z/OS

MVS Programming: Callable Services for High-Level Languages

SA22-7613-01

z/OS

MVS Programming: Callable Services for High-Level Languages

SA22-7613-01

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page C-1.

Second Edition, September 2002

This is a major revision of SA22-7613-00.

This edition applies to Version 1 Release 4 of z/OS (5694-A01), to Version 1 Release 4 of z/OS.e (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

Order documents through your IBM® representative or the IBM branch office serving your locality. Documents are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405
FAX (Other Countries):
Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: http://www.ibm.com/servers/eserver/zseries/zos/webqs.html

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this document
- Page number or topic related to your comment

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

Contents

Part 1. Window Services

Part 2. Reference Pattern Services

Part 3. Global Resource Serialization Latch Manager Services

Part 4. Resource Recovery Services (RRS)

Part 5. Other Callable Services

Part 6. Appendixes

Figures

Tables

About this document

This document supports z/OS (5694) and z/OS.e (5655-G52).

Callable services are for use by any program coded in C, COBOL, FORTRAN, Pascal, or PL/I — this document refers to programs written in these languages as high-level language (HLL) programs. Callable services enable HLL programs to use specific MVS™ services by issuing program CALLs.

Who should use this document

This document is for programmers who code in C, COBOL, FORTRAN, Pascal, or PL/I and want to use the callable services that MVS provides.

How to use this document

This document is one of the set of programming documents for MVS. This set describes how to write programs in assembler language or high-level languages, such as C, FORTRAN, and COBOL. For more information about the content of this set of documents, see *z/OS Information Roadmap*.

Where to find more information

Where necessary, this document references information in other documents, using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

Information updates on the web

For the latest information updates that have been provided in PTF cover letters and Documentation APARs for z/OS™ and z/OS.e, see the online document at:

`http://www.s390.ibm.com:80/bookmgr-cgi/bookmgr.cmd/BOOKS/ZIDOCMST/CCONTENTS`

This document is updated weekly and lists documentation changes before they are incorporated into z/OS publications.

Accessing z/OS licensed documents on the Internet

z/OS licensed documentation is available on the Internet in PDF format at the IBM Resource Link™ Web site at:

`http://www.ibm.com/servers/resourcelink`

Licensed documents are available only to customers with a z/OS license. Access to these documents requires an IBM Resource Link user ID and password, and a key code. With your z/OS order you received a Memo to Licensees, (GI10-0671), that includes this key code. [1]

To obtain your IBM Resource Link user ID and password, log on to:

`http://www.ibm.com/servers/resourcelink`

1. z/OS.e™ customers received a Memo to Licensees, (GI10-0684) that includes this key code.

To register for access to the z/OS licensed documents:

1. Sign in to Resource Link using your Resource Link user ID and password.
2. Select **User Profiles** located on the left-hand navigation bar.

Note: You cannot access the z/OS licensed documents unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

Printed licensed documents are not available from IBM.

You can use the PDF format on either **z/OS Licensed Product Library CD-ROM** or IBM Resource Link to print licensed documents.

Using LookAt to look up message explanations

LookAt is an online facility that allows you to look up explanations for most messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can access LookAt from the Internet at:

`http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/`

or from anywhere in z/OS where you can access a TSO/E command line (for example, TSO/E prompt, ISPF, z/OS UNIX System Services running OMVS). You can also download code from the *z/OS Collection* (SK3T-4269) and the LookAt Web site that will allow you to access LookAt from a handheld computer (Palm Pilot VIIx suggested).

To use LookAt as a TSO/E command, you must have LookAt installed on your host system. You can obtain the LookAt code for TSO/E from a disk on your *z/OS Collection* (SK3T-4269) or from the **News** section on the LookAt Web site.

Some messages have information in more than one document. For those messages, LookAt displays a list of documents in which the message appears.

Summary of changes

Summary of changes
for SA22-7613-01
z/OS Version 1 Release 4

This document contains information previously presented in *z/OS MVS Programming: Callable Services for HLL*, SA22-7613-00, which supports z/OS Version 1 Release 1.

New information

- An appendix listing general use C/C++ header files has been added.
- Information is added to indicate this document supports z/OS.e.
- An appendix with z/OS product accessibility information has been added.

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Starting with z/OS V1R2, you may notice changes in the style and structure of some content in this document—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

Summary of changes
for SA22-7613-00
z/OS Version 1 Release 1

This document contains information also presented in *OS/390® MVS Programming: Callable Services for High-Level Languages*.

Part 1. Window Services

Chapter 1. Introduction to Window Services

Window services allow HLL programs to:
- Read or update an existing permanent data object
- Create and save a new permanent data object
- Create and use a temporary data object

Window services enable your program to access data objects without your program performing any input or output (I/O) operations. All your program needs to do is issue a CALL to the appropriate service program. The service program performs any I/O operations that are required to make the data object available to your program. When you want to update or save a data object, window services again perform any required I/O operations.

Permanent Data Objects

A **permanent data object** is a virtual storage access method (VSAM) linear data set that resides on DASD. (This type of data set is also called a data-in-virtual object.) You can read data from an existing permanent object and also update the content of the object. You can create a new permanent object and when you are finished, save it on DASD. Because you can save this type of object on DASD, window services calls it a **permanent object**. Window services can handle very large permanent objects that contain as many as 4 gigabytes (four billion bytes).

Note: Installations whose FORTRAN programs used data-in-virtual objects prior to MVS/SP™ 3.1.0 had to write an Assembler language interface program to allow the FORTRAN program to invoke the data-in-virtual program. Window services eliminates the need for this interface program.

Temporary Data Objects

A **temporary data object** is an area of expanded storage that window services provides for your program. You can use this storage to hold temporary data, such as intermediate results of a computation, instead of using a DASD workfile. Or you might use the storage area as a temporary buffer for data that your program generates or obtains from some other source. When you finish using the storage area, window services deletes it. Because you cannot save the storage area, window services calls it a temporary object. Window services can handle very large temporary objects that contain as many as 16 terabytes (16 trillion bytes).

Structure of a Data Object

Think of a data object as a contiguous string of bytes organized into blocks, each 4096 bytes long. The first block contains bytes 0 to 4095 of the object, the second block contains bytes 4096 to 8191, and so forth.

Your program references data in the object by identifying the block or blocks that contain the desired data. Window services makes the blocks available to your program by mapping a **window** in your program storage to the blocks. A window is a storage area that your program provides and makes known to window services. **Mapping** the window to the blocks means that window services makes the data from those blocks available in the window when you reference the data. You can map a window to all or part of a data object depending on the size of the object and

the size of the window. You can examine or change data that is in the window by using the same instructions that you use to examine or change any other data in your program storage.

The following figure shows the structure of a data object and shows a window mapped to two of the object's blocks.

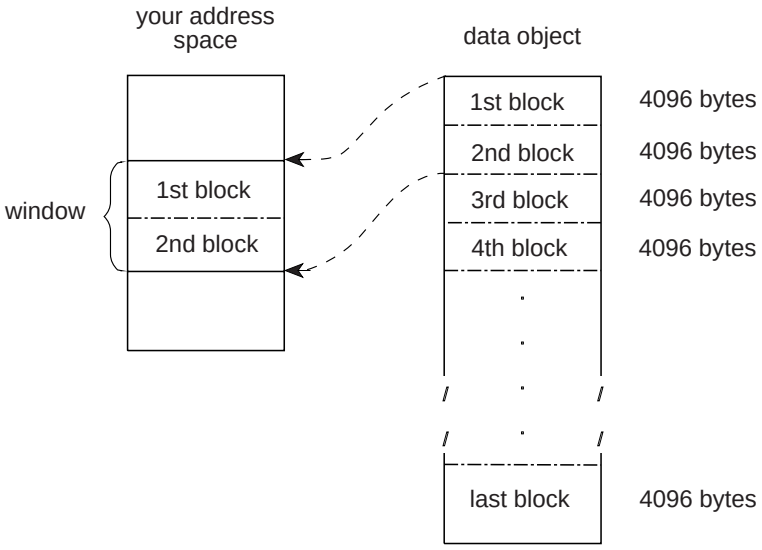


Figure 1-1. Structure of a Data Object

What Does Window Services Provide?

Window services allows you to view and manipulate data objects in a number of ways. You can have access to one or more data objects at the same time. You can also define multiple windows for a given data object. You can then view a different part of the object through each window. Before you can access any data object, you must request access from window services.

When you request access to a permanent data object, you must indicate whether you want a scroll area. A **scroll area** is an area of expanded storage that window services obtains and maps to the permanent data object. You can think of the permanent object as being available in the scroll area. When you request a view of the object, window services maps the window to the scroll area. If you do not request a scroll area, window services maps the window directly to the object on DASD.

A scroll area enables you to save interim changes to a permanent object without changing the object on DASD. Also, when your program accesses a permanent object through a scroll area, your program might attain better performance than it would if the object were accessed directly on DASD.

When you request a temporary object, window services provides an area of expanded storage. This area of expanded storage is the temporary data object. When you request a view of the object, window services maps the window to the temporary object. Window services initializes a temporary object to binary zeroes.

Notes:

1. Window services does not transfer data from the object on DASD, from the scroll area, or from the temporary object until your program references the data. Then window services transfers those blocks.
2. The expanded storage that window services uses for a scroll area or for a temporary object is called a **hiperspace**. A hiperspace is a range of contiguous virtual storage addresses that a program can indirectly access through a window in the program's virtual storage. Window services uses as many hiperspaces as needed to contain the data object.

The Ways That Window Services Can Map an Object

Window services can map a data object a number of ways. The following examples show how window services can:

- Map a permanent object that has no scroll area
- Map a permanent object that has a scroll area
- Map a temporary object
- Map an object to multiple windows
- Map multiple objects

Example 1 — Mapping a Permanent Object that has no Scroll Area

If a permanent object has no scroll area, window services maps the object from DASD directly to your window. In this example, your window provides a view of the first and second blocks of an object.

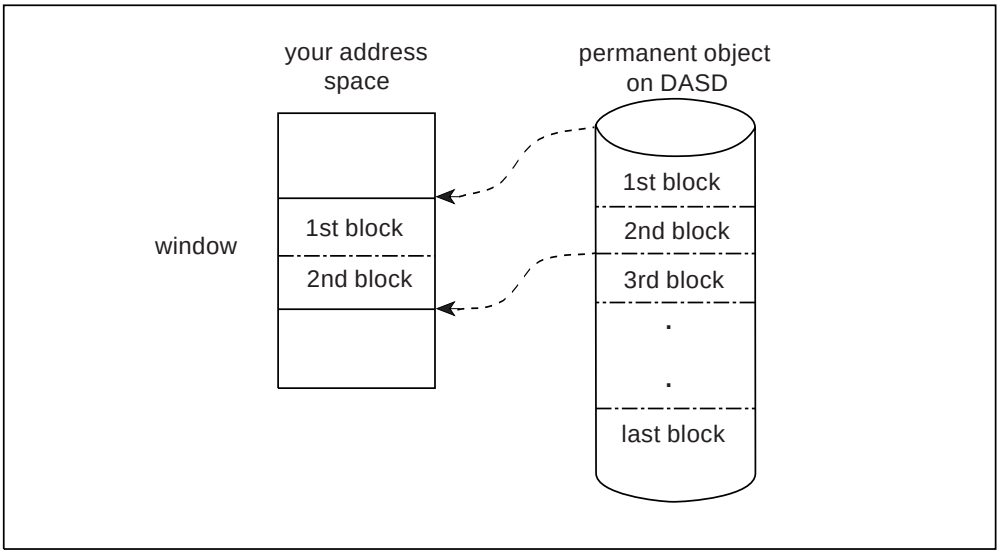


Figure 1-2. Mapping a Permanent Object That Has No Scroll Area

Example 2 — Mapping a Permanent Object that has a Scroll Area

If the object has a scroll area, window services maps the object from DASD to the scroll area. Window services then maps the blocks that you wish to view from the scroll area to your window. In this example, your window provides a view of the third and fourth blocks of an object.

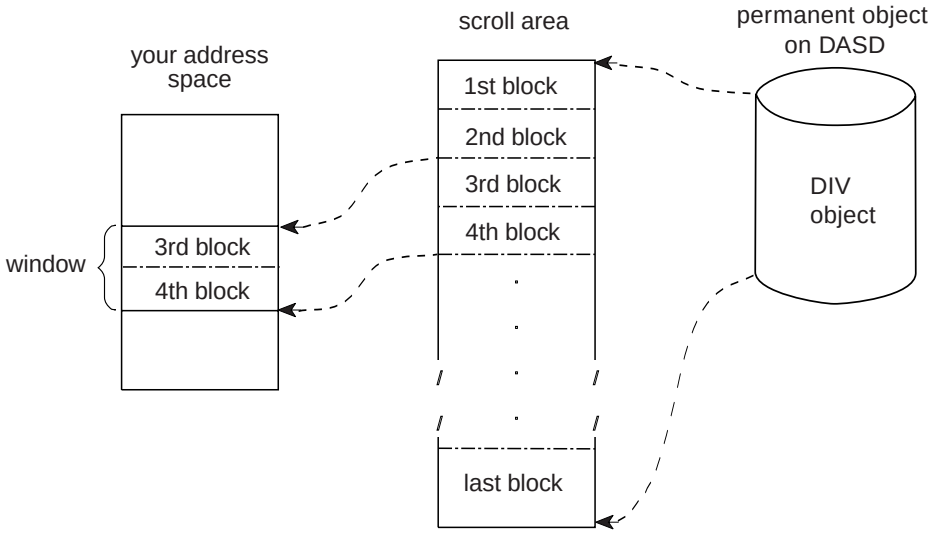


Figure 1-3. Mapping a Permanent Object That Has a Scroll Area

Example 3 — Mapping a Temporary Object

Window services uses a hiperspace as a temporary object. In this example, your window provides a view of the first and second blocks of a temporary object.

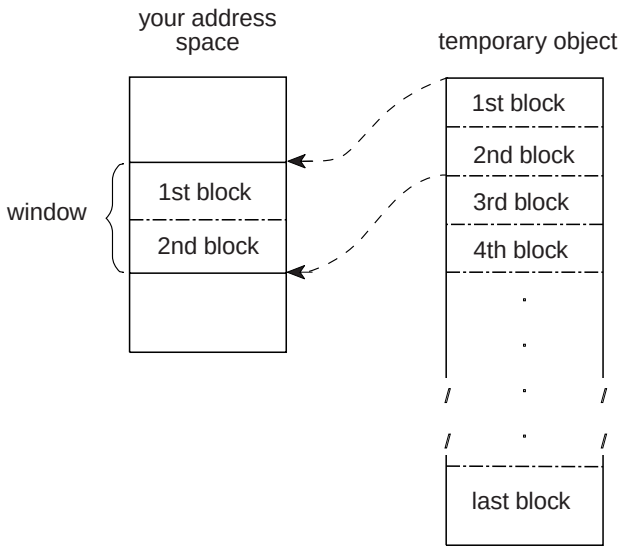


Figure 1-4. Mapping a Temporary Object

Example 4 — Mapping Multiple Windows® to an Object

Window services can map multiple windows to the same object. In this example, one window provides a view of the second and third blocks of an object, and a second window provides a view of the last block.

