) into variable LENGTH.  *
*                       R2 should have been set by mainline to point*
*                       to the record itself.                       *
*                    -  When CREATEing a LNKxxCG record (spec to    *
*                       group link) we need to specify a parm -     *
*                       NONE. - to indicate that we only want the   *
*                       CREATE to associate the spec to the group.  *
*                       Any systems in the group that need to be    *
*                       added to the spec have already been done    *
*                       by CREATE of LNKxxCS records (spec to       *
*                       system link).  If this is a LNKxxCG record, *
*                       set the PARM and PARMLENgth.                *
*                    -  Issue the proper format of EXEC CPSM CREATE *
*                       (either with PARM/PARMLEN or without).      *
*                    -  Check the RESPONSE - if not OK, issue msgs  *
*                       and exit.                                   *
*                    -  Return to caller.                           *
*-------------------------------------------------------------------*
         MVC   LENGTH,RECLEN
         CLC   OBJECT(4),=CL4'LNKS'
         BNE   CRENOPRM
         CLC   OBJECT+6(2),=CL2'CG'
         BNE   CRENOPRM
         MVC   PARM,=CL5'NONE.'
         MVC   PARMLEN,=F'5'
         EXEC  CPSM CREATE                                          X
                    OBJECT(OBJECT)                                  X
                    FROM(0(,R2))                                    X
                    LENGTH(LENGTH)                                  X
                    PARM(PARM)                                      X
                    PARMLEN(PARMLEN)                                X
                    THREAD(THREAD)                                  X
                    CONTEXT(CONTEXT)                                X
                    RESPONSE(RESPONSE)                              X
                    REASON(REASON)
         B     CRECHKRR
CRENOPRM DS    0H
         EXEC  CPSM CREATE                                          X
                    OBJECT(OBJECT)                                  X
                    FROM(0(,R2))                                    X
                    LENGTH(LENGTH)                                  X
                    THREAD(THREAD)                                  X
                    CONTEXT(CONTEXT)                                X
                    RESPONSE(RESPONSE)                              X
                    REASON(REASON)
CRECHKRR DS    0H
         CLC   RESPONSE,EYUVALUE(OK)
         BNE   ERRCREAT
         BR    R10
```

```
DISCARD  DS    0H
*-------------------------------------------------------------------*
*        Subroutine: DISCARD                                        *
*        Entry:      Via BAS R10,DISCARD                            *
*        Function:   Issue the EXEC CPSM DISCARD to discard a result  *
*                    set built by CPSM.  In our program, both GET   *
*                    and PERFORM build result sets.                 *
*        Processing: -  Issue EXEC CPSM DISCARD for the result set.   *
*                       The result set id must be placed into       *
*                       RESULTD by mainline.                        *
*                    -  Check the RESPONSE - if not OK, issue msgs  *
*                       and exit.                                   *
*                    -  Return to caller.                           *
*-------------------------------------------------------------------*
         EXEC  CPSM DISCARD                                         X
                    RESULT(RESULTD)                                 X
                    THREAD(THREAD)                                  X
                    RESPONSE(RESPONSE)                              X
                    REASON(REASON)
         CLC   RESPONSE,EYUVALUE(OK)
         BNE   ERRDISCA
         BR    R10
*-------------------------------------------------------------------*
*        End of subroutines.                                        *
*-------------------------------------------------------------------*
```

```
*-------------------------------------------------------------------*
*       Following is a list of all CPSM Resource Tables that can    *
*       be part of a CICSplex.  The order that they are in (which   *
*       is the order they will be built in our program) is          *
*       important, since some Resource Tables will reference other  *
*       Resource Tables previously built.  The following list is    *
*       complete and the order OK for the current release of        *
*       CPSM (V1R3M0).                                              *
*-------------------------------------------------------------------*
DEFLIST  DS    0C
         DC    CL8'PERIODEF'    Time period definitions
         DC    CL8'ACTION  '    RTA action definitions
         DC    CL8'CSYSDEF '    CICS system definitions
         DC    CL8'CSYSGRP '    CICS system group definitions
         DC    CL8'CSGLCGCS'    CICS systems in groups links
         DC    CL8'CSGLCGCG'    CICS groups in groups links
         DC    CL8'MONDEF  '    Monitor definitions
         DC    CL8'MONGROUP'    MON group definitions
         DC    CL8'MONSPEC '    MON specification definitions
         DC    CL8'MONINGRP'    MON def in MON group links
         DC    CL8'MONINSPC'    MON spec to MON group links
         DC    CL8'LNKSMSCS'    MON spec to CICS system links
         DC    CL8'LNKSMSCG'    MON spec to CICS group links
         DC    CL8'EVALDEF '    RTA evaluation definitions
         DC    CL8'RTADEF  '    Real time analysis definitions
         DC    CL8'STATDEF '    User status probe definitions
         DC    CL8'RTAGROUP'    RTA group definitions
         DC    CL8'RTASPEC '    RTA specification definitions
         DC    CL8'RTAINGRP'    RTADEF in RTA group links
         DC    CL8'STAINGRP'    STATDEF in RTA group links
         DC    CL8'RTAINSPC'    RTA spec to RTA group links
         DC    CL8'LNKSRSCS'    RTA spec to CICS group links
         DC    CL8'LNKSRSCG'    RTA spec to CICS system links
         DC    CL8'APSPEC  '    RTA/APM specification defs
         DC    CL8'RTAINAPS'    RTA/APM spec to RTA group links
         DC    CL8'CMDMPAPS'    RTA spec to primary CMAS links
         DC    CL8'CMDMSAPS'    RTA spec to secondary CMAS links
         DC    CL8'TRANGRP '    transaction group definitions
         DC    CL8'WLMDEF  '    Workload definitions
         DC    CL8'WLMGROUP'    WLM group definitions
         DC    CL8'WLMSPEC '    WLM specification definitions
         DC    CL8'DTRINGRP'    Transactions in trangrp links
         DC    CL8'WLMINGRP'    WLM def in WLM group links
         DC    CL8'WLMINSPC'    WLM spec to WLM group links
         DC    CL8'LNKSWSCS'    WLM spec to CICS group links
         DC    CL8'LNKSWSCG'    WLM spec to CICS system links
DEFNUM   EQU   (*-DEFLIST)/8
ADDC     DS    0X
         DC    CL09'CICSPLEX('
         DC    CL08' '
         DC    CL07') CMAS('
         DC    CL08' '
         DC    CL02').'
ADDCLEN  EQU   *-ADDC
SYSPRINT DCB   DDNAME=SYSPRINT,DSORG=PS,MACRF=PM
WORKSTOR DSECT
SAVEAREA DS    18F
DFHEIPL  DS    13F
         DS    51F
```