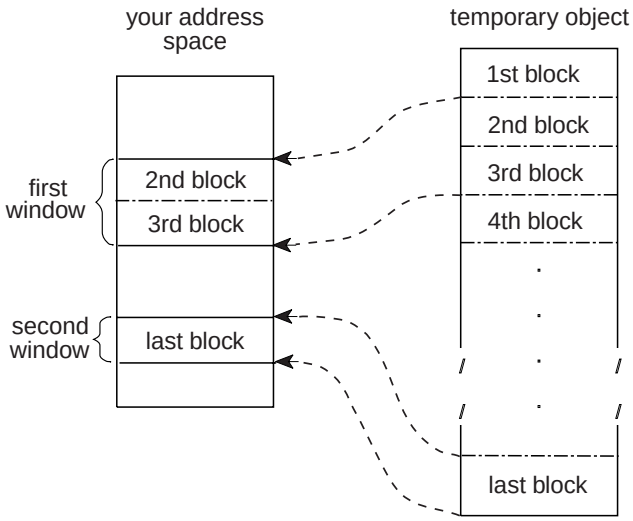


Figure 1-5. Mapping an Object to Multiple Windows

Example 5 — Mapping Multiple Objects

Window services can map windows in the same address space to multiple objects. The objects can be temporary objects, permanent objects, or a combination of temporary and permanent objects. In this example, one window provides a view of the second block of a temporary object, and a second window provides a view of the fourth and fifth blocks of a permanent object.

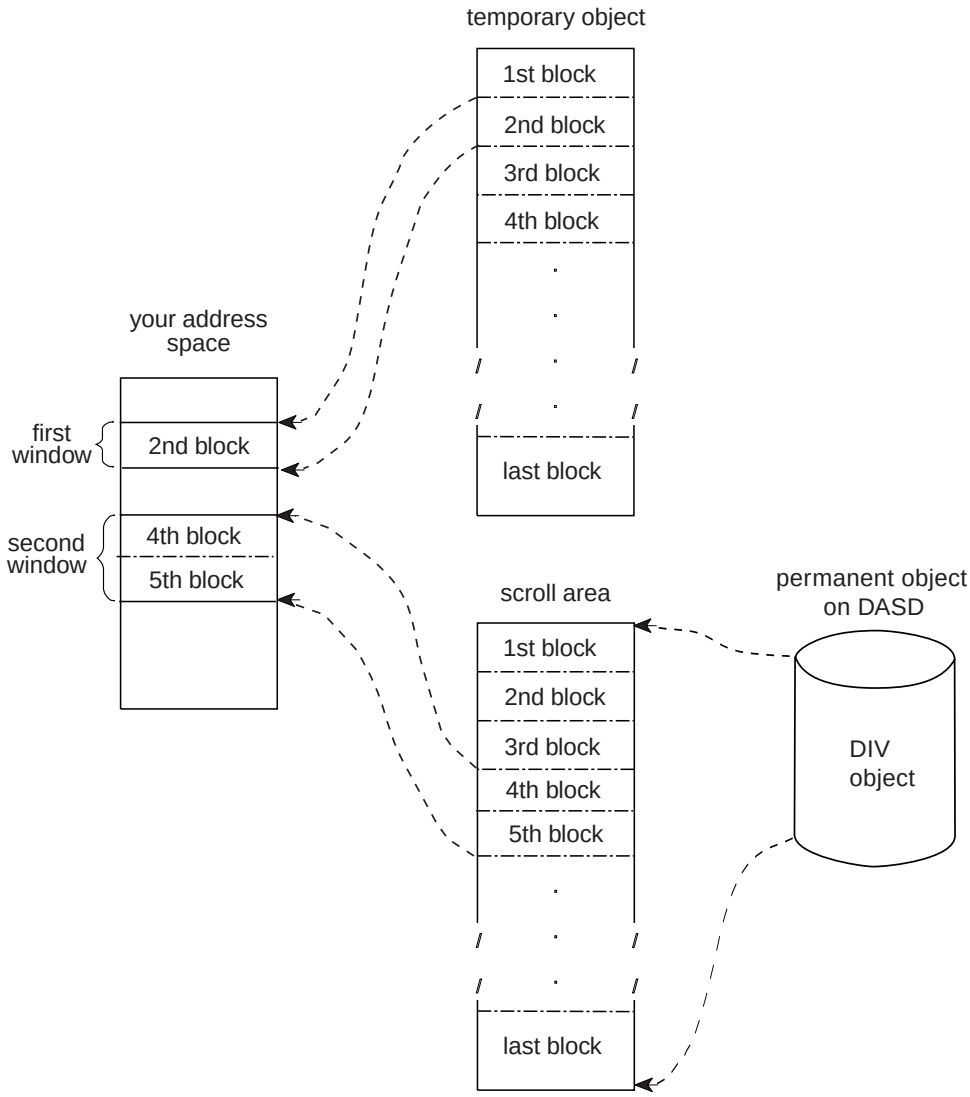


Figure 1-6. Mapping Multiple Objects

Access to Permanent Data Objects

When you have access to a permanent data object, you can:

- **View the object through one or more windows —** Depending on the object size and the window size, a single window can view all or part of a permanent object. If you define multiple windows, each window can view a different part of the object. For example, one window might view the first block of the permanent object and another window might view the second block. You can also have several windows view the same part of the object or have views in multiple windows overlap. For example, one window might view the first and second blocks of a data object while another window views the second and third blocks.
- **Change data that appears in a window —** You can examine or change data that is in a window by using the same instructions you use to examine or change any other data in your program's storage. These changes do not alter the object on DASD or in the scroll area.

- **Save interim changes in a scroll area —** After changing data in a window, you can have window services save the changed blocks in a scroll area, if you have requested one. Window services replaces blocks in the scroll area with corresponding changed blocks from the window. Saving changes in the scroll area does not alter the object on DASD or alter data in the window.
- **Refresh a window or the scroll area —** After you change data in a window or save changes in the scroll area, you may discover that you no longer need those changes. In that case, you can have window services refresh the changed data. To refresh the window or the scroll area, window services replaces changed data with data from the object as it appears on DASD.
- **Replace the view in a window —** After you finish using data that is in a window, you can have window services replace the view in the window with a different view of the object. For example, if you are viewing the third, fourth, and fifth blocks of an object and are finished with those blocks, you might have window services replace that view with a view of the sixth, seventh, and eighth blocks.
- **Update the object on DASD —** If you have changes available in a window or in the scroll area, you can save the changes on DASD. Window services replaces blocks on DASD with corresponding changed blocks from the window and the scroll area. Updating an object on DASD does not alter data in the window or in the scroll area.

Access to Temporary Data Objects

When you have access to a temporary data object, you can:

- **View the object through one or more windows —** Depending on the object size and the window size, a single window can view all or part of a temporary object. If you define multiple windows, each window can view a different part of the object. For example, one window might view the first block of the temporary object and another window might view the second block. Unlike a permanent object, however, you cannot define multiple windows that have overlapping views of a temporary object.
- **Change data that appears in a window —** This function is the same for a temporary object as it is for a permanent object: you can examine or change data that is in a window by using the same instructions you use to examine or change any other data in your address space.
- **Update the temporary object —** After you have changed data in a window, you can have window services update the object with those changes. Window services replaces blocks in the object with corresponding changed blocks from the window. The data in the window remains as it was.
- **Refresh a window or the object —** After you change data in a window or save changes in the object, you may discover that you no longer need those changes. In that case, you can have window services refresh the changed data. To refresh the window or the object, window services replaces changed data with binary zeroes.
- **Replace the view in a window —** After you finish using data that is in a window, you can have window services replace the view in the window with a different view of the object. For example, if you are viewing the third, fourth, and fifth blocks of an object and are finished with those blocks, you might have window services replace that view with a view of the sixth, seventh, and eighth blocks.

Chapter 2. Using Window Services

To use, create, or update a data object, you call a series of programs that window services provides. These programs enable you to:

- Access an existing object, create and save a new permanent object, or create a temporary object
- Obtain a scroll area where you can make interim changes to a permanent object
- Define windows and establish views of an object in those windows
- Change or terminate the view in a window
- Update a scroll area or a temporary object with changes you have made in a window
- Refresh changes that you no longer need in a window or a scroll area
- Update a permanent object on DASD with changes that are in a window or a scroll area
- Terminate access to an object

The window services programs that you call and the sequence in which you call them depends on your use of the data object.

The first step in using any data object is to gain access to the object. To gain access, call CSRIDAC. The object can be an existing permanent object, or a new permanent or temporary object you want to create. For a permanent object, you can request an optional scroll area. A scroll area enables you to make interim changes to an object's data without affecting the data on DASD. When CSRIDAC grants access, it provides an object identifier that identifies the object. Use that identifier to identify the object when you request other services from window services.

After obtaining access to an object, define one or more windows and establish views of the object in those windows. To establish a view of an object, tell window services which blocks you want to view and in which windows. You can view multiple objects and multiple parts of each object at the same time. To define windows and establish views, call CSRVIEW or CSREVW. After establishing a view, you can examine or change data that is in the window using the same instructions you use to examine or change other data in your program's storage.

After making changes to the part of an object that is in a window, you will probably want to save those changes. How you save changes depends on whether the object is permanent, is temporary, or has a scroll area.

If the object is permanent and has a scroll area, you can save changes in the scroll area without affecting the object on DASD. Later, you can update the object on DASD with changes saved in the scroll area. If the object is permanent and has no scroll area, you can update it on DASD with changes that are in a window. If the object is temporary, you can update it with changes that are in a window. To update an object on DASD, call CSRSAVE. To update a temporary object or a scroll area, call CSRSCOT.

After making changes in a window and possibly saving them in a scroll area or using them to update a temporary object, you might decide that you no longer need those changes. In this case, you can refresh the changed blocks. After refreshing a block of a permanent object or a scroll area to which a window is mapped, the refreshed block contains the same data that the corresponding block contains on

DASD. After refreshing a block of a temporary object to which a window is mapped, the block contains binary zeroes. To refresh a changed block, call CSRREFR.

After finishing with a view in a window, you can use the same window to view a different part of the object or to view a different object. Before changing the view in a window, you must terminate the current view. If you plan to view a different part of the same object, terminate the current view by calling CSRVIEW. If you plan to view a different object or will not reuse the window, you can terminate the view by calling CSRIDAC.

When you finish using a data object, terminate access to the object by calling CSRIDAC.

The following restrictions apply to using window services:

1. When you attach a new task, you cannot pass ownership of a mapped virtual storage window to the new task. That is, you cannot use the ATTACH or ATTACHX keywords GSPV and GSPL to pass the mapped virtual storage.
2. While your program is in cross-memory mode, your program cannot invoke data-in-virtual services; however, your program can reference and update data in a mapped virtual storage window.
3. The task that obtains the ID (through DIV IDENTIFY) is the only one that can issue other DIV services for that ID.
4. When you identify a data-in-virtual object using the IDENTIFY service, you cannot request a checkpoint until you invoke the corresponding UNIDENTIFY service.

This chapter explains how to do the previously described functions and contains the following topics:

Obtaining Access to a Data Object

To obtain access to a permanent or temporary data object, call CSRIDAC. Indicate that you want to access an object by specifying BEGIN as the value for *op_type*. For a description of the CSRIDAC parameters and return codes, see “CSRIDAC — Request or Terminate Access to a Data Object” on page 3-5.

Identifying the Object

You must identify the data object you wish to access. How you identify the object depends on whether the object is permanent or temporary.

Permanent Object

For a permanent object, *object_name* and *object_type* work together. For *object_name* you have a choice: specify either the data set name of the object or the DDNAME to which the object is allocated. The *object_type* parameter must then indicate whether *object_name* is a DDNAME or a data set name:

- If *object_name* is a DDNAME, specify DDNAME as the value for *object_type*.
- If *object_name* is a data set name, specify DSNAME as the value for *object_type*.

If you specify DSNAME for *object_type*, indicate whether the object already exists or whether window services is to create it:

- If the object already exists, specify OLD as the value for *object_state*.
- If window services is to create the object, specify NEW as the value for *object_state*.

Requirement for NEW objects

If you specify NEW as the value for *object_state*, your system must include MVS/Data Facility Product. (MVS/DFP™) 3.1.0 and SMS must be active.

Temporary Object

To identify a temporary object, specify TEMPSPACE as the value for *object_type*. Window services assumes that a temporary object is new and ignores the value that you specify for *object_state*.

Specifying the Object's Size

If the object is permanent and new or is temporary, you must tell window services the size of the object. You specify object size through the *object_size* parameter. The size specified becomes the maximum size that window services will allow for that object. You express the size as a number of 4096-byte blocks. If the number of bytes in the object is not an exact multiple of 4096, round *object_size* to the next whole number. For example:

- If the object size is to be less than 4097 bytes, specify 1.
- If the object size is 5000 bytes, specify 2.
- If the object size is 410,000 bytes, specify 101.

Specifying the Type of Access

For an existing (OLD) permanent object, you must specify how you intend to access the object. You specify your intentions through the *access_mode* parameter:

- If you intend to only read the object, specify READ for *access_mode*.
- If you intend to update the object, specify UPDATE for *access_mode*.

For a new permanent object and for a temporary object, window services assumes you will update the object and ignores the value you specify for *access_mode*.

Obtaining a Scroll Area

A scroll area is storage that window services provides for your use. This storage is outside your program's storage area and is accessible only through window services.

For a permanent object, a scroll area is optional. A scroll area allows you to make interim changes to a permanent object without altering the object on DASD. Later, if you want, you can update the object on DASD with the interim changes. A scroll area might also improve performance when your program accesses a permanent object.

For a temporary object, the scroll area is the object. Therefore, for a temporary object, a scroll area is required.

To indicate whether you want a scroll area, provide the appropriate value for *scroll_area*:

- To request a scroll area, supply a value of YES. YES is required for a temporary object.
- To indicate you do not want a scroll area, supply a value of NO.

Defining a View of a Data Object

To view all or part of a data object, you must provide window services with information about the object and how you want to view it. You must provide window services with the following information:

- The object identifier
- Where the window is in your address space
- Window disposition — that is, whether window services is to initialize the window the first time you reference data in the window
- Whether you intend to reference blocks of data sequentially or randomly
- The blocks of data that you want to view
- Whether you want to extend the size of the object

To define a view of a data object, call CSRVIEW or CSREVW. Whether you use CSRVIEW or CSREVW depends on how you plan to reference the data. “Defining the Expected Reference Pattern” on page 2-5 describes the differences between the two services. Specify BEGIN on CSRVIEW or CSREVW as the type of operation. For descriptions of the CALL syntax and return codes from CSRVIEW or CSREVW, see “CSRVIEW — View an Object” on page 3-16 or “CSREVW — View an Object and Sequentially Access It” on page 3-1.

Identifying the Data Object

To identify the object you want to view, specify the object identifier as the value for *object_id*. Use the same value CSRIDAC returned in *object_id* when you requested access to the object.

Identifying a Window

You must identify the window through which you will view the object. The window is a virtual storage area in your address space. You are responsible for obtaining the storage, which must meet the following requirements:

- The storage must not be page fixed.
- Pages in the window must not be page loaded (must not be loaded by the PGLOAD macro).
- The storage must start on a 4K boundary and must be a multiple of 4096 bytes in length.

To identify the window, use the *window_name* parameter. The value supplied for *window_name* must be the symbolic name you assigned to the window storage area in your program.

Defining a window in this way provides one window through which you can view the object. To define multiple windows that provide simultaneous views of different parts of the object, see “Defining Multiple Views of an Object” on page 2-7.

Defining the Disposition of a Window's Contents

You must specify whether window services is to replace or retain the window contents. You do this by selecting either the replace or retain option. This option determines how window services handles the data that is in the window the first time you reference the data. You select the option by supplying a value of REPLACE or RETAIN® for *disposition*.

Replace Option

If you specify the replace option, the first time you reference a block to which a window is mapped, window services replaces the data in the window with corresponding data from the object. For example, assume you have requested a view of the first block of a permanent object and have specified the replace option. The first time you reference the window, window services replaces the data in the window with the first 4096 bytes (the first block) from the object.

If you have selected the replace option and then call CSRSAVE to update a permanent object, or call CSRSCOT to update a scroll area, or call CSRSCOT to update a temporary object, window services updates only the specified blocks that have changed and to which a window is mapped.

Select the replace option when you want to examine, use, or change data that is currently in an object.

Retain Option

If you select the retain option, window services retains data that is in the window. When you reference a block in the window the first time, the block contains the same data it contained before the reference.

When you select the retain option, window services considers all of the data in the window as changed. Therefore, if you call CSRSCOT to update a scroll area or a temporary object, or call CSRSAVE to update a permanent object, window services updates all of the specified blocks to which a window or scroll area are mapped.

Select the retain option when you want to replace data in an object without regard for the data that it currently contains. You also use the retain option when you want to initialize a new object.

Defining the Expected Reference Pattern

You must tell window services whether you intend to reference the blocks of an object sequentially or randomly. An intention to access randomly tells window services to bring one block (4096 bytes) of data into the window at a time. An intention to access sequentially tells window services to read more than one block into your window at one time. The performance gain is in having blocks of data already in central storage at the time the program needs to reference them. You specify the intent on either CSRVIEW or CSREVW, two services that differ on how to specify sequential access.

- CSRVIEW allows you a choice between random or sequential access.

  If you specify **random**, when you reference data that is not in your window, window services brings in one block — the one that contains the data your program references.

  If you specify **sequential**, when you reference data that is not in your window, window services transfers up to 16 blocks — the one that contains the data your program requests, plus the next 15 consecutive blocks. The number of consecutive blocks varies, depending on the size of the window and availability of central storage. Use CSRVIEW if one of the following is true:

  - You are going to access randomly.
  - You are going to access sequentially, and you are satisfied with a maximum of 16 blocks coming into the window at a time.
- CSREVW is for sequential access only. It allows you to specify the maximum number of consecutive blocks that window services brings into the window at one time. The number ranges from one block through 256 blocks. Use CSREVW if you want fewer than 16 blocks or more than 16 blocks at one time. Programs that benefit from having more than 16 blocks come into a window at one time reference data areas that are greater than one megabyte.

To specify the reference pattern on CSRVIEW, supply a value of SEQ or RANDOM for *usage*.

To specify the reference pattern on CSREVW, supply a number from 0 through 255 for *pfcount*. *pfcount* represents the number of blocks window services will bring into the window, in addition to the one that it always brings in.

Note that window services brings in multiple pages differently depending on whether your object is permanent or temporary and whether the system has had to move pages of your data from central storage to make those pages of central available for other programs. The rule is that SEQ on CSRVIEW and *pfcount* on CSREVW apply to:

- A **permanent object** when movement is from the object on DASD to central storage
- A **temporary object** when your program has scrolled the data out and references it again

SEQ and *pfcount* do not apply after the system has had to move data (either changed or unchanged) to auxiliary or expanded storage, and your program again references it, requiring the system to bring the data back into central storage.

End the view, whether established with CSRVIEW or CSREVW, with CSRVIEW END.

Identifying the Blocks You Want to View

To identify the blocks of data you want to view, use *offset* and *span*. The values you assign to *offset* and *span*, together, define a contiguous string of blocks that you want to view:

- The value assigned to *offset* specifies the relative block at which to start the view. An offset of 0 means the first block; an offset of 1 means the second block; an offset of 2 means the third block, and so forth.
- The value assigned to *span* specifies the number of blocks to view. A span of 1 means one block; a span of 2 means two blocks, and so forth. A span of 0 has special meaning: it means the view is to start at the specified offset and extend until the currently defined end of the object.

The following table shows examples of several *offset* and *span* combinations and the resulting view in the window.

Offset	Span	Resulting view in the window
0	0	view the entire object
0	1	view the first block only
1	0	view the second block through the last block
1	1	view the second block only

Offset	Span	Resulting view in the window
2	2	view the third and fourth blocks only

Extending the Size of a Data Object

You can use *offset* and *span* to extend the size of an object up to the previously defined maximum size for the object. You can extend the size of either permanent objects or temporary objects. For objects created through CSRIDAC, the value assigned to *object_size* defines the maximum allowable size. When you call CSRIDAC to gain access to an object, CSRIDAC returns a value in *high_offset* that defines the current size of the object.

For example, assume you have access to a permanent object whose maximum allowable size is four 4096-byte blocks. The object is currently two blocks long. If you define a window and specify an offset of 1 and a span of 2, the window contains a view of the second block and a view of a third block, which does not yet exist in the permanent object. When you reference the window, the content of the second block, as seen in the window, depends on the disposition you selected, replace or retain. The third block, as seen in the window, initially contains binary zeroes. If you later call CSRSAVE to update the permanent object with changes from the window, window services extends the size of the permanent object to three blocks by appending the new block of data to the object.

Defining Multiple Views of an Object

You might need to view different parts of an object at the same time. For a permanent object, you can define windows that have nonoverlapping views as well as windows that have overlapping views. For a temporary object, you can define windows that have only nonoverlapping views.

- A nonoverlapping view means that no two windows view the same block of the object. For example, a view is nonoverlapping when one window views the first and second blocks of an object and another window views the ninth and tenth blocks of the same object. Neither window views a common block.
- An overlapping view means that two or more windows view the same block of the object. For example, the view overlaps when the second window in the previous example views the second and third blocks. Both windows view a common block, the second block.

Nonoverlapping Views

To define multiple windows that have a nonoverlapping view, call CSRIDAC once to obtain the object identifier. Then call CSRVIEW or CSREVW once to define each window. On each call, specify the value BEGIN for *operation_type*, the same object identifier for *object_id*, and a different value for *window_name*. Define each window's view by specifying values for *offset* and *span* that create windows with nonoverlapping views.

Overlapping Views

To define multiple windows that have an overlapping view of a permanent object, define each window as though it were viewing a different object. That is, define each window under a different object identifier. To obtain the object identifiers, call CSRIDAC once for each identifier you need. Only one of the calls to CSRIDAC can specify an access mode of UPDATE. Other calls to CSRIDAC must specify an access mode of READ.

After calling CSRIDAC, call CSRVIEW or CSREVW once to define each window. On each call, specify the value BEGIN for the operation type, a different object identifier for *object_id*, and a different value for *window_name*. Define each window's view by specifying values for *offset* and *span* that create windows with the required overlapping views.

Saving Interim Changes to a Permanent Data Object

Window services allows you to save interim changes you make to a permanent object. You must have previously requested a scroll area for the object, however. You request a scroll area when you call CSRIDAC to gain access to the object. Window services saves changes by replacing blocks in the scroll area with corresponding changed blocks from a window. Saving changes in the scroll area does not alter the object on DASD.

After you have a view of the object and have made changes in the window, you can save those changes in the scroll area. To save changes in the scroll area, call CSRSCOT. For a description of the CSRSCOT parameters and return codes, see "CSRSCOT — Save Object Changes in a Scroll Area" on page 3-14.

To identify the object, you must supply an object identifier for *object_id*. The value supplied for *object_id* must be the same value CSRIDAC returned in *object_id* when you requested access to the object.

To identify the blocks in the object that you want to update, use *offset* and *span*. The values assigned to *offset* and *span*, together, define a contiguous string of blocks in the object:

- The value assigned to *offset* specifies the relative block at which to start. An offset of 0 means the first block; an offset of 1 means the second block; an offset of 2 means the third block, and so forth.
- The value assigned to *span* specifies the number of blocks to save. A span of 1 means one block; a span of 2 means two blocks, and so forth. A span of 0 has special meaning: it requests that window services save all changed blocks to which a window is mapped.

Window services replaces each block within the range specified by *offset* and *span* providing the block has changed and a window is mapped to the block.

Updating a Temporary Data Object

After making changes in a window to a temporary object, you can update the object with those changes. You must identify the object and must specify the range of blocks that you want to update. To be updated, a block must be mapped to a window and must contain changes in the window. Window services replaces each block within the specified range with the corresponding changed block from a window.

To update a temporary object, call CSRSCOT. For a description of the CSRSCOT parameters and return codes, see "CSRSCOT — Save Object Changes in a Scroll Area" on page 3-14.

To identify the object, you must supply an object identifier for *object_id*. The value you supply for *object_id* must be the same value CSRIDAC returned in *object_id* when you requested access to the object.

To identify the blocks in the object that you want to update, use *offset* and *span*. The values assigned to *offset* and *span*, together, define a contiguous string of blocks in the object:

- The value assigned to *offset* specifies the relative block at which to start. An offset of 0 means the first block; an offset of 1 means the second block; an offset of 2 means the third block, and so forth.
- The value assigned to *span* specifies the number of blocks to save. A span of 1 means one block; a span of 2 means two blocks, and so forth. A span of 0 has special meaning: it requests that window services update all changed blocks to which a window is mapped.

Window services replaces each block within the range specified by *offset* and *span* providing the block has changed and a window is mapped to the block.

Refreshing Changed Data

You can refresh blocks that are mapped to either a temporary object or to a permanent object. You must identify the object and specify the range of blocks you want to refresh. When you refresh blocks mapped to a temporary object, window services replaces, with binary zeros, all changed blocks that are mapped to the window. When you refresh blocks mapped to a permanent object, window services replaces specified changed blocks in a window or in the scroll area with corresponding blocks from the object on DASD.

To refresh an object, call CSRREFR. For a description of CSRREFR parameters and return codes, see “CSRREFR — Refresh an Object” on page 3-9.

To identify the object, you must supply an object identifier for *object_id*. The value supplied for *object_id* must be the same value CSRIDAC returned in *object_id* when you requested access to the object.

To identify the blocks of the object that you want to refresh, use *offset* and *span*. The values assigned to *offset* and *span*, together, define a contiguous string of blocks in the object:

- The value assigned to *offset* specifies the relative block at which to start. An offset of 0 means the first block; an offset of 1 means the second block; an offset of 2 means the third block, and so forth.
- The value assigned to *span* specifies the number of blocks to save. A span of 1 means one block; a span of 2 means two blocks, and so forth. A span of 0 has special meaning: it requests that window services refresh all changed blocks to which a window is mapped, or that have been saved in a scroll area.

Window services refreshes each block within the range specified by *offset* and *span* providing the block has changed and a window or a scroll area is mapped to the block. At the completion of the refresh operation, blocks from a permanent object that have been refreshed appear the same as the corresponding blocks on DASD. Refreshed blocks from a temporary object contain binary zeroes.

Updating a Permanent Object on DASD

You can update a permanent object on DASD with changes that appear in a window or in the object’s scroll area. You must identify the object and specify the range of blocks that you want to update.

To update an object, call CSRSAVE. For a description of theCSRSAVE parameters and return codes, see “CSRSAVE — Save Changes Made to a Permanent Object” on page 3-11.

To identify the object, you must supply an object identifier for *object_id*. The value you provide for *object_id* must be the same value CSRIDAC returned when you requested access to the object.

To identify the blocks of the object that you want to update, use *offset* and *span*. The values assigned to *offset* and *span*, together, define a contiguous string of blocks in the object:

- The value assigned to *offset* specifies the relative block at which to start. An offset of 0 means the first block; an offset of 1 means the second block; an offset of 2 means the third block, and so forth.
- The value assigned to *span* specifies the number of blocks to save. A span of 1 means one block; a span of 2 means two blocks, and so forth. A span of 0 has special meaning: it requests that window services update all changed blocks to which a window is mapped, or have been saved in the scroll area.

When There is a Scroll Area

When the object has a scroll area, window services first updates blocks in the scroll area with corresponding blocks from windows. To be updated, a scroll area block must be within the specified range, a window must be mapped to the block, and the window must contain changes. Window services next updates blocks on DASD with corresponding blocks from the scroll area. To be updated, a DASD block must be within the specified range and have changes in the scroll area. Blocks in the window remain unchanged.

When There is No Scroll Area

When there is no scroll area, window services updates blocks of the object on DASD with corresponding blocks from a window. To be updated, a DASD block must be within the specified range, mapped to a window, and have changes in the window. Blocks in the window remain unchanged.

Changing a View in a Window

To change the view in a window so you can view a different part of the same object or view a different object, you must first terminate the current view. To terminate the view, whether the view was mapped by CSRVIEW or CSREVW, call CSRVIEW and supply a value of END for *operation_type*. You must also identify the object, identify the window, identify the blocks you are currently viewing, and specify a disposition for the data that is in the window. For a description of CSRVIEW parameters and return codes, see “CSRVIEW — View an Object” on page 3-16.

To identify the object, supply an object identifier for *object_id*. The value supplied for *object_id* must be the value you supplied when you established the view.

To identify the window, supply the window name for *window_name*. The value supplied for *window_name* must be the same value you supplied when you established the view.

To identify the blocks you are currently viewing, supply values for *offset* and *span*. The values you supply must be the same values you supplied for *offset* and *span* when you established the view.

To specify a disposition for the data you are currently viewing, supply a value for *disposition*. The value determines what data will be in the window after the CALL to CSRVIEW completes.

- For a permanent object that has no scroll area:
  - To retain the data that is currently in the window, supply a value of RETAIN for *disposition*.
  - To discard the data that is currently in the window, supply a value of REPLACE for *disposition*. After the operation completes, the window contents are unpredictable.

    For example, assume that a window is mapped to one block of a permanent object that has no scroll area. The window contains the character string AAA.....A and the block to which the window is mapped contains BBB.....B. If you specify a value of RETAIN, upon completion of the CALL, the window still contains AAA.....A, and the mapped block contains BBB.....B. If you specify a value of REPLACE, upon completion of the CALL, the window contents are unpredictable and the mapped block still contains BBB.....B.
- For a permanent object that has a scroll area or for a temporary object:
  - To retain the data that is currently in the window, supply a value of RETAIN for *disposition*. CSRVIEW also updates the mapped blocks of the scroll area or temporary object so that they contain the same data as the window.
  - To discard the data that is currently in the window, supply a value of REPLACE for *disposition*. Upon completion of the operation, the window contents are unpredictable.