```
DOUBLE   DS    D
RETCODE  DS    F
RESPONSE DS    F
REASON   DS    F
THREAD   DS    F
RESULT   DS    F
RESULT2  DS    F
RESULTD  DS    F
COUNT    DS    F
LENGTH   DS    F
PARMLEN  DS    F
BUFLEN   DS    F
BUFFER   DS    F
RECLEN   DS    F
NEWPLXDL DS    F
CRITLEN  DS    F
CRITERIA DS    CL80
CONTEXT  DS    CL8
SCOPE    DS    CL8
OBJECT   DS    CL8
OLDPLEX  DS    CL8
NEWPLEX  DS    CL8
MPCMAS   DS    CL8
OUTLINE  DS    0CL80
OUTTXT1  DS    CL10
OUTDAT1  DS    CL8
         DS    CL2
OUTTXT2  DS    CL10
OUTDAT2  DS    CL8
         DS    CL42
PARM     DS    CL5
PLEXBLT  DS    CL1
ADDCPARM DS    0XL(ADDCLEN)
         DS    CL09
ADDCPLEX DS    CL08
         DS    CL07
ADDCCMAS DS    CL08
         DS    CL02
         DS    D
NEWPLXD  DS    XL(CPLEXDEF_TBL_LEN)
WORKLEN  EQU   *-WORKSTOR
         COPY  CPLEXDEF
         COPY  CICSPLEX
         COPY  RTAINAPS
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END   EYUAAPI3
```

## Sample program EYULAPI4

Program EYULAPI4 is written in COBOL for the CICS environment.

**EYUxAPI4**

This program does the following:

- Establishes a connect to the API.
- Creates a BAS definition for a TS Model (TSMDEF) specifying a version of 1.
- Creates a result set containing the previously defined TSMDEF.
- Issues a PERFORM OBJECT command to INSTALL the TSMDEF into the target scope.
- Terminates the API connection.
- BAS errors are processed using BINCONRS, BINCONSC, and BINSTERR resource table records.

**Commands Used:** CONNECT, CREATE, GET, PERFORM OBJECT, FEEDBACK, FETCH, TERMINATE, TRANSLATE

## sample program EYULAPI4

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. EYULAPI4
*****************************************************************
*                                                               *
* MODULE NAME = EYULAPI4                                        *
*                                                               *
* DESCRIPTIVE NAME = CPSM SAMPLE API PROGRAM 4                  *
*                    (SAMPLE COBOL VERSION)                     *
*                                                               *
*  COPYRIGHT = Licensed Materials - Property of IBM             *
*              5695-081                                         *
*              (C) Copyright IBM Corp. 1995, 1997               *
*              All Rights Reserved                              *
*                                                               *
*              US Government Users Restricted Rights - Use,     *
*              duplication or disclosure restricted by GSA ADP  *
*              Schedule Contract with IBM Corp.                 *
*                                                               *
* STATUS = %CP00                                                *
*                                                               *
* FUNCTION =                                                    *
*                                                               *
* TO PROVIDE AN EXAMPLE OF THE USE OF THE FOLLOWING EXEC CPSM   *
* COMMANDS: CONNECT, CREATE, FEEDBACK, FETCH, GET,              *
*           PERFORM OBJECT, TERMINATE.                          *
*                                                               *
* WHEN INVOKED, THE PROGRAM DEPENDS UPON THE VALUES HELD IN THE *
* W-CONTEXT AND W-SCOPE DECLARATIONS WHEN ESTABLISHING A        *
* CONNECTION WITH CICSPLEX SM. THEY MUST TAKE THE FOLLOWING     *
* VALUES:                                                       *
*                                                               *
* W-CONTEXT  = THE NAME OF A CMAS OR CICSPLEX. REFER TO THE     *
*              DESCRIPTION OF THE EXEC CPSM CONNECT COMMAND     *
*              FOR FURTHER INFORMATION REGARDING THE CONTEXT    *
*              OPTION.                                          *
*                                                               *
* W-SCOPE    = THE NAME OF A CICSPLEX, CICS SYSTEM, OR CICS     *
*              SYSTEM GROUP WITHIN THE CICSPLEX. REFER TO THE   *
*              DESCRIPTION OF THE EXEC CPSM CONNECT COMMAND     *
*              FOR FURTHER INFORMATION REGARDING THE SCOPE      *
*              OPTION.                                          *
*                                                               *
* THIS SAMPLE REQUIRES NO PARAMETERS AT INVOCATION TIME.        *
*                                                               *
* WHEN CREATING THE BAS DEFINITION THE PROGRAM DEPENDS UPON THE *
* VALUES HELD IN THE W-DEFNAME AND W-DEFPREFIX DECLARATIONS.    *
* THEY MUST TAKE THE FOLLOWING VALUES:                          *
*                                                               *
* W-DEFNAME  = THE NAME OF THE CREATED BAS DEFINITION. A        *
*              1 TO 8 CHARACTER VALUE.                          *
*                                                               *
* W-DEFPFIX  = THE MODEL PREFIX OF THE CREATED BAS DEFINITION.  *
*              A 1 TO 16 CHARACTER VALUE.                       *
*                                                               *
*                                                               *
```

```
* WHEN INSTALLING THE BAS DEFINITION THE PROGRAM USES THE       *
* VALUE HELD IN THE W-TSCOPE DECLARATION AS THE TARGET FOR      *
* THE INSTALL OPERATION. IT MUST TAKE THE FOLLOWING VALUE :     *
*                                                               *
* W-TSCOPE   = THE NAME OF A CICS SYSTEM, OR CICS               *
*              SYSTEM GROUP WITHIN THE CICSPLEX. REFER TO THE   *
*              DESCRIPTION OF THE TARGET PARAMETER OF AN        *
*              INSTALL ACTION IN THE RESOURCE TABLE REFERENCE   *
*              FOR FURTHER INFORMATION REGARDING THE TARGET     *
*              SCOPE VALUE.                                     *
*                                                               *
*                                                               *
* THE SAMPLE ESTABLISHES AN API CONNECTION AND ISSUES A CREATE  *
* COMMAND TO CREATE A BAS DEFINITION. A GET COMMAND IS ISSUED   *
* TO OBTAIN A RESULT SET CONTAINING THE CREATED BAS DEFINITION. *
*                                                               *
* USING THE PERFORM OBJECT ACTION(INSTALL) COMMAND EACH RECORD  *
* IN THE RESULT SET IS INSTALLED INTO THE TARGET SCOPE          *
* IDENTIFIED BY THE W-SCOPE DECLARATION.                        *
*                                                               *
* FINALLY, THE API CONNECTION IS TERMINATED.                    *
*                                                               *
* ANY BAS ERRORS ARE REPORTED USING THE BINCONRS, BINCONSC, AND *
* BINSTERR RESOURCE TABLES.                                     *
*                                                               *



* NOTES :                                                       *
*   DEPENDENCIES = S/390, CICS                                  *
*   RESTRICTIONS = NONE                                         *
*   REGISTER CONVENTIONS =                                      *
*   MODULE TYPE  = EXECUTABLE                                   *
*   PROCESSOR    = COBOL                                        *
*   ATTRIBUTES   = READ ONLY, SERIALLY REUSABLE                 *
*                                                               *
* ------------------------------------------------------------- *
* ENTRY POINT = EYULAPI4                                        *
*                                                               *
* PURPOSE = ALL FUNCTIONS.                                      *
*                                                               *
* LINKAGE = FROM CICS EITHER WITH EXEC CICS LINK OR AS A CICS   *
*           TRANSACTION.                                        *
*                                                               *
* INPUT   = NONE.                                               *
*                                                               *
* ------------------------------------------------------------- *
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*---------------------------------------------------------------*
*    CHANGE W-CONTEXT AND W-SCOPE TO MATCH YOUR INSTALLATION  *
*    CHANGE W-DEFNAME AND W-DEFPFIX FOR THE CREATE COMMAND.   *
*    CHANGE W-TSCOPE FOR THE PERFORM OBJECT COMMAND.          *
*---------------------------------------------------------------*
 01 W-CONTEXT         PIC X(8) VALUE 'RTGA    '.
 01 W-SCOPE           PIC X(8) VALUE 'RTGA    '.
 01 W-DEFNAME         PIC X(8) VALUE 'EYULAPI4'.
 01 W-DEFPFIX         PIC X(16) VALUE 'EYUL*           '.
 01 W-TSCOPE          PIC X(8) VALUE 'RTGF    '.
*---------------------------------------------------------------*
```

```
 01 W-RESPONSE        PIC S9(8) USAGE BINARY.
 01 W-REASON          PIC S9(8) USAGE BINARY.
 01 W-BUFFER          PIC X(32767).
 01 W-BUFFERLEN       PIC S9(8) COMP.
 01 W-FBBUFF          PIC X(248).
 01 W-FBTTKN          PIC S9(8) COMP.
 01 W-THREAD          PIC S9(8) USAGE BINARY.
 01 W-RESULT          PIC S9(8) USAGE BINARY.
 01 W-RECCNT          PIC S9(8) USAGE BINARY.
 01 W-CRITERIA        PIC X(80) VALUE SPACES.
 01 W-CRITERIALEN     PIC S9(8) USAGE BINARY.
 01 W-PARM            PIC X(80) VALUE SPACES.
 01 W-PARMLEN         PIC S9(8) USAGE BINARY.
 01 W-MSG-TEXT.
   02 W-TEXT          PIC X(80) VALUE SPACES.
   02 W-LINECTL       PIC X(1) VALUE X'13'.
 01 ARRAYS.
   02 CH8ARR          OCCURS 20 TIMES PIC X(8).
   02 FULLARR         OCCURS 60 TIMES PIC S9(8) COMP.
 01 III               PIC S9(8) VALUE ZERO.
 01 CODEV             PIC S9(8) COMP.
 01 CHARV             PIC X(12).
 01 LASTCMD           PIC X(20).
 01 LASTTHR           PIC S9(8) COMP.
 01 LASTRES           PIC S9(8) COMP VALUE 0.
 01 BINZERO           PIC X(1)  VALUE X'00'.
 01 BLNKPAD           PIC X(40)
     VALUE '                                        '.
 01 FBCHAR2           PIC X(2).
 01 FBHALF4           REDEFINES FBCHAR2.
   02 FBHALF          PIC S9(4) COMP.
 01 PICZZZ9A          PIC ZZZ9.
 01 PICZZZ9B          PIC ZZZ9.
 01 PICZZZ9           PIC ZZZ9.
 01 PYCZZZ9           PIC ZZZ9.
 01 PIKZZZ9           PIC ZZZ9.
 01 PYKZZZ9           PIC ZZZ9.
 01 PICZZZZZZZ9       PIC ZZZZZZZ9.
 01 CHR8              PIC X(8).
 01 CHR12             PIC X(12).
 01 CHAR6             PIC X(6).
 01 CHAR12            PIC X(12).
*Include the resource table copybooks...
 COPY TSMDEF.
 COPY FEEDBACK.
 COPY BINCONRS.
 COPY BINCONSC.
 COPY BINSTERR.
```

```
**************************
* Start of LINKAGE section *
**************************
 LINKAGE SECTION.

 PROCEDURE DIVISION.
 EYULAPI4-START SECTION.
 EYULAPI4-00.

*-------------------------------------------------------------*
*    OBTAIN A CPSM API CONNECTION.                            *
*                                                             *
*    THE API WILL RETURN A TOKEN IDENTIFYING THE THREAD IN    *
*    VARIABLE W-THREAD.                                       *