    For example, assume that a window is mapped to one block of a temporary object. The window contains the character string AAA.....A and the block to which the window is mapped contains BBB.....B. If you specify a value of RETAIN, upon completion of the CALL, the window still contains AAA.....A and the mapped block of the object also contains AAA.....A. If you specify a value of REPLACE, upon completion of the CALL, the window contents are unpredictable and the mapped block still contains BBB.....B.

CSRVIEW ignores the values you assign to the other parameters.

When you terminate the view of an object, the type of object that is mapped and the value you specify for *disposition* determine whether CSRVIEW updates the mapped blocks. CSRVIEW updates the mapped blocks of a temporary object or a permanent object's scroll area if you specify a disposition of RETAIN. In all other cases, to update the mapped blocks, call the appropriate service before terminating the view:

- To update a temporary object, or to update the scroll area of a permanent object, call CSRSCOT.
- To update an object on DASD, call CSRSAVE.

Upon successful completion of the CSRVIEW operation, the content of the window depends on the value specified for disposition. The window is no longer mapped to a scroll area or to an object, however. The storage used for the window is available for other use, perhaps to use as a window for a different part of the same object or to use as a window for a different object.

Terminating Access to a Data Object

When you finish using a data object, you must terminate access to the object. When you terminate access, window services returns to the system any virtual storage it obtained for the object: storage for a temporary object or storage for a scroll area. If the object is temporary, window services deletes the object. If the object is permanent and window services dynamically allocated the data set when you requested access to the object, window services dynamically unallocates the data set. Your window is no longer mapped to the object or to a scroll area.

When you terminate access to a permanent object, window services does not update the object on DASD with changes that are in a window or the scroll area. To update the object, call CSRSAVE before terminating access to the object.

To terminate access to an object, call CSRIDAC and supply a valueof END for *operation_type*. To identify the object, supply an object identifier for *object_id*. The value you supply for *object_id* must be the same value CSRIDAC returned when you obtained access to the object.

Upon successful completion of the call, the storage used for the window is available for other use, perhaps as a window for viewing a different part of the same object or to use as a window for viewing a different object.

Handling Return Codes and Abnormal Terminations

Each time you call a service, your program receives either a return code and reason code or an abend code and a reason code. These codes indicate whether the service completed successfully, encountered an unusual condition, or was unable to complete successfully.

When you receive a return code that indicates a problem or an unusual condition, your program can either attempt to correct the problem or can terminate its execution. Return codes and reason codes are explained in Chapter 3, “Window Services” with the description of each callable service program.

When an abend occurs, the system passes control to a recovery routine, if you or your installation have provided one. A recovery routine might be able to correct the problem that caused the abend and allow your program to continue execution. If a recovery routine has been provided, it can handle the abend condition the same way it handles other abend conditions. If a recovery routine has not been provided, the system terminates execution of your program. For an explanation of the abend codes, see *z/OS MVS System Codes*.

Chapter 3. Window Services

To use window services, you issue CALLs that invoke the appropriate window services program. Each service program performs one or more functions and requires a set of parameters coded in a specific order on the CALL statement.

Depending on the function requested from a service, there might be one or more parameter values that the service ignores. Although a service might ignore a parameter value, you must still code that parameter on the CALL statement. Because the service ignores the parameter value, you can assign the parameter any value that is acceptable for the parameter's data type. If the service uses a particular parameter value, the CALL statement description in this chapter defines the allowable values that you can assign to the parameter.

This chapter describes the CALL statements that invoke window services. Each description includes a syntax diagram, parameter descriptions, and return code and reason code explanations with recommended actions. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. For examples of how to code the CALL statements, see Chapter 4, "Window Services Coding Examples" on page 4-1.

This chapter contains the following topics:

CSREVW — View an Object and Sequentially Access It

Call CSREVW if you reference data in a sequential pattern and you want to:

- Map a window to one or more blocks (4096 bytes) of a data object. If you specified scrolling when you called CSRIDAC to identify the object, CSREVW maps the window to the blocks in the scroll area and maps the scroll area to the object.
- Specify how many blocks window services is to bring into the window each time CSREVW needs more data from the object.

Mapping a data object enables your program to access the data that is viewed through the window the same way it accesses other data in your storage.

The CSREVW and CSRVIEW services differ on how to specify sequential access:

- If you use CSRVIEW and specify **sequential**, when you reference data that is not in your window, window services reads up to 16 blocks — the one that contains the data your program requests, plus the next 15 consecutive blocks. The number of consecutive blocks varies, depending on the size of the window and the availability of central storage.
- If you use CSREVW, you can specify the number of additional consecutive blocks that window services reads into the window at one time. The number ranges from 0 through 255.

Use CSREVW if your program has sequential access and can benefit from having more than 16 blocks come into a window at one time, or fewer than 16 blocks at one time.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown below. For parameters that CSREVW uses to obtain input values, assign appropriate values. For parameters that CSREVW ignores, assign any value that is valid for the particular parameter's data type.

- To map a window to a data object and begin viewing the object, specify BEGIN and SEQ and assign values, acceptable to CSREVW, to:
  - *object_id*
  - *offset*
  - *span*
  - *window_name*
  - *disposition*
  - *pfcount*

CSREVW returns values in *return_code* and in *reason_code*.

- To end the view and unmap the data object, use CSRVIEW END and specify all values, except for *pfcount*, that you specified when you mapped the window.

CALL CSREVW	(operation_type ,object_id ,offset ,span ,window_name ,usage ,disposition ,pfcount ,return_code ,reason_code)

operation_type
Specify BEGIN to request that CSREVW map a data object.

,object_id
Specifies the object identifier. Supply the object identifier that CSRIDAC returned when you obtained access to the object.

Define *object_id* as character data of length 8.

,offset
Specifies the offset of the view into the object. Specify the offset in blocks of 4096 bytes.

Define *offset* as integer data of length 4.

,span
Specifies the window size in blocks of 4096 bytes.

Define *span* as integer data of length 4.

,window_name
Specifies the symbolic name you assigned to the window in your address space.

,usage
Specify SEQ to tell CSREVW that the expected pattern of references to data in the object will be sequential.

Define this field as character data of length 6. Pad the string on the right with 1 blank.

,disposition
Defines how CSREVW is to handle data that is in the window when you begin a view. When you specify CSREVW BEGIN and a disposition of:

REPLACE The first time you reference a block to which the window is mapped, CSREVW replaces the data in the window with the data from the referenced block.

RETAIN When you reference a block to which the window is mapped, the data in the window remains unchanged. When you call CSRSAVE to save the mapped blocks, CSRSAVE saves all of the mapped blocks because CSRSAVE considers them changed.

Define *disposition* as character data of length 7. If you specify RETAIN, pad the string on the right with 1 blank.

,pfcount
Specifies the number of additional blocks you want window services to bring into the window each time your program references data that is not already in the window. The number you specify is added to the minimum of one block that window services always brings in. That is, if you specify a value of 20, window services brings in a total of 21. The number of additional blocks ranges from zero through 255.

Define *pfcount* as integer data of length 4.

,return_code
When CSREVW completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

Return codes and reason codes are explained under “Return Codes and Reason Codes”.

,reason_code
When CSREVW completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes are explained under “Return Codes and Reason Codes”.

Abend Codes

CSREVW issues abend code X'019'. For more information, see *z/OS MVS System Codes*.

Return Codes and Reason Codes

When CSREVW returns control to your program, *return_code* contains a return code and *reason_code* contains a reason code. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. Table 3-1 on page 3-4 identifies return code and reason code combinations, tells what each means, and recommends an action that you should take.

A return code of X'4' with a reason code of X'0125' or a return code of X'C' with any reason code means that data-in-virtual encountered a problem or an unexpected condition. Data-in-virtual reason codes, which are two bytes long and right justified, are explained in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. To resolve a data-in-virtual problem, request help from your system programmer.

Table 3-1. CSREVW Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	**Meaning**: The operation was successful. **Action**: Continue normal program execution.
00000004 (4)	xxxx0125 (293)	**Meaning**: The operation was successful. The service could not retain all the data that was in the scroll area, however. **Action**: Notify your system programmer.
00000012 (18)	xxxx000A (10)	**Meaning**: There is another service currently executing with the specified ID. **Action**: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	**Meaning**: An I/O error has occurred. **Action**: Notify your system programmer.
0000000C (12)	xxxx001A (26)	**Meaning**: The specified range does not encompass any mapped area of the object. **Action**: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx001C (28)	**Meaning**: The object cannot be accessed at the current time. **Action**: Try running your program at a later time. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0040 (64)	**Meaning**: The specified MAP range would cause the hiperspace data-in-virtual object to be extended such that the installation data space limits would be exceeded. **Action**: Change the MAP range you have specified or request your system programmer to increase the installation's data space limits.
0000000C (12)	xxxx0801 (2049)	**Meaning**: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. **Action**: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	**Meaning**: System error — I/O driver failure. **Action**: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	**Meaning**: System error — A necessary page table could not be read into real storage. **Action**: Notify your system programmer.
0000000C (12)	xxx00804 (2052)	**Meaning**: System error — Catalog update failed. **Action**: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	**Meaning**: System error — I/O error. **Action**: Notify your system programmer.

Table 3-1. CSREVW Return and Reason Codes (continued)

Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx0808 (2056)	**Meaning**: System error — I/O from a previous request has not completed. **Action**: Notify your system programmer.
0000002C (44)	00000004 (4)	**Meaning**: Window services have not been defined to your system or the link to the service failed. **Action**: Notify your system programmer.

CSRIDAC — Request or Terminate Access to a Data Object

Call CSRIDAC to:

- Request access to a data object
- Terminate access to a data object

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown below. For parameters that CSRIDAC uses to obtain input values, assign values that are acceptable to CSRIDAC. For parameters that CSRIDAC ignores, assign any value that is valid for the particular parameter's data type.

The parameter values that CSRIDAC uses depends on whether you are requesting access to an object or terminating access.

- To request access to a data object, specify BEGIN for *operation_type*, and assign values, acceptable to CSRIDAC, to the following parameters:
  - *object_type*
  - *object_name* if the object is permanent
  - *scroll_area*
  - *object_state* if the object is permanent and *object_type* specifies DSNAME
  - *access_mode* if the object exists and is permanent
  - *object_size* if the object is new or temporary
  - *object_size* if the object is new or temporary

  CSRIDAC ignores other parameter values. CSRIDAC returns values in *object_id*, *high_offset*, *return_code*, and *reason_code*.
- To terminate access to a data object, specify END for *operation_type*, and assign a value, acceptable to CSRIDAC, to *object_id*. CSRIDAC ignores other parameter values. CSRIDAC returns values in *return_code* and *reason_code*.

```
CALL  CSRIDAC     (operation_type
                  ,object_type
                  ,object_name
                  ,scroll_area
                  ,object_state
                  ,access_mode
                  ,object_size
                  ,object_id
                  ,high_offset
                  ,return_code
                  ,reason_code)
```

operation_type
Specifies the type of operation the service is to perform:
- To request access to an object, specify BEGIN.
- To terminate access to an object, specify END. If the object is temporary, CSRIDAC deletes it.

Define *operation_type* as character data of length 5. If you specify END, pad the string on the right with 1 or 2 blanks.

,object_type
Specifies the type of object. The types are:

DDNAME The object is an existing (OLD) VSAM linear data set allocated to the file whose DDNAME is specified by *object_name*.

DSNAME The object is the linear VSAM data set whose name is specified by *object_name*. The data set may already exist or may be a new data set that you want window services to create.

TEMPSPACE The object is a temporary data object. Window services deletes the object when your program calls CSRIDAC and *operation_type* equals END.

If *operation_type* is BEGIN, you must supply a value.

Define this parameter as character data of length 9. If you specify either DDNAME or DSNAME, pad the string on the right with 1 to 3 blanks.

,object_name
Specifies the data set name of a permanent object or the DDNAME of a data definition (DD) statement that defines a permanent object.
- If *object_type* is DDNAME, *object_name* must contain the name of a DD statement.
- If *object_type* is DSNAME, *object_name* must contain the data set name of the permanent object.

If *operation_type* is BEGIN and *object_type* is DDNAME or DSNAME, you must supply a value for *object_name*.

Define *object_name* as character data of length 1 to 45. If *object_name* contains fewer than 45 characters, pad the name on the right with a blank.

,scroll_area
Specifies whether window services is to create a scroll area for the data object.
YES Create a scroll area.
NO Do not create a scroll area.

If *operation_type* is BEGIN and *object_type* is TEMPSPACE, specify YES.

Define *scroll_area* as character data of length 3. If you specify NO, pad the string on the right with a blank.

,object_state
Specifies the state of the object.
OLD The object exists.
NEW The object does not exist and window services must create it.

If *operation_type* is BEGIN and *object_type* is DSNAME, you must supply a value for *object_state*.

Define *object_state* as character data of length 3.

,access_mode
Specifies the type of access required.
READ READ access.
UPDATE UPDATE access.

If *operation_type* is BEGIN and *object_type* is DDNAME or DSNAME, you must supply a value for *access_mode*. For a new or temporary data object, window services assumes UPDATE.

Define *access_mode* as character data of length 6. If you specify READ, pad the string on the right with 1 or 2 blanks.

,object_size
Specifies the maximum size of the new object in units of 4096 bytes.

This parameter is required if either of the following conditions is true:

- *Operation_type* is BEGIN, *object_type* is DSNAME, and *object_state* is NEW
- *Operation_type* is BEGIN and *object_type* is TEMPSPACE

Define *object_size* as integer data of length 4.

,object_id
Specifies the object identifier.

When *operation_type* is BEGIN, the service returns the object identifier in this parameter. Use the identifier to identify the object to other window services.

When *operation_type* is END, you must supply the object identifier in this parameter.

Define *object_id* as character data of length 8.

,high_offset
When CSRIDAC completes, *high_offset* contains the size of the existing object expressed in blocks of 4096 bytes

Define *high_offset* as integer data of length 4.

,return_code
When CSRIDAC completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

Return codes and reason codes are explained under “Return Codes and Reason Codes”.

,reason_code
When CSRIDAC completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes are explained under “Return Codes and Reason Codes”.

Abend Codes

CSRIDAC issues abend code X'019'. For more information, see *z/OS MVS System Codes*.

Return Codes and Reason Codes

When CSRIDAC returns control to your program, *return_code* contains a return code and *reason_code* contains a reason code. Return codes and reason codes

are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. Table 3-2 identifies return code and reason code combinations, tells what each means, and recommends an action that you should take.

A return code of X'C' means that data-in-virtual encountered a problem or an unexpected condition. The associated reason codes are data-in-virtual reason codes. Data-in-virtual reason codes are two bytes long and right justified. To resolve a data-in-virtual problem, request help from your system programmer. For information about data-in-virtual, see the *z/OS MVS Programming: Assembler Services Guide*.

Table 3-2. CSRIDAC Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	**Meaning**: The operation was successful. **Action**: Continue normal program execution.
00000008 (8)	00000118 (280)	**Meaning**: The system could not obtain enough storage to create a hiperspace for the temporary object or the scroll area. **Note:** Hiperspace™ is the name the system uses to identify the storage it uses to create a temporary object or a scroll area for a permanent object. **Action**: Notify your system programmer. The system programmer might have to increase the SMF limit for data spaces and hiperspace that are intended for the user.
00000008 (8)	00000119 (281)	**Meaning**: The system could not delete or unidentify the temporary object or the scroll area. **Action**: Notify your system programmer.
00000008 (8)	0000011A (282)	**Meaning**: The system was unable to create a new VSAM linear data set. DFP 3.1 must be running and SMS must be active. **Action**: Notify your system programmer.
0000000C (12)	xxxx000A (10)	**Meaning**: Another service currently is executing with the specified ID. **Action**: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx001C (28)	**Meaning**: The object cannot be accessed at the current time. **Action**: Try running your program at a later time. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0037 (55)	**Meaning**: The caller invoked ACCESS. The access is successful, but the system is issuing a warning that the data set was not allocated with a SHAREOPTIONS(1,3). **Action**: Notify your system programmer.
0000000C (12)	xxxx003E (62)	**Meaning**: The hiperspace data-in-virtual object may not be accessed at this time. (If MODE=READ, the object is already accessed under a different ID for UPDATE. If MODE=UPDATE, the object is already accessed under at least one other ID.) **Action**: Try running your program at a later time. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0801 (2049)	**Meaning**: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. **Action**: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	**Meaning**: System error — I/O driver failure. **Action**: Notify your system programmer.

Table 3-2. CSRIDAC Return and Reason Codes (continued)

Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx0805 (2053)	**Meaning**: System error — A system error of indeterminate origin has occurred. **Action**: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	**Meaning**: System error — I/O from a previous request has not completed. **Action**: Notify your system programmer.
00000010 (16)	rrrrnnnn	**Meaning**: The system was unable to allocate or unallocate the data set specified as *object_name*. The value *rrrr* is the return code from dynamic allocation. The value *nnnn* is the two-byte reason code from dynamic allocation. See *z/OS MVS Programming: Authorized Assembler Services Guide* for dynamic allocation return and reason codes. **Action**: If *object_state* is NEW, make sure that a data set of the same name does not already exist. If one does already exist, either use the existing data set or change the name of your data set. If you are unable to correct the problem, notify your system programmer.
0000002C (44)	00000004 (4)	**Meaning**: Window services have not been defined to your system or the link to the service failed. **Action**: Notify your system programmer.

CSRREFR — Refresh an Object

To refresh changed data that is in a window, a scroll area, or a temporary object, call CSRREFR. CSRREFR refreshes changed data within specified blocks as follows:

- If the object is permanent, CSRREFR replaces specified changed blocks in windows or the scroll area with corresponding blocks from the object on DASD.
- For a temporary object, CSRREFR refreshes specified changed blocks in windows and the object by setting the blocks to binary zeroes.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown below. For parameters that CSRREFR uses to obtain input values, assign values that are acceptable to CSRREFR. For parameters that CSRREFR ignores, assign any value that is valid for the particular parameter's data type.

Assign values, acceptable to CSRREFR, to *object_id*, *offset*, and *span*. CSRREFR ignores other parameter values. CSRREFR returns values in *return_code* and *reason_code*.

CALL CSRREFR	(object_id ,offset ,span ,return_code ,reason_code)

object_id
Specifies the object identifier. Supply the same object identifier that CSRIDAC returned when you obtained access to the object.

Define *object_id* as character data of length 8.

,offset
Specifies the offset into the object in blocks of 4096 bytes. A value of 0 specifies the first block of 4096 bytes or bytes 0 to 4095 of the object; a value of 1 specifies the second block of 4096 bytes, or bytes 4096 to 8191 of the object, and so forth.

Define *offset* as integer data of length 4.

offset and *span*, together, determine which part of the object window services refreshes. To refresh the entire object, specify 0 for *offset* and 0 for *span*.

,span
Specifies how many 4096-byte blocks CSRREFR is to refresh.

Define *span* as integer data of length 4.

,return_code
When CSRREFR completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

Return codes and reason codes are explained under “Return Codes and Reason Codes”.

,reason_code
When CSRREFR completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes are explained under “Return Codes and Reason Codes”.

Abend Codes

CSRREFR issues abend code X'019'. For more information, see *z/OS MVS System Codes*.

Return Codes and Reason Codes

When CSRREFR returns control to your program, *return_code* contains a return code and *reason_code* contains a reason code. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. Table 3-3 identifies return code and reason code combinations, tells what each means, and recommends an action that you should take.

A return code of X'C' means that data-in-virtual encountered a problem or an unexpected condition. The associated reason codes are data-in-virtual reason codes. Data-in-virtual reason codes are two bytes long and right justified. To resolve a data-in-virtual problem, request help from your system programmer.

Table 3-3. CSRREFR Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	**Meaning**: The operation was successful. **Action**: Continue normal program execution.
00000008 (8)	00000152 (338)	**Meaning**: The system could not refresh all of the temporary object within the specified span. **Action**: Notify your system programmer.

Table 3-3. CSRREFR Return and Reason Codes (continued)

Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx000A (10)	**Meaning**: There is another service currently executing with the specified ID. **Action**: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	**Meaning**: An I/O error has occurred. **Action**: Notify your system programmer.
0000000C (12)	xxxx001A (26)	**Meaning**: The specified range does not include any mapped block of the object. **Action**: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx0801 (2049)	**Meaning**: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. **Action**: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	**Meaning**: System error — A necessary page table could not be read into real storage. **Action**: Notify your system programmer.
0000000C (12)	xxxx0805 (2053)	**Meaning**: System error — A system error of indeterminate origin has occurred. **Action**: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	**Meaning**: System error — I/O error. **Action**: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	**Meaning**: System error — I/O from a previous request has not completed. **Action**: Notify your system programmer.
0000002C (44)	00000004 (4)	**Meaning**: Window services have not been defined to your system or the link to the service failed. **Action**: Notify your system programmer.

CSRSAVE — Save Changes Made to a Permanent Object

To update specified blocks of a permanent object with changes, call CSRSAVE. The changes can be in blocks that are mapped to the scroll area, in blocks that are mapped to windows, or in a combination of these places.

Usage Note

You cannot use CSRSAVE to save changes made to a temporary object. If you call CSRSAVE for a temporary object, CSRSAVE ignores the request and returns control to your program with a return code of 8. To save changes made to a temporary object, call CSRSCOT.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown below. For parameters that CSRSAVE

uses to obtain input values, assign values that are acceptable to CSRSAVE. For parameters that CSRSAVE ignores, assign any value that is valid for the particular parameter's data type.

Assign values, acceptable to CSRSAVE, to *object_id*, *offset*, and *span*. CSRSAVE ignores other parameter values. CSRSAVE returns values in *new_hi_offset*, *return_code*, and *reason_code*.

CALL CSRSAVE	(object_id ,offset ,span ,new_hi_offset ,return_code ,reason_code)

object_id
Specifies the object identifier. Supply the same object identifier that CSRIDAC returned when you obtained access to the object.

Define *object_id* as character data of length 8.

,offset
Specifies the offset into the object in blocks of 4096 bytes. A value of 0 specifies the first block of 4096 bytes or bytes 0 to 4095 of the object; a value of 1 specifies the second block of 4096 bytes, or bytes 4096 to 8191 of the object, and so forth.

Define *offset* as integer data of length 4.

offset and *span*, together, determine which part of the object window services saves. To save the entire object, specify 0 for *offset* and 0 for *span*.

,span
Specifies how many 4096-byte blocks CSRSAVE is to save.

Define *span* as integer data of length 4.

,new_hi_offset
When CSRSAVE completes, *new_hi_offset* contains the new size of the object expressed in units of 4096 bytes.

Define *new_hi_offset* as integer data of length 4.

,return_code
When CSRSAVE completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

Return codes and reason codes are explained under "Return Codes and Reason Codes" on page 3-13.

,reason_code
When CSRSAVE completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes are explained under "Return Codes and Reason Codes" on page 3-13.

Abend Codes

CSRSAVE issues abend code X'019'. For more information, see *z/OS MVS System Codes*.

Return Codes and Reason Codes

When CSRSAVE returns control to your program, *return_code* contains a return code and *reason_code* contains a reason code. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. Table 3-4 identifies return code and reason code combinations, tells what each means, and recommends an action that you should take.

A return code of X'4' with a reason code of X'0807' or a return code of X'C' with any reason code means that data-in-virtual encountered a problem or an unexpected condition. Data-in-virtual reason codes are two bytes long and right justified. To resolve a data-in-virtual problem, request help from your system programmer. For information about data-in-virtual, see the *z/OS MVS Programming: Assembler Services Guide*.

Table 3-4. CSRSAVE Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	**Meaning**: The operation was successful. **Action**: Continue normal program execution.
00000004 (4)	xxxx0807 (2055)	**Meaning**: Media damage may be present in allocated DASD space. The damage is beyond the currently saved portion of the object. The SAVE operation completed successfully. **Action**: Notify your system programmer.
00000008 (8)	xxxx0143 (323)	**Meaning**: You cannot use the SAVE service for a temporary object. **Action**: Use the scrollout (CSRSCOT) service.
0000000C (12)	xxxx000A (10)	**Meaning**: There is another service currently executing with the specified ID. **Action**: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	**Meaning**: An I/O error has occurred. **Action**: Notify your system programmer.
0000000C (12)	xxxx001A (26)	**Meaning**: The specified range does not encompass any mapped area of the object. **Action**: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx0801 (2049)	**Meaning**: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. **Action**: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	**Meaning**: System error — I/O driver failure. **Action**: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	**Meaning**: System error — A necessary page table could not be read into real storage. **Action**: Notify your system programmer.
0000000C (12)	xxxx0804 (2052)	**Meaning**: System error — Catalog update failed. **Action**: Notify your system programmer.

Table 3-4. CSRSAVE Return and Reason Codes (continued)

Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx0806 (2054)	**Meaning**: System error — I/O error. **Action**: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	**Meaning**: System error — I/O from a previous request has not completed. **Action**: Notify your system programmer.
0000002C (44)	00000004 (4)	**Meaning**: Window services have not been defined to your system or the link to the service failed. **Action**: Notify your system programmer.

CSRSCOT — Save Object Changes in a Scroll Area

Call CSRSCOT to:

- Update specified blocks of a permanent object's scroll area with changes that appear in a window you have defined for the object. CSRSCOT requires that the permanent object have a scroll area. CSRSCOT changes only the content of the scroll area and not the content of the permanent data object.
- Update specified blocks of a temporary data object with the changes that appear in a window you have defined for the data object.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown below. For parameters that CSRSCOT uses to obtain input values, assign values that are acceptable to CSRSCOT. For parameters that CSRSCOT ignores, assign any value that is valid for the particular parameter's data type.

Assign values, acceptable to CSRSCOT, to *object_id*, *offset*, and *span*. CSRSCOT ignores other parameter values. CSRSCOT returns values in *return_code* and *reason_code*.

```
CALL  CSRSCOT     (object_id
                  ,offset
                  ,span
                  ,return_code
                  ,reason_code)
```

object_id
Specifies the object identifier. Supply the same object identifier that CSRIDAC returned when you obtained access to the object.

Define *object_id* as character data of length 8.

,offset
Specifies the offset into the object in blocks of 4096 bytes. A value of 0 specifies the first block of 4096 bytes or bytes 0 to 4095 of the object; a value of 1 specifies the second block of 4096 bytes, or bytes 4096 to 8191 of the object, and so forth.

Define *offset* as integer data of length 4.

offset and *span*, together, determine which part of the object CSRSCOT updates. To update the entire object, specify 0 for *offset* and 0 for *span*.

,span
Specifies how many 4096-byte blocks CSRSCOT is to update.

Define *span* as integer data of length 4.

,return_code
When CSRSCOT completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

Return codes and reason codes are explained under “Return Codes and Reason Codes”.

,reason_code
When CSRSCOT completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes are explained under “Return Codes and Reason Codes”.

Abend Codes

CSRSCOT issues abend code X'019'. For more information, see *z/OS MVS System Codes*.

Return Codes and Reason Codes

When CSRSCOT returns control to your program, *return_code* contains a return code and *reason_code* contains a reason code. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. Table 3-5 identifies return code and reason code combinations, tells what each means, and recommends an action that you should take.

A return code of X'C' means that data-in-virtual encountered a problem or an unexpected condition. The associated reason codes are data-in-virtual reason codes. Data-in-virtual reason codes are two bytes long and right justified. For information about data-in-virtual, see *z/OS MVS Programming: Assembler Services Guide*. To resolve the problem, request help from your system programmer.

Table 3-5. CSRSCOT Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	**Meaning**: The operation was successful. **Action**: Continue normal program execution.
00000004 (4)	xxxx0807 (2055)	**Meaning**: Media damage may be present in allocated DASD space. The damage is beyond the currently saved portion of the object. The SAVE operation completed successfully. **Action**: Notify your system programmer.
0000000C (12)	xxxx000A (10)	**Meaning**: There is another service currently executing with the specified ID. **Action**: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	**Meaning**: An I/O error has occurred. **Action**: Notify your system programmer.

Table 3-5. CSRSCOT Return and Reason Codes (continued)

Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx001A (26)	**Meaning**: The specified range does not encompass any mapped area of the object. **Action**: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx0801 (2049)	**Meaning**: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. **Action**: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	**Meaning**: System error — I/O driver failure. **Action**: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	**Meaning**: System error — A necessary page table could not be read into real storage. **Action**: Notify your system programmer.
0000000C (12)	xxxx0804 (2052)	**Meaning**: System error — Catalog update failed. **Action**: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	**Meaning**: System error — I/O error. **Action**: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	**Meaning**: System error — I/O from a previous request has not completed. **Action**: Notify your system programmer.
0000002C (44)	00000004 (4)	**Meaning**: Window services have not been defined to your system or the link to the service failed. **Action**: Notify your system programmer.

CSRVIEW — View an Object

Call CSRVIEW to:

- Map a window to one or more blocks of a data object. If you specified scrolling when you called CSRIDAC to identify the object, CSRVIEW maps the window to the scroll area and the scroll area to the object.
- Specify that the reference pattern you are using is either random or sequential.
- End a view that you previously created through CSRVIEW or CSREVW and unmap the object.

Mapping a data object enables your program to access the data that is viewed through the window the same way it accesses other data in your storage.

The CSREVW service also maps a data object. Use that service if your program can benefit from having more than 16 blocks come into a window at one time or if it can benefit from having fewer than 16.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown below. For parameters that CSRVIEW uses to obtain input values, assign values that are acceptable to CSRVIEW. For parameters that CSRVIEW ignores, assign any value that is valid for the particular parameter's data type.

The type of function you request determines which parameter values CSRVIEW uses to obtain input values:

- To map a window to a data object and begin viewing the object, specify BEGIN for *operation_type*, and assign values, acceptable to CSRVIEW, to:
  - *object_id*
  - *offset*
  - *span*
  - *window_name*
  - *usage*
  - *disposition*

  CSRVIEW ignores other parameter values. CSRVIEW returns values in *return_code* and in *reason_code*.
- To end a view set by either CSRVIEW or CSREVW and to unmap the data object, specify END for *operation_type*, and assign values, acceptable to CSRVIEW, to:
  - *object_id*
  - *offset*
  - *span*
  - *window_name*
  - *usage*
  - *disposition*

  CSRVIEW ignores other parameter values. CSRVIEW returns values in *return_code* and *reason_code*.