*-------------------------------------------------------------*
     MOVE 'Establishing Connection...' TO W-TEXT.
*    DISPLAY W-TEXT.
     EXEC CICS SEND FROM(W-TEXT) LENGTH(81) ERASE END-EXEC.
     EXEC CPSM CONNECT
               CONTEXT(W-CONTEXT)
               SCOPE(W-SCOPE)
               VERSION('0140')
               THREAD(W-THREAD)
               RESPONSE(W-RESPONSE)
               REASON(W-REASON)
     END-EXEC.
     IF W-RESPONSE NOT = EYUVALUE(OK) GO TO NO-CONNECT.


*-------------------------------------------------------------*
*    CREATE A TS MODEL DEFINITION (TSMDEF)                    *
*                                                             *
*    A TSMDEF is created with a version of 1.                 *
*-------------------------------------------------------------*
     INITIALIZE TSMDEF.
     MOVE X'01' TO DEFVER OF TSMDEF.
     MOVE W-DEFNAME TO NAME-R OF TSMDEF.
     MOVE W-DEFPFIX TO PREFIX OF TSMDEF.
     MOVE DFHVALUE(AUXILIARY) TO LOCATION OF TSMDEF.
     MOVE EYUVALUE(NO) TO RECOVERY OF TSMDEF.
     MOVE EYUVALUE(NO) TO SECURITY-R OF TSMDEF.
     MOVE 'Sample TSMDEF definition' TO DESCRIPTION OF TSMDEF.
* Copy the definition into our buffer...
     MOVE TSMDEF TO W-BUFFER.
     MOVE TSMDEF-TBL-LEN TO W-BUFFERLEN.
     MOVE 'Creating TSMDEF...' TO W-TEXT.
*    DISPLAY W-TEXT.
     EXEC CICS SEND FROM(W-TEXT) LENGTH(81) WAIT END-EXEC.
     EXEC CPSM CREATE
               OBJECT('TSMDEF')
               FROM(W-BUFFER)
               LENGTH(W-BUFFERLEN)
               THREAD(W-THREAD)
               RESPONSE(W-RESPONSE)
               REASON(W-REASON)
     END-EXEC.
     MOVE 'CREATE' TO LASTCMD.
     MOVE W-THREAD TO LASTTHR.
     MOVE 0 TO LASTRES.
     IF W-RESPONSE NOT = EYUVALUE(OK) GO TO UNEXPECTED.
```

```
*-------------------------------------------------------------*
*    GET THE TSMDEF RESOURCE TABLE.                           *
*                                                             *
*    CREATE A RESULT SET CONTAINING ENTRIES FOR ALL TSMDEFS   *
*    WITH NAMES EQUAL TO THE VALUE OF W-DEFNAME. .            *
*    THE NUMBER OF ENTRIES MEETING THE CRITERIA IS RETURNED   *
*    IN VARIABLE W-RECCNT.                                    *
*-------------------------------------------------------------*
     MOVE 'Get the created TSMDEF Resource Table...' TO W-TEXT.
*    DISPLAY W-TEXT.
     EXEC CICS SEND FROM(W-TEXT) LENGTH(81) WAIT END-EXEC.
     STRING 'NAME=' DELIMITED BY SIZE
            W-DEFNAME DELIMITED BY SIZE
            '.' DELIMITED BY SIZE
            INTO W-CRITERIA.
     MOVE LENGTH OF W-CRITERIA TO W-CRITERIALEN.
     MOVE BINZERO TO W-RESULT.
     EXEC CPSM GET OBJECT('TSMDEF')
                   CRITERIA(W-CRITERIA)
                   LENGTH(W-CRITERIALEN)
                   COUNT(W-RECCNT)
                   RESULT(W-RESULT)
                   THREAD(W-THREAD)
                   RESPONSE(W-RESPONSE)
                   REASON(W-REASON)
     END-EXEC.
     IF W-RESPONSE NOT = EYUVALUE(OK) GO TO NO-GET.


*-------------------------------------------------------------*
*    INSTALL EACH RECORD INTO THE SCOPE IDENTIFIED BY THE     *
*    VALUE OF W-TSCOPE.                                       *
*-------------------------------------------------------------*
     MOVE W-RECCNT TO PICZZZZZZZ9.
     STRING 'Installing ' DELIMITED BY SIZE
            PICZZZZZZZ9 DELIMITED BY SIZE
            ' TSMDEF Entries...' DELIMITED BY SIZE
            INTO W-TEXT.
*    DISPLAY W-TEXT
     EXEC CICS SEND FROM(W-TEXT) LENGTH(81) WAIT END-EXEC.
     STRING '(USAGE(LOCAL) TARGET(' DELIMITED BY SIZE
            W-TSCOPE DELIMITED BY SIZE
            ')).' DELIMITED BY SIZE
            INTO W-PARM.
     MOVE LENGTH OF W-PARM TO W-PARMLEN.

     EXEC CPSM PERFORM OBJECT('TSMDEF')
               ACTION('INSTALL')
               PARM(W-PARM)
               PARMLEN(W-PARMLEN)
               RESULT(W-RESULT)
               THREAD(W-THREAD)
               RESPONSE(W-RESPONSE)
               REASON(W-REASON)
     END-EXEC.
     MOVE 'PERFORM OBJECT' TO LASTCMD.
     MOVE W-THREAD TO LASTTHR.
     MOVE W-RESULT TO LASTRES.
     IF W-RESPONSE NOT = EYUVALUE(OK) GO TO UNEXPECTED.

     MOVE 'Completed. Remove TSMDEF to re-run.' TO W-TEXT.
     GO TO SCRNLOG2.
```

```
************************************************
* Branch here if an unexpected CPSM error occurs *
************************************************
 UNEXPECTED.
     MOVE W-RESPONSE TO PICZZZ9.
     STRING '*** RESPONSE=' DELIMITED BY SIZE PICZZZ9
     DELIMITED BY SIZE BLNKPAD DELIMITED BY SIZE INTO W-TEXT.
     PERFORM SCRNLOG2.
     MOVE W-REASON TO PICZZZ9.
     STRING '*** REASON=' DELIMITED BY SIZE PICZZZ9
     DELIMITED BY SIZE BLNKPAD DELIMITED BY SIZE INTO W-TEXT.
     PERFORM SCRNLOG2.
     MOVE '*** Unexpected error condition arose' TO W-TEXT.
     PERFORM SCRNLOG2.
* Obtain FEEDBACK information
     IF LASTCMD = 'DISCONNECT' GO TO NOFEED.
     IF LASTCMD = 'FEEDBACK' GO TO NOFEED.
     IF LASTCMD = 'TERMINATE' GO TO NOFEED.
     STRING
     '*** Getting FEEDBACK data for ' DELIMITED BY SIZE
     LASTCMD DELIMITED BY SIZE
     INTO W-TEXT.
     PERFORM SCRNLOG2.
     STRING
     BLNKPAD DELIMITED BY SIZE
     BLNKPAD DELIMITED BY SIZE
     INTO W-TEXT.
* Get the FEEDBACK data
 GETFEED.
* Clear error result set count
     MOVE 0 TO FULLARR(1).
     PERFORM GETFB THROUGH EGETFB
* Display FEEDBACK information
* Display information
     IF W-RESPONSE = EYUVALUE(OK)
       PERFORM DISPFEED