CALL CSRVIEW	(operation_type ,object_id ,offset ,span ,window_name ,usage ,disposition ,return_code ,reason_code)

operation_type
Specifies the type of operation CSRVIEW is to perform. To begin viewing an object, specify BEGIN. To end a view, specify END.

Define *operation_type* as character data of length 5. If you specify END, pad the string on the right with 1 or 2 blanks.

,object_id
Specifies the object identifier. Supply the object identifier that CSRIDAC returned when you obtained access to the object.

Define *object_id* as character data of length 8.

,offset
Specifies the offset of the view into the object. Specify the offset in blocks of 4096 bytes.

Define *offset* as integer data of length 4.

,span
Specifies the window size in blocks of 4096 bytes.

Define *span* as integer data of length 4.

,window_name
Specifies the symbolic name you assigned to the window in your address space.

,usage
Specifies the expected pattern of references to pages in the object. Specify one of the following values:

SEQ The reference pattern is expected to be sequential. If you specify SEQ, window services brings up to 16 blocks of data into the window at a time, depending on the size of the window.

RANDOM The reference pattern is expected to be random. If you specify RANDOM, window services brings data into the window one block at a time.

Define *usage* as character data of length 6. If you specify SEQ, pad the string on the right with 1 to 3 blanks.

,disposition
Defines how CSRVIEW is to handle data that is in the window when you begin or end a view.

- When you specify CSRVIEW with an *operation_type* of BEGIN and a disposition of:

  REPLACE The first time you reference a block to which the window is mapped, CSRVIEW replaces the data in the window with the data from the referenced block.

  RETAIN When you reference a block to which the window is mapped, the data in the window remains unchanged. When you call CSRSAVE to save the mapped blocks, CSRSAVE saves all of the mapped blocks because CSRSAVE considers them changed.

- When you specify CSRVIEW with an *operation_type* of END and a disposition of:

  REPLACE CSRVIEW discards the data that is in the window making the window contents unpredictable. CSRVIEW does not update mapped blocks of the object or scroll area.

  RETAIN If the object is permanent and has no scroll area, CSRVIEW retains the data that is in the window. CSRVIEW does not update mapped blocks of the object. If the object is permanent and has a scroll area, or if the object is temporary, CSRVIEW retains the data that is in the window and updates the mapped blocks of the object or scroll area.

Define *disposition* as character data of length 7. If you specify RETAIN, pad the string on the right with a blank.

,return_code
When CSRVIEW completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

Return codes and reason codes are explained under “Return Codes and Reason Codes”.

,reason_code
When CSRVIEW completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes are explained under “Return Codes and Reason Codes”.

Abend Codes

CSRVIEW issues abend code X'019'. For more information, see *z/OS MVS System Codes*.

Return Codes and Reason Codes

When CSRVIEW returns control to your program, *return_code* contains a return code and *reason_code* contains a reason code. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. Table 3-6 identifies return code and reason code combinations, tells what each means, and recommends an action that you should take.

A return code of X'4' with a reason code of X'0125' or a return code of X'C' with any reason code means that data-in-virtual encountered a problem or an unexpected condition. Data-in-virtual reason codes are two bytes long and right justified. For information about data-in-virtual, see *z/OS MVS Programming: Assembler Services Guide*. To resolve the problem, request help from your system programmer.

Table 3-6. CSRVIEW Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	**Meaning**: The operation was successful. **Action**: Continue normal program execution.
00000004 (4)	xxxx0125 (293)	**Meaning**: The operation was successful. The service could not retain all the data that was in the scroll area, however. **Action**: Notify your system programmer.
0000000C (12)	xxxx000A (10)	**Meaning**: There is another service currently executing with the specified ID. **Action**: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	**Meaning**: An I/O error has occurred. **Action**: Notify your system programmer.
0000000C (12)	xxxx001A (26)	**Meaning**: The specified range does not encompass any mapped area of the object. **Action**: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx001C (28)	**Meaning**: The object cannot be accessed at the current time. **Action**: Try running your program at a later time. If the problem persists, notify your system programmer.

Table 3-6. CSRVIEW Return and Reason Codes (continued)

Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx0040 (64)	**Meaning**: The specified MAP range would cause the hiperspace data-in-virtual object to be extended such that the installation data space limits would be exceeded. **Action**: Change the MAP range you have specified or request your system programmer to increase the installation's data space limits.
0000000C (12)	xxxx0801 (2049)	**Meaning**: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. **Action**: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	**Meaning**: System error — I/O driver failure. **Action**: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	**Meaning**: System error — A necessary page table could not be read into real storage. **Action**: Notify your system programmer.
0000000C (12)	xxx00804 (2052)	**Meaning**: System error — Catalog update failed. **Action**: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	**Meaning**: System error — I/O error. **Action**: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	**Meaning**: System error — I/O from a previous request has not completed. **Action**: Notify your system programmer.
0000002C (44)	00000004 (4)	**Meaning**: Window services have not been defined to your system or the link to the service failed. **Action**: Notify your system programmer.

Chapter 4. Window Services Coding Examples

The following examples show how to invoke window services from each of the supported languages. Following each program example is an example of the JCL needed to compile, link edit, and execute the program example. Use these examples to supplement and reinforce information that is presented elsewhere in this book.

Note: Included in the FORTRAN example is the code for a required assembler language program. This program ensures that the window for the FORTRAN program is aligned on a 4K boundary.

The examples are presented on the following pages:

ADA Example

```
-- ------------------------------------------------------------------ --
--   This program illustrates how Data Window services are invoked   --
--   using ADA. Note that the data object referenced in this program --
--   is permanent and already allocated, and is defined by the DD    --
--   statement CSRDD1 in the JCL.                                     --
--                                                                    --
--   This program must be linkedited with the CSR linkage-assist     --
--   routines (also known as stubs) in SYS1.CSSLIB.                   --
-- ------------------------------------------------------------------ --

with EBCDIC; use EBCDIC;
with System;
with Text_Io;
with Unchecked_Conversion;
with Td_Standard; use Td_Standard;

procedure CRTPAN06 is

  subtype Str3  is EString (1..3);
  subtype Str5  is EString (1..5);
  subtype Str6  is EString (1..6);
  subtype Str7  is EString (1..7);
  subtype Str8  is EString (1..8);
  subtype Str9  is EString (1..9);

  function Integer_Address is new Unchecked_Conversion
           (System.Address, Integer);

  function Int_To_32 is new Unchecked_Conversion
           (Integer, Integer_32);
  Orig,                         -- Index to indicate the 'start'
                                -- of an array
  Ad, I        : Integer;       -- Temporary variables
  Voffset,                      -- Offset passed as parameter
  Vofset2,                      -- Offset passed as parameter
  Vobjsiz,                      -- Object size, as parameter
  Vwinsiz,                      -- Window size, as parameter
  High_Offset,                  -- Size of object in pages
  New_Hi_Offset,                -- New max size of the object
```

ADA Example

```
Return_Code,                -- Return code
Reason_Code  : Integer_32;  -- Reason code
Object_Id    : Str8;        -- Identifying token
Cscroll      : Str3;        -- Scroll area  YES/NO
Cobstate     : Str3;        -- Object state NEW/OLD
Coptype      : Str5;        -- Operation type BEGIN/END
Caccess      : Str6;        -- Access RANDOM/SEQ
Cusage       : Str6;        -- Usage READ/UPDATE
Cdisp        : Str7;        -- Disposition RETAIN/REPLACE
Csptype      : Str9;        -- Object type DSNAME/DDNAME/TEMPSPACE
Cobname      : Str7;        -- Object name
K            : constant Integer    := 1024;  -- One kilo-byte
Pagesize     : constant Integer    := 4 * K; -- Page (4K) boundary
Offset       : constant Integer_32 := 0; -- Start of permanent object
Window_Size  : constant Integer    := 40;    -- Window size in pages
Num_Win_Elem : constant Integer    := Window_Size*K; -- Num of 4-byte
                                                -- elements in window
Object_Size  : constant Integer    := 3*Window_Size; -- Chosen object
                                                     -- size in pages
Num_Sp_Elem  : constant Integer    := (Window_Size+1)*K; -- Num of
                                           -- 4-byte elements in space

type S is array (positive range <>) of Integer; -- Define byte
                                                -- aligned space
Sp : S (1..Num_Sp_Elem); -- Space allocated for window

procedure CSRIDAC (Op_Type     : in Str5;
                   Object_Type : in Str9;
                   Object_Name : in Str7;
                   Scroll_Area : in Str3;
                   Object_State: in Str3;
                   Access_Mode : in Str6;
                   Vobjsiz     : in Integer_32;
                   Object_Id   : out Str8;
                   High_Offset : out Integer_32;
                   Return_Code : out Integer_32;
                   Reason_Code : out Integer_32);
pragma Interface (Assembler, CSRIDAC);

procedure CSRVIEW (Op_Type     : in Str5;
                   Object_Id   : in Str8;
                   Offset      : in Integer_32;
                   Window_Size : in Integer_32;
                   Window_Name : in S;
                   Usage       : in Str6;
                   Disposition : in Str7;
                   Return_Code : out Integer_32;
                   Reason_Code : out Integer_32);
pragma Interface (Assembler, CSRVIEW);

procedure CSRSCOT (Object_Id   : in Str8;
                   Offset      : in Integer_32;
                   Span        : in Integer_32;
                   Return_Code : out Integer_32;
                   Reason_Code : out Integer_32);
pragma Interface (Assembler, CSRSCOT);

procedure CSRSAVE (Object_Id   : in Str8;
                   Offset      : in Integer_32;
                   Span        : in Integer_32;
                   New_Hi_Offset : out Integer_32;
                   Return_Code : out Integer_32;
                   Reason_Code : out Integer_32);
pragma Interface (Assembler, CSRSAVE);

procedure CSRREFR (Object_Id   : in Str8;
                   Offset      : in Integer_32;
```

```
                    Span        : in Integer_32;
                    Return_Code : out Integer_32;
                    Reason_Code : out Integer_32);
 pragma Interface (Assembler, CSRREFR);

begin
  Text_Io.Put_Line ("<<Begin Window Services Interface Validation>>");
  Text_Io.New_Line;

  Vobjsiz := Int_To_32(Object_Size); -- Set object size in variable
  Voffset := Offset;                 -- Set offset to 0 for 1st map
  Vwinsiz := Int_To_32(Window_Size); -- Set window size in variable
  Vofset2 := Offset+Vwinsiz;         -- Set offset to 40 for 2nd map

  Coptype  := "BEGIN";
  Csptype  := "DDNAME   ";
  Cobname  := "CSRDD1 ";
  Cscroll  := "YES";
  Cobstate := "OLD";
  Caccess  := "UPDATE";

  CSRIDAC (Coptype,                      -- Set up access to the
           Csptype,                      -- permanent object and
           Cobname,                      -- request a scroll area
           Cscroll,
           Cobstate,
           Caccess,
           Vobjsiz,
           Object_Id,
           High_Offset,
           Return_Code,
           Reason_Code);

-- When you want to map a window to your object, data window services
-- expects the address of the start of the window to be on a page (4K)
-- boundary, and the length of the window to be a multiple of 4096 bytes.
-- If your window is an array, the address of the first element
-- of the array must be on a page boundary. If this is not the case,
-- you can appropriately choose one slice of your array that starts
-- on a 4K boundary and is a multiple of 4096 bytes in length to map
-- onto your object.
-- To illustrate, consider the array A(1..max_len). If the address of
-- A(1) is not on page boundary, you cannot map A(1..max_len) to your
-- object. You can, however, map A(n..m) to your object if you choose
-- some appropriate values n and m such that A(n) starts on a 4K
-- boundary and A(n..m) is a multiple of 4096 bytes in length.

  Ad := Integer_Address(Sp(1)'Address); -- Get address of start of array

-- Determine the first element whose address is on page boundary
-- and use that element as the origin of the array.

  Orig := (Ad mod Pagesize);             -- See where the start of
                                         -- array is in page

  if Orig = 0 then                       -- If already on page boundary
     Orig := 1;                          -- Keep the old origin
  else
     Orig := (Pagesize - Orig) / 4 + 1;  -- Need new origin
  end if;

  Coptype := "BEGIN";
  Cusage  := "RANDOM";
  Cdisp   := "REPLACE";

-- You can pass an array slice as a parameter to a non-Ada subprogram,
-- and because the slice is a composite object, the parameter list
```

ADA Example

```
-- contains the actual address of the first element in the slice.
-- To elaborate further:
-- Scalar data is passed by copy, but composite data is passed by
-- reference. If the scalar value was passed as a scalar, the assemble\
-- program would receive the address of the copy and not the address of
-- the scalar.  By passing the scalar value as an array slice, a
-- composite data type is being passed and thus is passed by reference.
-- Using this technique, the assembler code receives the actual address
-- of the scalar, not a copy of the scalar.


  CSRVIEW (Coptype,                -- Now map a window (the array)
           Object_Id,              -- to the permanent object.
           Voffset,                -- (Actually, CSRVIEW will map the
           Vwinsiz,                -- window to the blocks in the
           Sp(Orig..Num_Sp_Elem),  -- scroll area and map the scroll
           Cusage,                 -- area to the object.)
           Cdisp,
           Return_Code,
           Reason_Code);

  for I in 0 .. Num_Win_Elem-1 loop    -- Put data in window area
      Sp(I+Orig) := I+1;
  end loop;

  CSRSCOT (Object_Id,                   -- Capture the view in window.
           Voffset,                     -- Note: only the scroll area
           Vwinsiz,                     -- is updated, the permanent
           Return_Code,                 -- object remains unchanged.
           Reason_Code);

  Coptype := "END  ";
  Cusage  := "RANDOM";
  Cdisp   := "RETAIN ";

  CSRVIEW (Coptype,                     -- End the view in window
           Object_Id,
           Voffset,
           Vwinsiz,
           Sp(Orig..Num_Sp_Elem),
           Cusage,
           Cdisp,
           Return_Code,
           Reason_Code);

  Coptype := "BEGIN";
  Cusage  := "RANDOM";
  Cdisp   := "REPLACE";

  CSRVIEW (Coptype,                     -- Now map the same window
           Object_Id,                   -- to different part of the
           Vofset2,                     -- permanent object.
           Vwinsiz,
           Sp(Orig..Num_Sp_Elem),
           Cusage,
           Cdisp,
           Return_Code,
           Reason_Code);

  for I in 0 .. Num_Win_Elem-1 loop    -- Put data in window area
      Sp(I+Orig) := I+1;
  end loop;

  CSRSAVE (Object_Id,                -- Capture the view in window.
           Vofset2,                  -- Note: this time the permanent
           Vwinsiz,                  -- object is updated with the
           New_Hi_Offset,            -- changes.
```

```
         Return_Code,
         Reason_Code);

Coptype := "END  ";
CUsage  := "RANDOM";
Cdisp   := "RETAIN ";

CSRVIEW (Coptype,                  -- End the current view in
         Object_Id,                -- the window
         Vofset2,
         Vwinsiz,
         Sp(Orig..Num_Sp_Elem),
         Cusage,
         Cdisp,
         Return_Code,
         Reason_Code);

Coptype := "BEGIN";
Cusage  := "RANDOM";
Cdisp   := "REPLACE";

CSRVIEW (Coptype,                  -- Now go back to reestablish
         Object_Id,                -- the 1st map using the same
         Voffset,                  -- window area
         Vwinsiz,
         Sp(Orig..Num_Sp_Elem),
         Cusage,
         Cdisp,
         Return_Code,
         Reason_Code);

CSRREFR (Object_Id,                -- Refresh the data in the window
         Voffset,
         Vwinsiz,
         Return_Code,
         Reason_Code);

Coptype := "END  ";
Cusage  := "RANDOM";
Cdisp   := "RETAIN ";

CSRVIEW (Coptype,                  -- End the view in window
         Object_Id,
         Voffset,
         Vwinsiz,
         Sp(Orig..Num_Sp_Elem),
         Cusage,
         Cdisp,
         Return_Code,
         Reason_Code);

Coptype  := "END  ";
Csptype  := "DDNAME   ";
Cobname  := "CSRDD1 ";
Cscroll  := "YES";
Cobstate := "OLD";
Caccess  := "UPDATE";

CSRIDAC (Coptype,                  -- Terminate access to the
         Csptype,                  -- permanent object
         Cobname,
         Cscroll,
         Cobstate,
         Caccess,
         Vwinsiz,
         Object_Id,
         High_Offset,
```