       IF FULLARR(1) NOT = 0 PERFORM GETFERT THROUGH EGETFER END-I
-F
       IF LASTRES NOT = 0 GO TO GETFEED END-IF
       MOVE '*** End of FEEDBACK data' TO W-TEXT
       PERFORM SCRNLOG2
       GO TO NOFEED
     END-IF.
     MOVE W-RESPONSE TO PICZZZ9.
     MOVE W-REASON TO PYCZZZ9.
     STRING '*** FEEDBACK not available (' DELIMITED BY SIZE
     PICZZZ9 DELIMITED BY SIZE ',' DELIMITED BY SIZE
     PYCZZZ9 DELIMITED BY SIZE ')' DELIMITED BY SIZE
     BLNKPAD DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
 NOFEED.
     EXEC CICS DELAY FOR SECONDS(10) END-EXEC.
* Exit from test case
     EXEC CICS RETURN END-EXEC.
     GOBACK.
     EXIT.
```

```
******************************************
* This subroutine obtains the FEEDBACK data *
******************************************
 GETFB.
* Use exact buffer size
     MOVE FEEDBACK-TBL-LEN TO W-BUFFERLEN.
     IF LASTRES = 0 GO TO NORESULT.
 RESULT.
     EXEC CPSM FEEDBACK
               INTO(W-FBBUFF) LENGTH(W-BUFFERLEN)
               RESULT(LASTRES)
               THREAD(LASTTHR)
               RESPONSE(W-RESPONSE)
               REASON(W-REASON)
     END-EXEC.

* If command didn't execute, get FEEDBACK no result set
* Command didn't execute?
     IF W-RESPONSE = EYUVALUE(NODATA)
       MOVE 0 TO LASTRES
       GO TO NORESULT
     END-IF.
     GO TO ENDFBACK.
 NORESULT.
* Use exact buffer size
     MOVE FEEDBACK-TBL-LEN TO W-BUFFERLEN.
     EXEC CPSM FEEDBACK
               INTO(W-FBBUFF) LENGTH(W-BUFFERLEN)
               THREAD(LASTTHR)
               RESPONSE(W-RESPONSE)
               REASON(W-REASON)
     END-EXEC.

 ENDFBACK.
 EGETFB.
     EXIT.
```

```
*******************************************************
* Branch here if FEEDBACK Error Result Token available *
*******************************************************
 GETFERT.
     MOVE ERR-OBJECT OF FEEDBACK TO CH8ARR(1).
     STRING
     '*** Getting ' DELIMITED BY SIZE
     CH8ARR(1) DELIMITED BY SIZE
     ' error result set data for FEEDBACK' DELIMITED BY SIZE
     INTO W-TEXT.
     PERFORM SCRNLOG2.
 FERTRES.
* Use largest buffer size
     MOVE FEEDBACK-TBL-LEN TO W-BUFFERLEN.
     EXEC CPSM FETCH
               INTO(W-BUFFER) LENGTH(W-BUFFERLEN)
               RESULT(ERR-RESULT OF FEEDBACK)
               THREAD(LASTTHR)
               RESPONSE(W-RESPONSE)
               REASON(W-REASON)
     END-EXEC.

* Display FEEDBACK Error Result Token information
* Display information
     IF W-RESPONSE = EYUVALUE(OK)
       IF CH8ARR(1)= 'FEEDBACK'
         MOVE W-BUFFER TO W-FBBUFF
         PERFORM DISPFEED
       END-IF
       IF CH8ARR(1)= 'BINSTERR'
         PERFORM DISPBIER
       END-IF
       IF CH8ARR(1)= 'BINCONRS'
         PERFORM DISPBIRS
       END-IF
       IF CH8ARR(1)= 'BINCONSC'
         PERFORM DISPBISC
       END-IF
       GO TO FERTRES
     END-IF.
     MOVE W-RESPONSE TO PICZZZ9.
     MOVE W-REASON TO PYCZZZ9.
     STRING '*** FEEDBACK not available (' DELIMITED BY SIZE
     PICZZZ9 DELIMITED BY SIZE ',' DELIMITED BY SIZE
     PYCZZZ9 DELIMITED BY SIZE ')' DELIMITED BY SIZE
     BLNKPAD DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
 EGETFER.
     EXIT.
```

```
*************************************************
* This subroutine displays FEEDBACK information *
*************************************************
 DISPFEED.
     MOVE W-FBBUFF TO FEEDBACK.
     STRING BINZERO COMMAND OF FEEDBACK DELIMITED BY SIZE
     INTO FBCHAR2.
     MOVE FBHALF TO PICZZZ9.
     MOVE RESPONSE OF FEEDBACK TO PYCZZZ9.
     MOVE REASON OF FEEDBACK TO PIKZZZ9.
     MOVE RSLTRECID OF FEEDBACK TO PYKZZZ9.
     MOVE SPACES TO W-TEXT.
     STRING 'Cmd=' PICZZZ9 ' Attr=' ATTRDATAVAL OF
     FEEDBACK ' Eib=' CEIBDATAVAL OF FEEDBACK ' Err='
     ERRCODEVAL OF FEEDBACK ' Rspn=' PYCZZZ9 ' Reas='
     PIKZZZ9 ' ResId=' PYKZZZ9
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     MOVE ERROR-CODE OF FEEDBACK TO PICZZZ9.
     MOVE CEIBRESP OF FEEDBACK TO PYCZZZ9.
     MOVE CEIBRESP1 OF FEEDBACK TO PIKZZZ9.
     MOVE CEIBFN OF FEEDBACK TO PYKZZZ9.
     MOVE SPACES TO W-TEXT.
     STRING '  ECode=' PICZZZ9 ' RESP=' PYCZZZ9
     ' RESP1=' PIKZZZ9 ' EibFn=' PYKZZZ9 ' Obj='
     OBJECT-A OF FEEDBACK ' OAct=' OBJECT-ACT OF FEEDBACK
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     MOVE SPACES TO W-TEXT.
     STRING '  Att1=' ATTR-NM1 OF FEEDBACK ' 2='
     ATTR-NM2 OF FEEDBACK ' 3=' ATTR-NM3 OF FEEDBACK
     ' 4=' ATTR-NM4 OF FEEDBACK ' 5=' ATTR-NM5 OF
     FEEDBACK DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     MOVE ERR-COUNT OF FEEDBACK TO PICZZZ9.
     MOVE SPACES TO W-TEXT.
     STRING '  FObj=' ERR-OBJECT OF FEEDBACK
     '  FCnt=' PICZZZ9
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     MOVE ERR-COUNT OF FEEDBACK TO FULLARR(1).
     EXIT.
```

```
*************************************************
* This subroutine displays BINSTERR information *
*************************************************
 DISPBIER.
     MOVE W-BUFFER TO BINSTERR.
     MOVE SPACES TO W-TEXT.
     STRING 'CMAS=' CMASNAME OF BINSTERR ' Plex='
     PLEXNAME OF BINSTERR ' CSys=' CICSNAME OF BINSTERR
     ' ResName=' RESNAME OF BINSTERR
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     MOVE RESVER OF BINSTERR TO PICZZZ9.
     MOVE ERRCODE OF BINSTERR TO PYCZZZ9.
     MOVE CRESP1 OF BINSTERR TO PIKZZZ9.
     MOVE CRESP2 OF BINSTERR TO PYKZZZ9.
     MOVE SPACES TO W-TEXT.
     STRING '  ResVer=' PICZZZ9 ' ECode=' PYCZZZ9
     ' RESP=' PIKZZZ9 ' RESP1=' PYKZZZ9
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     MOVE CEIBFN OF BINSTERR TO PICZZZ9.
     MOVE SPACES TO W-TEXT.
     STRING ' EibFn=' PICZZZ9
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     EXIT.
```

```
*************************************************
* This subroutine displays BINCONRS information *
*************************************************
 DISPBIRS.
     MOVE W-BUFFER TO BINCONRS.
     MOVE ERROP OF BINCONRS TO PICZZZ9.
     MOVE SPACES TO W-TEXT.
     STRING 'CMAS=' CMASNAME OF BINCONRS ' Plex='
     PLEXNAME OF BINCONRS ' CSys=' CICSNAME OF BINCONRS
     ' ResType=' RESTYPE OF BINCONRS ' EOp=' PICZZZ9
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     MOVE CANDVER OF BINCONRS TO PICZZZ9.
     MOVE SPACES TO W-TEXT.
     STRING '  CandName=' CANDNAME OF BINCONRS
     ' CandVer=' PICZZZ9 ' CResGrp=' CANDRGRP OF BINCONRS
     ' CResAss=' CANDRASG OF BINCONRS ' CResDes='
     CANDRDSC OF BINCONRS
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     MOVE CANDUSAGE OF BINCONRS TO CODEV.
     MOVE 'BINCONRS' TO CHR8.
     MOVE 'CANDUSAGE' TO CHR12.
     PERFORM XCV2CH
     MOVE CHARV TO CHAR6.
     MOVE CANDTYPE OF BINCONRS TO CODEV.
     MOVE 'BINCONRS' TO CHR8.
     MOVE 'CANDTYPE' TO CHR12.
     PERFORM XCV2CH
     MOVE CHARV TO CHAR12.
     MOVE CANDASGOVR OF BINCONRS TO CODEV.
     MOVE 'BINCONRS' TO CHR8.
     MOVE 'CANDASGOVR' TO CHR12.
     PERFORM XCV2CH
     MOVE SPACES TO W-TEXT.
     STRING '  CandUsa=' CHAR6
     ' CandSGrp=' CANDSGRP OF BINCONRS
     ' CandSTyp=' CHAR12 ' CandAssO=' CHARV
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     MOVE EXISTVER OF BINCONRS TO PICZZZ9.
     MOVE EXISTUSAGE OF BINCONRS TO CODEV.
     MOVE 'BINCONRS' TO CHR8.
     MOVE 'EXISTUSAGE' TO CHR12.
     PERFORM XCV2CH
     MOVE SPACES TO W-TEXT.
     STRING '  ExistName=' EXISTNAME OF BINCONRS
     ' ExistVer=' PICZZZ9 ' EResGrp=' EXISTRGRP OF
     BINCONRS ' EResAss=' EXISTRASG OF BINCONRS
     ' EResDes=' EXISTRDSC OF BINCONRS ' ExistUsa=' CHARV
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     MOVE EXISTTYPE OF BINCONRS TO CODEV.
     MOVE 'BINCONRS' TO CHR8.
     MOVE 'EXISTTYPE' TO CHR12.
     PERFORM XCV2CH
     MOVE CHARV TO CHAR12.
     MOVE EXISTASGOVR OF BINCONRS TO CODEV.
     MOVE 'BINCONRS' TO CHR8.
     MOVE 'EXISTASGOVR' TO CHR12.
     PERFORM XCV2CH
     MOVE SPACES TO W-TEXT.
     STRING '  ExistSGrp=' EXISTSGRP OF BINCONRS
     ' ExistSTyp=' CHAR12 ' ExistAssO=' CHARV
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     EXIT.
```

```
************************************************
* This subroutine displays BINCONSC information *
************************************************
 DISPBISC.
     MOVE W-BUFFER TO BINSTERR.
     MOVE ERROP OF BINCONSC TO PICZZZ9.
     MOVE ERRCODE OF BINCONSC TO PYCZZZ9.
     MOVE SPACES TO W-TEXT.
     STRING 'CMAS=' CMASNAME OF BINCONSC ' Plex='
     PLEXNAME OF BINCONSC ' EOp=' PICZZZ9 ' ECode='
     PYCZZZ9 ' TScope=' TARGSCOPE OF BINCONSC
     ' TAssgn=' TARGRASG OF BINCONSC
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     MOVE SPACES TO W-TEXT.
     STRING '  TDesc=' TARGRDSC OF BINCONSC ' RScope='
     RELSCOPE OF BINCONSC ' RAssgn=' RELRASG OF BINCONSC
     ' RDesc=' RELRDSC OF BINCONSC ' CSys=' CICSNAME OF
     BINCONSC
     DELIMITED BY SIZE INTO W-TEXT END-STRING.
     PERFORM SCRNLOG2.
     EXIT.


************************************************************
* This subroutine converts coded value to character string *
************************************************************
 XCV2CH.
* Use new thread for TRANSLATE
     EXEC CPSM CONNECT
               VERSION('0140')
               THREAD(W-FBTTKN)
               RESPONSE(W-RESPONSE)
               REASON(W-REASON)
     END-EXEC.

* Translate internal coded value to character value
     EXEC CPSM TRANSLATE
               OBJECT(CHR8)
               ATTRIBUTE(CHR12)
               FROMCV(CODEV) TOCHAR(CHARV)
               THREAD(W-FBTTKN)
               RESPONSE(W-RESPONSE)
               REASON(W-REASON)
     END-EXEC.
     EXIT.

*-------------------------------------------------------------*
```

```
*    PROCESSING FOR API FAILURES.                              *
*--------------------------------------------------------------*
 NO-CONNECT.
     MOVE 'ERROR CONNECTING TO API.' TO W-MSG-TEXT.
     GO TO SCRNLOG.
 NO-CREATE.
     MOVE 'ERROR CREATING DEFINITION.' TO W-MSG-TEXT.
     GO TO SCRNLOG.
 NO-GET.
     MOVE 'ERROR GETTING RESOURCE TABLE.' TO W-MSG-TEXT.
     GO TO SCRNLOG.
 NO-INSTALL.
     MOVE 'ERROR INSTALLING RESULT SET.' TO W-MSG-TEXT.
     GO TO SCRNLOG.
 NO-TRANSLATE.
     MOVE 'ERROR TRANSLATING ATTRIBUTE.' TO W-MSG-TEXT.
     GO TO SCRNLOG.
 SCRNLOG.
*    DISPLAY W-MSG-TEXT.
     EXEC CICS SEND FROM(W-MSG-TEXT) LENGTH(81) WAIT END-EXEC.
     MOVE W-RESPONSE TO PICZZZ9A.
     MOVE W-REASON TO PICZZZ9B.
     STRING 'RESPONSE=' DELIMITED BY SIZE
            PICZZZ9A DELIMITED BY SIZE
            ' REASON= ' DELIMITED BY SIZE
            PICZZZ9B DELIMITED BY SIZE
            INTO W-MSG-TEXT.
 SCRNLOG2.
*    DISPLAY W-MSG-TEXT.
     EXEC CICS SEND FROM(W-MSG-TEXT) LENGTH(81) WAIT END-EXEC.

 ENDIT.
*--------------------------------------------------------------*
*    TERMINATE API CONNECTION.                                 *
*--------------------------------------------------------------*
     EXEC CPSM TERMINATE RESPONSE(W-RESPONSE) REASON(W-REASON)
     END-EXEC.
     EXEC CICS RETURN END-EXEC.
*    GOBACK
     EXIT.
 EYULAPI4-END.
```