ADA Example

```
           Return_Code,
           Reason_Code);

end CRTPAN06;
```

```
//ADAJOB    JOB                                                          00000100
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *   00000500
//* JCL USED TO COMPILE, LINK, AND EXECUTE THE ADA PROGRAM CRTPAN06     00000600
//* THAT USES DATA WINDOW SERVICES                                      00000700
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *   00000800
/*JOBPARM  T=2,L=99                                                     00050000
//ADACOBI  EXEC PGM=IKJEFT01,DYNAMNBR=133                              00055813
//SYSTSPRT DD SYSOUT=*                                                 00055913
//SYSTSIN  DD *                                                        00056008
  ALLOC FI(SYSLIB) DS('SYS1.CSSLIB') SHR                               00056147
  EX 'HLQ.SEVGEXE1(ADA)' 'USERID.DWS.ADA'' (MAI CRE'                  00056251
/*                                                                     00057008
//ADARUN   EXEC PGM=CRTPAN06,DYNAMNBR=133                             00070036
//STEPLIB  DD DISP=SHR,DSN=HLQ.SEVHMOD1                               00100051
//         DD DISP=SHR,DSN=USERID.LOAD                                 00110051
//CSRDD1   DD DSN=USERID.ADA.DWSTEST.DATA,DISP=SHR                     00120051
//CONOUT   DD SYSOUT=*,                                               00130013
//            DCB=(LRECL=133,RECFM=F)                                  00140027
```

C/370™ Example

The following example, coded in C/370, creates and uses a temporary data object.

```
#include <stdio.h>
#include <stdlib.h>
/* Defined macros that will be used in the program.            */
#define SIZE  8*1024
#define OBJ_SIZE  8
#define PAGE_SIZE  (4*1024)
#define DWS_FILE  "DWS.FILE1 "
#define TRUE  1
#define FALSE 0
char windows[SIZE];
char *view;
void init_mem(char init_value, char *low_mem, int size);
int chk_code(long int ret, long int reason, int linenumber);
main()
{
  /* Initialized variables that will be used in the Callable    */
  /* Services.                                                  */
  char op_type1[5] = "BEGIN";
  char op_type2[5] = "END ";
  char object_type[9] = "TEMPSPACE";
  char object_name[45] = DWS_FILE;
  char scroll_area[3] = "YES";
  char object_state[3] = "NEW";
  char access_mode[6] = "UPDATE";
  long int object_size = OBJ_SIZE;
  char disposition[7] = "REPLACE";
  char usage[6] = "SEQ ";
  char object_id[8];
  long int high_offset, return_code, reason_code;
  long int offset, window_size, window_addr;
  long int span, new_hi_offset;
  long int addr;
  int i, ret, origin, errflag = FALSE;
  double id;
  /* Set up access to a Hiperspace object using TEMPSPACE.      */
  /* Check for return code and reason code after the call.      */
  csridac(op_type1, object_type, object_name, scroll_area, object_state,
          access_mode,&object_size,&object_id,&high_offset,&return_code,;
          &reason_code);
```

```
chk_code(return_code,reason_code,__LINE__);
/* Define a window in a 4K region and initialize              */
/* variables for CSRVIEW. Define the window for  the          */
/* TEMPSPACE and verify the return code and reason code.      */
init_mem('0',windows,SIZE);
addr = (int) windows % 4096;
if (addr != 0) view = windows + 4096 - addr;
offset = 0; window_size = 1;
csrview(op_type1,&object_id,&offset,&window_size,view,;
        usage, disposition, &return_code, &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Change values in the window into 1.                        */
init_mem('1',view,4096);
/* Capture the view in the 1st window.                        */
offset = 0; window_size = 1;
csrscot(&object_id, &offset, &window_size,&return_code,;
        &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Make sure that CSRSAVE will not save changes for temporary */
/* object. The return code should be equal to 8 and control   */
/* will be returned to the program.                           */
offset = 0; window_size = 1;
csrsave(&object_id, &offset, &window_size, &high_offset,;
        &return_code, &reason_code);
if (return_code != 8) {
    errflag = TRUE;
    printf("return_code was not set to proper value.\n");
}
/* Terminate the view to the window.                          */
offset = 0; window_size = 1;
csrview(op_type2,&object_id,&offset,&window_size,view,;
        usage, disposition, &return_code, &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Change values in the window array into 0's.                */
init_mem('0',view,4096);
/* View the window again.                                     */
offset = 0; window_size = 1;
csrview(op_type1,&object_id,&offset,&window_size,view,;
        usage, disposition, &return_code, &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* The values in the window should remain to 1's.             */
for (i=0; i<4096; i++) {
    if (errflag == TRUE) printf("%d %c  ", i, view[i]);
    if (view[i] != '1') errflag = TRUE;
}
/* Refresh the window to 0's.                                 */
offset = 0; window_size = 1;
csrrefr(&object_id, &offset, &window_size,;
        &return_code, &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* The values inside the window should equal to 0's.          */
for (i=0; i<4096; i++) {
    if (errflag == TRUE) printf("%d %c  ", i, view[i]);
    if (view[i] != 0) errflag = TRUE;
}
/* Terminate the view to the window.                          */
offset = 0; window_size = 1;
csrview(op_type2,&object_id,&offset,&window_size,view,;
        usage, disposition, &return_code, &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Terminate the access to the Hiperspace object.             */
csridac(op_type2, object_type, object_name, scroll_area, object_state,
        access_mode,&object_size,&object_id,&high_offset,&return_code,;
        &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Report the status of the test.                             */
if (errflag) {
```

C/370 Example

```
      printf("Test failed at line %d\n", __LINE__);
      exit(1);
  }
  else {
      printf("Test successful : %s\n", __FILE__);
      exit(0);
  }
}
/* Functions that will be used in the program.                  */
/* chk_code will check return code and reason code returned from*/
/* the Callable Services. It will report an error if the code(s)*/
/* is not equal to 0.                                           */
int chk_code(long int ret, long int reason, int linenumber)
{
    if (ret != 0)
        printf("return_code = %ld instead of 0 at line %d\n",
                ret, linenumber);
    if (reason != 0)
        printf("reason_code = %ld instead of 0 at line %d\n",
                reason, linenumber);
}
/* init_mem will initialize a block of memory starting at a     */
/* given location to a specified value.                         */
void init_mem(char init_val, char *low_mem, int size)
{
    int i;
    for (i=0; i<size; i++) *(low_mem+i) = init_val;
}
//*
//*-------------------------------------------------------------------
//* JCL USED TO COMPILE, LINK, AND, EXECUTE THE C/370 PROGRAM
//*-------------------------------------------------------------------
//*
//DPTTST1A JOB 'DPT04P,DPT,?,S=I','DPTTST1',MSGCLASS=H,
//        CLASS=J,NOTIFY=DPTTST1,MSGLEVEL=(1,1)
//CC      EXEC EDCC,INFILE='DPTTST1.DWS.SOURCE(DWS1)',
//        CPARM='NOOPT,SOURCE,NOSEQ,NOMAR',
//        OUTFILE='DPTTST1.DWS.OBJECT(DWS1)'
//*-------------------------------------------------------------------
//* LINK STEP
//*-------------------------------------------------------------------
//LKED    EXEC PGM=IEWL,PARM='MAP,RMODE=ANY,AMODE=31'
//SYSLIB  DD DSN=CEE.SCEELKED,DISP=SHR
//        DD DSN=SYS1.CSSLIB,DISP=SHR
//OBJECT  DD DSN=DPTTST1.DWS.OBJECT,DISP=SHR
//SYSLIN  DD *
   ENTRY CEESTART
   INCLUDE OBJECT(DWS1)
   NAME DWS1(R)
//SYSLMOD   DD DSN=DPTTST1.DWS.LOAD,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSUT1    DD DSN=&&SYSUT1,UNIT=SYSDA,DISP=(NEW,DELETE,DELETE),
//    SPACE=(32000,(30,30))
//*-------------------------------------------------------------------
//* GO STEP.  THIS STEP DEFINES A NAME FOR A PERMANENT OBJECT THAT
//* THE DDNAME OBJECT TYPE WILL REFERENCE.
//*-------------------------------------------------------------------
//GO      EXEC PGM=DWS1,REGION=4M
//STEPLIB   DD DSN=CEE.SCEERUN,DISP=SHR
//          DD DSN=DPTTST1.DWS.LOAD,DISP=SHR
//SYSPRINT  DD SYSOUT=*,DCB=(RECFM=VB,LRECL=125,BLKSIZE=6000)
//PLIDUMP   DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//DD1       DD DSN=DPTTST1.DWS.FILE1,DISP=SHR
```

COBOL Example

```
       IDENTIFICATION DIVISION.
      ****************************************************************
      * Program using COBOL to create a 40-page window               *
      * aligned on a page boundary. This is done by locating a       *
      * page boundary within a 40*4096+4095 byte work area.          *
      * The DWS interface validation routine is then called passing  *
      * the 40 page window.                                          *
      ****************************************************************
       PROGRAM-ID. DWSCBSAM.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       1   WORKAREA.
         2 FILLER PIC X OCCURS 167935 TIMES.
       PROCEDURE DIVISION.
           DISPLAY " DWSCBSAM CALLING DWSCB4K "
           CALL "DWSCB4K" USING WORKAREA
           DISPLAY " DWSCBSAM BACK FROM DWSCB4K "
           GOBACK.
-----------------------------------------------------------------------
-----------------------------------------------------------------------
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DWSCB4K.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       1   P POINTER.
       1   PR REDEFINES P PIC 9(9) COMP.
       1   DUMMY  PIC 9(9) COMP.
       1   R      PIC 9(9) COMP.
       LINKAGE SECTION.
       1   INWORK PIC X(167935).
       1   WINDOW.
           2 FILLER PIC X(4096) OCCURS 40 TIMES.
       PROCEDURE DIVISION USING INWORK.
           SET P TO ADDRESS OF INWORK
           DIVIDE PR BY 4096
             GIVING DUMMY
             REMAINDER R
           IF R NOT EQUAL 0 THEN
           COMPUTE PR = PR + 4096 - R
           SET ADDRESS OF WINDOW TO P
           DISPLAY " DWSCBK4 CALLING DWSCB2 "
           CALL "DWSCB2" USING WINDOW.
           DISPLAY " DWSCBK4 BACK FROM  DWSCB2 "
           GOBACK.
-----------------------------------------------------------------------
-----------------------------------------------------------------------
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DWSCB2.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
      * WINDOW SIZE CHOSEN TO BE 40 PAGES
       1   NWINPG PIC 9(9) COMP VALUE 40.
       1   NWINEL PIC 9(9) COMP.
       1   NWLAST PIC 9(9) COMP.
       1   NOBJPG PIC 9(9) COMP.
      * WINDOWS WILL BEGIN ORIGIN-ING AT OFFSET 0 IN DATA OBJECT
       1   WINOFF PIC 9(9) COMP VALUE 0.
       1   RETRN1 PIC 9(9) COMP.
       1   REASON PIC 9(9) COMP.
       1   NEWOFF PIC 9(9) COMP.
       1   OBSIZ  PIC 9(9) COMP.
       1   TOKEN PIC X(8).
```

COBOL Example

```
 1   K      PIC 9(9) COMP.
 LINKAGE SECTION.
 1   WINDOW.
     2 FILLER PIC X(4096) OCCURS 40 TIMES.
 1   WINDOW-ARRAY REDEFINES WINDOW.
     2 A PIC S9(8) COMP OCCURS 40960 TIMES.
 PROCEDURE DIVISION USING WINDOW.
     DISPLAY "Begin Data Windowing Services Interface Validation"
* WINDOW COMPOSED OF 4-BYTE ELEMENTS
     COMPUTE NWINEL  = 1024 * NWINPG.
* WINDOW MAY NOT BEGIN AT ARRAY ELEMENT 1, SO LEAVE ROOM
     COMPUTE NWLAST = 1024 * NWINPG + 1023
* IN THE FOLLOWING, ARBITRARILY SET OBJECT SIZE = 3 WINDOWS WORTH
     COMPUTE NOBJPG = 3 * NWINPG
* SET UP ACCESS TO A HIPERSPACE OBJECT
     CALL "CSRIDAC" USING
       BY CONTENT
         "BEGIN",
         "TEMPSPACE",
         "MY FIRST HIPERSPACE",
         "YES",
         "NEW",
         "UPDATE",
       BY REFERENCE
          NOBJPG,
          TOKEN,
          OBSIZ,
          RETRN1,
          REASON
* PUT SOME DATA INTO THE WINDOW AREA
     MOVE ALL "DATA" TO WINDOW
* NOW VIEW SOMETHING IN THE WINDOW
     CALL "CSRVIEW" USING
       BY CONTENT
         "BEGIN",
       BY REFERENCE
          TOKEN,
          WINOFF,
          NWINPG,
          WINDOW,
       BY CONTENT
         "RANDOM",
         "REPLACE",
       BY REFERENCE
          RETRN1,
          REASON
* CALCULATE SOMETHING IN THE WINDOW AREA
     PERFORM VARYING K FROM 1 BY 1 UNTIL K = NWINEL
        MOVE K TO A(K)
     END-PERFORM
* CAPTURE THE VIEW IN THE WINDOW
     CALL "CSRSCOT" USING
          TOKEN,
          WINOFF,
          NWINPG,
          RETRN1,
          REASON
* END THE VIEW IN THE WINDOW
     CALL "CSRVIEW" USING
       BY CONTENT
         "END ",
       BY REFERENCE
          TOKEN,
          WINOFF,
          NWINPG,
          WINDOW,
       BY CONTENT
```

```
          "RANDOM",
          "RETAIN ",
        BY REFERENCE
           RETRN1,
           REASON
* NOW VIEW SOMETHING ELSE (2ND WINDOW"S WORTH OF DATA) IN WINDOW
     ADD NWINPG TO WINOFF
     CALL "CSRVIEW" USING
       BY CONTENT
         "BEGIN",
       BY REFERENCE
          TOKEN,
          WINOFF
          NWINPG,
          WINDOW,
       BY CONTENT
         "RANDOM",
         "RETAIN",
       BY REFERENCE
          RETRN1,
          REASON
* CALCULATE SOMETHING NEW IN THE WINDOW AREA
     PERFORM VARYING K FROM 1 BY 1 UNTIL K = NWINEL
        COMPUTE A(K) = - K
     END-PERFORM
* SAVE THE DATA IN THE WINDOW
     CALL "CSRSCOT" USING
          TOKEN,
          WINOFF,
          NWINPG,
          RETRN1,
          REASON
* NOW END THE CURRENT VIEW IN WINDOW
     CALL "CSRVIEW" USING
       BY CONTENT
         "END ",
       BY REFERENCE
          TOKEN,
          WINOFF
          NWINPG,
          WINDOW,
       BY CONTENT
         "RANDOM",
         "RETAIN ",
       BY REFERENCE
          RETRN1,
          REASON
* NOW GO BACK TO THE FIRST VIEW IN THE WINDOW
     MOVE 0 TO WINOFF
     CALL "CSRVIEW" USING
       BY CONTENT
         "BEGIN",
       BY REFERENCE
          TOKEN,
          WINOFF,
          NWINPG,
          WINDOW,
       BY CONTENT
         "RANDOM",
         "REPLACE",
       BY REFERENCE
          RETRN1,
          REASON
* REFRESH THE DATA IN THE WINDOW FOR THIS VIEW
     CALL "CSRREFR" USING
          TOKEN,
          WINOFF,
```