The COBOL version of EYUxAPI4 is written for the CICS environment and can be converted to run in the MVS/ESA batch environment by commenting the EXEC CICS SEND commands, and uncommenting the preceding language specific output statement.

# Bibliography

## The CICS Transaction Server for z/OS library

The published information for CICS Transaction Server for z/OS is delivered in the following forms:

**The CICS Transaction Server for z/OS Information Center**
The CICS Transaction Server for z/OS Information Center is the primary source of user information for CICS Transaction Server. The Information Center contains:

- Information for CICS Transaction Server in HTML format.
- Licensed and unlicensed CICS Transaction Server books provided as Adobe Portable Document Format (PDF) files. You can use these files to print hardcopy of the books. For more information, see "PDF-only books."
- Information for related products in HTML format and PDF files.

One copy of the CICS Information Center, on a CD-ROM, is provided automatically with the product. Further copies can be ordered, at no additional charge, by specifying the Information Center feature number, 7014.

Licensed documentation is available only to licensees of the product. A version of the Information Center that contains only unlicensed information is available through the publications ordering system, order number SK3T-6945.

**Entitlement hardcopy books**
The following essential publications, in hardcopy form, are provided automatically with the product. For more information, see "The entitlement set."

## The entitlement set

The entitlement set comprises the following hardcopy books, which are provided automatically when you order CICS Transaction Server for z/OS, Version 3 Release 1:

*Memo to Licensees*, GI10-2559
*CICS Transaction Server for z/OS Program Directory*, GI10-2586
*CICS Transaction Server for z/OS Release Guide*, GC34-6421
*CICS Transaction Server for z/OS Installation Guide*, GC34-6426
*CICS Transaction Server for z/OS Licensed Program Specification*, GC34-6608

You can order further copies of the following books in the entitlement set, using the order number quoted above:

*CICS Transaction Server for z/OS Release Guide*
*CICS Transaction Server for z/OS Installation Guide*
*CICS Transaction Server for z/OS Licensed Program Specification*

## PDF-only books

The following books are available in the CICS Information Center as Adobe Portable Document Format (PDF) files:

### CICS books for CICS Transaction Server for z/OS

**General**

*CICS Transaction Server for z/OS Program Directory*, GI10-2586
*CICS Transaction Server for z/OS Release Guide*, GC34-6421
*CICS Transaction Server for z/OS Migration from CICS TS Version 2.3*, GC34-6425

*CICS Transaction Server for z/OS Migration from CICS TS Version 1.3*, GC34-6423
*CICS Transaction Server for z/OS Migration from CICS TS Version 2.2*, GC34-6424
*CICS Transaction Server for z/OS Installation Guide*, GC34-6426

**Administration**

*CICS System Definition Guide*, SC34-6428
*CICS Customization Guide*, SC34-6429
*CICS Resource Definition Guide*, SC34-6430
*CICS Operations and Utilities Guide*, SC34-6431
*CICS Supplied Transactions*, SC34-6432

**Programming**

*CICS Application Programming Guide*, SC34-6433
*CICS Application Programming Reference*, SC34-6434
*CICS System Programming Reference*, SC34-6435
*CICS Front End Programming Interface User's Guide*, SC34-6436
*CICS C++ OO Class Libraries*, SC34-6437
*CICS Distributed Transaction Programming Guide*, SC34-6438
*CICS Business Transaction Services*, SC34-6439
*Java Applications in CICS*, SC34-6440
*JCICS Class Reference*, SC34-6001

**Diagnosis**

*CICS Problem Determination Guide*, SC34-6441
*CICS Messages and Codes*, GC34-6442
*CICS Diagnosis Reference*, GC34-6899
*CICS Data Areas*, GC34-6902
*CICS Trace Entries*, SC34-6443
*CICS Supplementary Data Areas*, GC34-6905

**Communication**

*CICS Intercommunication Guide*, SC34-6448
*CICS External Interfaces Guide*, SC34-6449
*CICS Internet Guide*, SC34-6450

**Special topics**

*CICS Recovery and Restart Guide*, SC34-6451
*CICS Performance Guide*, SC34-6452
*CICS IMS Database Control Guide*, SC34-6453
*CICS RACF Security Guide*, SC34-6454
*CICS Shared Data Tables Guide*, SC34-6455
*CICS DB2 Guide*, SC34-6457
*CICS Debugging Tools Interfaces Reference*, GC34-6908

## CICSPlex SM books for CICS Transaction Server for z/OS

**General**

*CICSPlex SM Concepts and Planning*, SC34-6459
*CICSPlex SM User Interface Guide*, SC34-6460
*CICSPlex SM Web User Interface Guide*, SC34-6461

**Administration and Management**

*CICSPlex SM Administration*, SC34-6462
*CICSPlex SM Operations Views Reference*, SC34-6463
*CICSPlex SM Monitor Views Reference*, SC34-6464
*CICSPlex SM Managing Workloads*, SC34-6465
*CICSPlex SM Managing Resource Usage*, SC34-6466
*CICSPlex SM Managing Business Applications*, SC34-6467

**Programming**

*CICSPlex SM Application Programming Guide*, SC34-6468
*CICSPlex SM Application Programming Reference*, SC34-6469

**Diagnosis**

*CICSPlex SM Resource Tables Reference*, SC34-6470
*CICSPlex SM Messages and Codes*, GC34-6471
*CICSPlex SM Problem Determination*, GC34-6472

### CICS family books

**Communication**

*CICS Family: Interproduct Communication*, SC34-6473
*CICS Family: Communicating from CICS on System/390*, SC34-6474

### Licensed publications

The following licensed publications are not included in the unlicensed version of the Information Center:

*CICS Diagnosis Reference*, GC34-6899
*CICS Data Areas*, GC34-6902
*CICS Supplementary Data Areas*, GC34-6905
*CICS Debugging Tools Interfaces Reference*, GC34-6908

## Other CICS books

The following publications contain further information about CICS, but are not provided as part of CICS Transaction Server for z/OS, Version 3 Release 1.

| | |
|---|---|
| *Designing and Programming CICS Applications* | SR23-9692 |
| *CICS Application Migration Aid Guide* | SC33-0768 |
| *CICS Family: API Structure* | SC33-1007 |
| *CICS Family: Client/Server Programming* | SC33-1435 |
| *CICS Transaction Gateway for z/OS Administration* | SC34-5528 |
| *CICS Family: General Information* | GC33-0155 |
| *CICS 4.1 Sample Applications Guide* | SC33-1173 |
| *CICS/ESA 3.3 XRF Guide* | SC33-0661 |

## Determining if a publication is current

IBM regularly updates its publications with new and changed information. When first published, both hardcopy and BookManager® softcopy versions of a publication are usually in step. However, due to the time required to print and distribute hardcopy books, the BookManager version is more likely to have had last-minute changes made to it before publication.

Subsequent updates will probably be available in softcopy before they are available in hardcopy. This means that at any time from the availability of a release, softcopy versions should be regarded as the most up-to-date.

For CICS Transaction Server books, these softcopy updates appear regularly on the *Transaction Processing and Data Collection Kit* CD-ROM, SK2T-0730-xx. Each reissue of the collection kit is indicated by an updated order number suffix (the -xx part). For example, collection kit SK2T-0730-06 is more up-to-date than SK2T-0730-05. The collection kit is also clearly dated on the cover.

Updates to the softcopy are clearly marked by revision codes (usually a # character) to the left of the changes.

# Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully.

You can perform most tasks required to set up, run, and maintain your CICSPlex SM system in one of these ways:

- using a 3270 emulator connected to CICSPlex SM
- using a 3270 emulator logged on to CICS
- using a 3270 emulator logged on to TSO
- using a 3270 emulator as an MVS system console
- using the CICSPlex SM web user interface.

IBM Personal Communications (Version 5.0.1 for Windows 95, Windows 98, Windows NT and Windows 2000; version 4.3 for OS/2) provides 3270 emulation with accessibility features for people with disabilities. You can use this product to provide the accessibility features you need in your CICSPlex SM system.

# Index

## A



## B



## C

## D



## E



## F



## G



## I



## L

# M



# N



# O



# P



# R

## S



## T



## U



## V



## W



## X

# Notices

This information was developed for products and services offered in the U.S.A. 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 (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

**The following paragraph does not apply in 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 publication 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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

## Sample programs

This publication contains sample programs. Permission is hereby granted to copy and store the sample programs into a data processing machine and to use the stored copies for internal study and instruction only. No permission is granted to use the sample programs for any other purpose.

## Programming interface information

This book is intended to help you write application programs using the CICSPlex SM application programming interface (API). This book documents General-use Programming Interface and Associated Guidance Information provided by CICSPlex SM.

General-use programming interfaces allow the customer to write programs that obtain the services of CICSPlex SM.

## Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Microsoft and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other product and service names might be trademarks of IBM or other companies.

# Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To ask questions, make comments about the functions of IBM products or systems, or to request additional publications, contact your IBM representative or your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  IBM United Kingdom Limited
  User Technologies Department (MP095)
  Hursley Park
  Winchester
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–816151
  - From within the U.K., use 01962–816151
- Electronically, use the appropriate network ID:
  - IBMLink: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Product Number: 5655-M15

SC34-6468-04

Spine information:

CICS TS for z/OS

CICSPlex SM Application Programming Guide

Version 3
Release 1