COBOL Example

```
                NWINPG,
                RETRN1,
                REASON
      * NOW END THE VIEW IN THE WINDOW
           CALL "CSRVIEW" USING
             BY CONTENT
               "END ",
             BY REFERENCE
                TOKEN,
                WINOFF,
                NWINPG,
                WINDOW,
             BY CONTENT
               "RANDOM",
               "RETAIN ",
             BY REFERENCE
                RETRN1,
                REASON
      * TERMINATE ACCESS TO THE HIPERSPACE OBJECT
           CALL "CSRIDAC" USING
             BY CONTENT
               "END  ",
               "TEMPSPACE",
               "MY FIRST HIPERSPACE ENDS HERE ",
               "YES",
               "NEW",
               "UPDATE",
             BY REFERENCE
                NOBJPG,
                TOKEN,
                OBSIZ,
                RETRN1,
                REASON
           DISPLAY "-*** Run ended with Object Size in pages = " NEWOFF
           GOBACK
*************************************************
*                                               *
*          JCL FOR COBOL EXAMPLE                *
*                                               *
*************************************************
//JOB1XXX JOB 'A9907P,B9222095',                                        00010000
//  'A.A.USER',RD=R,                                                    00020000
//  MSGCLASS=H,NOTIFY=AAUSER,                                           00030000
//  MSGLEVEL=(1,1),CLASS=7                                              00040000
//LKED   EXEC  PGM=IEWL,PARM='SIZE=(1024K,512K),LIST,XREF,LET,MAP',     00080000
//             REGION=1024K                                           00090000
//SYSLIN   DD  DDNAME=SYSIN                                           00110000
//SYSLMOD  DD  DSNAME=AAUSER.USER.LOAD(CRTCON01),DISP=SHR             00120000
//SYSLIB   DD  DSNAME=CEE.SCEELED,DISP=SHR                            00140000
//*                                                                   00150100
//*   FF310.OBJ HOLDS OBJECT CODE FROM THE COMPILE                    00150200
//*                                                                   00150300
//MYLIB    DD  DSN=AAUSER.FF310.OBJ,DISP=SHR                          00151000
//*                                                                   00151100
//*   THE CSR STUBS ARE IN SYS1.CSSLIB                                00151200
//*                                                                   00151300
//INLIB    DD  DSN=SYS1.CSSLIB,DISP=SHR                               00152000
//SYSPRINT DD  SYSOUT=*                                               00170000
//SYSIN    DD *                                                       00230000
 INCLUDE MYLIB(DWSCBSAM,DWSCB4K,DWSCB2)                               00231000
 LIBRARY INLIB(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC)       00240000
 NAME CRTCON01(R)                                                     00250000
```

FORTRAN Example

```
*********************************************************************
*                                                                   *
*                                                                   *
*      FORTRAN EXAMPLE.  THE FORTRAN EXAMPLE IS FOLLOWED BY AN      *
*      ASSEMBLER PROGRAM CALLED ADDR.  YOU MUST LINKEDIT THIS       *
*      ASSEMBLER PROGRAM WITH THE FORTRAN PROGRAM OBJECT            *
*      CODE AND THE CSR STUBS.  THE ASSEMBLER PROGRAM ENSURES       *
*      THAT YOUR WINDOW IS ALIGNED ON A 4K BOUNDARY .               *
*                                                                   *
*********************************************************************
@PROCESS DC(WINCOM)
      PROGRAM CRTFON01
C
C     Test Program for Data Window Services
C
C     Window size chosen to be 40 pages
      PARAMETER (NWINPG = 40)
C     Window composed of 4-byte elements
      PARAMETER (NWINEL = 1024*NWINPG)
C     Window may not begin at array element 1, so leave room
      PARAMETER (NWLAST = 1024*NWINPG+1023)
C     In the following, arbitrarily set object size = 3 windows worth
      PARAMETER (NOBJPG = 3*NWINPG)
C     Windows will begin origin-ing at offset 0 in data object
      INTEGER WINOFF
      PARAMETER (WINOFF = 0)
C
      INTEGER RETRN1, REASON, HIOFF, NEWOFF, OBSIZ, OFF
      INTEGER ADDR, PAGE, A
      INTEGER JUNK /-1599029040/
      REAL*8 TOKEN
      COMMON /WINCOM/ A(NWLAST)
C
C
      WRITE (6, 91)
   91 FORMAT('1*** Begin Data Windowing Services Interface Validation')
C
C     Set up access to a Hiperspace object
      CALL CSRIDAC('BEGIN',
     *             'TEMPSPACE',
     *             'MY FIRST HIPERSPACE',
     *             'YES',
     *             'NEW',
     *             'UPDATE',
     *              NOBJPG,
     *              TOKEN,
     *              OBSIZ,
     *              RETRN1,
     *              REASON )
C
C     Determine first page-boundary element in Window Array "A"
      PAGE = ADDR(A(1))
      PAGE = MOD(PAGE, 4096)
      IF (PAGE .NE. 0) PAGE = (4096 - PAGE) / 4
      PAGE = PAGE + 1
C
C     Put data into the window
      DO 100 K = 1, NWINEL
        A(K+PAGE-1) = JUNK
  100 CONTINUE
C
C     Now view data in the window
      CALL CSRVIEW('BEGIN',
     *             TOKEN,
     *             WINOFF,
```

FORTRAN Example

```
     *              NWINPG,
     *              A(PAGE),
     *             'RANDOM',
     *             'REPLACE',
     *              RETRN1,
     *              REASON )
C
C     Calculate a value in the window area
      DO 101 K = 1, NWINEL
        A(K+PAGE-1) = K
  101 CONTINUE
C
C     Capture the view in the window
      CALL CSRSCOT( TOKEN,
     *              WINOFF,
     *              NWINPG,
     *              RETRN1,
     *              REASON )
C
C     End the view in the window
      CALL CSRVIEW('END ',
     *              TOKEN,
     *              WINOFF,
     *              NWINPG,
     *              A(PAGE),
     *             'RANDOM',
     *             'RETAIN ',
     *              RETRN1,
     *              REASON )
C
C     Now view other data (2nd window's worth of data) in window
      CALL CSRVIEW('BEGIN',
     *              TOKEN,
     *              WINOFF + NWINPG,
     *              NWINPG,
     *              A(PAGE),
     *             'RANDOM',
     *             'REPLACE',
     *              RETRN1,
     *              REASON )
C
C     Calculate a new value in the window
      DO 102 K = 1, NWINEL
        A(K+PAGE-1) = -K
  102 CONTINUE
C
C     Capture the view in the window
      CALL CSRSCOT( TOKEN,
     *              WINOFF + NWINPG,
     *              NWINPG,
     *              RETRN1,
     *              REASON )
C
C     Now end the current view in window
      CALL CSRVIEW('END ',
     *              TOKEN,
     *              WINOFF + NWINPG,
     *              NWINPG,
     *              A(PAGE),
     *             'RANDOM',
     *             'RETAIN ',
     *              RETRN1,
     *              REASON )
C
C     Now go back to the first view in the window
      CALL CSRVIEW('BEGIN',
     *              TOKEN,
```

```
     *              WINOFF,
     *              NWINPG,
     *              A(PAGE),
     *             'RANDOM',
     *             'REPLACE',
     *              RETRN1,
     *              REASON )
C
C     Refresh the data in the window for this view
      CALL CSRREFR( TOKEN,
     *              WINOFF,
     *              NWINPG,
     *              RETRN1,
     *              REASON )
C
C     Now end the view in the window
      CALL CSRVIEW('END ',
     *              TOKEN,
     *              WINOFF,
     *              NWINPG,
     *              A(PAGE),
     *             'RANDOM',
     *             'RETAIN ',
     *              RETRN1,
     *              REASON )
C
C     Terminate access to the Hiperspace object
      CALL CSRIDAC('END  ',
     *             'TEMPSPACE',
     *             'MY FIRST HIPERSPACE ENDS HERE ',
     *             'YES',
     *             'NEW',
     *             'UPDATE',
     *              NOBJPG,
     *              TOKEN,
     *              OBSIZ,
     *              RETRN1,
     *              REASON )
C
      STOP
      END
*********************************************************************
*                                                                   *
*                                                                   *
*     THIS ASSEMBLER PROGRAM ENSURES THAT YOUR WINDOW IS ALIGNED    *
*     ON A 4K BOUNDARY.  ASSEMBLE THIS PROGRAM AND LINKEDIT THE     *
*     OBJECT CODE WITH THE FORTRAN CODE AND THE CSR STUBS.          *
*                                                                   *
*********************************************************************
ADDR     TITLE 'LOC/ADDR Function for Fortran'
*
*        Calling Sequence:
*
*        INTEGER ADDR
*        - - -
*        L = LOC(x)
*        L = ADDR(x)
*
*        Returns address of "x" in R0, with high-order bit set to zero
*
ADDR     CSECT
         ENTRY LOC
LOC      EQU   *
         USING *,15
         L     0,0(,1)             Get pointer to x
         N     0,MASK              Set sign bit to 0
         BR    14                  Return
```

FORTRAN Example

```
MASK     DC    A(X'7FFFFFFF')      Mask with high-order bit 0
         END
*********************************************************************
*                                                                   *
*      JCL TO COMPILE AND LINKEDIT THE ASSEMBLER PROGRAM, THE       *
*      FORTRAN PROGRAM, AND THE STUBS.                              *
*                                                                   *
*********************************************************************
//FORTJOB  JOB                                                          00255013
//*                                                                     00003100
//*                                                                     00003100
//*  Compile and linkedit for FORTRAN                                   00003100
//*                                                                     00003100
//*                                                                     00003100
//VSF2CL PROC  FVPGM=FORTVS2,FVREGN=2100K,FVPDECK=NODECK,               00001000
//         FVPOLST=NOLIST,FVPOPT=0,FVTERM='SYSOUT=A',                   00002000
//         PGMNAME=MAIN,PGMLIB='&&GOSET',FVLNSPC='3200,(25,6)'          00003000
//*                                                                     00003100
//*            PARAMETER  DEFAULT-VALUE     USAGE                     00003900
//*                                                                     00004000
//*              FVPGM    FORTVS2           COMPILER NAME             00005000
//*              FVREGN   2100K             FORT-STEP REGION          00006000
//*              FVPDECK  NODECK            COMPILER DECK OPTION      00007000
//*              FVPOLST  NOLIST            COMPILER LIST OPTION      00008000
//*              FVPOPT   0                 COMPILER OPTIMIZATION     00009000
//*              FVTERM   SYSOUT=A          FORT.SYSTERM OPERAND      00010000
//*              FVLNSPC  3200,(25,6)       FORT.SYSLIN SPACE         00011000
//*              PGMLIB   &&GOSET           LKED.SYSLMOD DSNAME       00012000
//*              PGMNAME  MAIN              LKED.SYSLMOD MEMBER NAME  00013000
//*                                                                    00014000
//FORT   EXEC  PGM=&FVPGM,REGION=&FVREGN,COND=(4,LT),                  00015000
//             PARM='&FVPDECK,&FVPOLST,OPT(&FVPOPT)'                   00016000
//STEPLIB      DD DSN=HLLDS.FORT230.VSF2COMP,DISP=SHR                  00017000
//SYSPRINT     DD SYSOUT=A,DCB=BLKSIZE=3429                            00018000
//SYSTERM      DD &FVTERM                                              00019000
//SYSPUNCH     DD SYSOUT=B,DCB=BLKSIZE=3440                            00020000
//SYSLIN       DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,            00021000
//             SPACE=(&FVLNSPC),DCB=BLKSIZE=3200                       00022000
//LKED   EXEC  PGM=HEWL,REGION=768K,COND=(4,LT),                       00023000
//             PARM='LET,LIST,XREF'                                    00024000
//SYSPRINT     DD SYSOUT=A                                             00025000
//SYSLIB       DD DSN=CEE.SCEELKED,DISP=SHR                            00026000
//SYSUT1       DD UNIT=SYSDA,SPACE=(1024,(200,20))                     00027000
//SYSLMOD      DD DSN=&PGMLIB.(&PGMNAME),DISP=(,PASS),UNIT=SYSDA,      00028000
//             SPACE=(TRK,(10,10,1),RLSE)                              00029000
//SYSLIN       DD DSN=&&LOADSET,DISP=(OLD,DELETE)                      00030000
//             DD DDNAME=SYSIN                                         00040000
// PEND
//       EXEC  VSF2CL,FVTERM='SYSOUT=H',
//         PGMNAME=CRTFON01,PGMLIB='WINDOW.USER.LOAD'                  00003000
//FORT.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTFON01),DISP=SHR
//LKED.SYSLIB       DD DSN=CEE.SCEELKED,DISP=SHR                      00026000
//LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,DISP=SHR,UNIT=3380,
// VOL=SER=VM2TSO
//LKED.SYSIN DD *
  LIBRARY  IN(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC,ADDR)
  NAME CRTFON01(R)
/*
//*     The CSR stubs are available in SYS1.CSSLIB.
//*     The object code for the ADDR routine is in
//*     TEST.OBJ
//*
//LKED.IN     DD DSN=SYS1.CSSLIB,DISP=SHR
//             DD DSN=WINDOW.TEST.OBJ,DISP=SHR
//*
//*
*********************************************************************
```

```
*                                                                     *
*      JCL TO EXECUTE THE FORTRAN PROGRAM.                            *
*                                                                     *
***********************************************************************
//FON01  JOB  MSGLEVEL=(1,1)
//VSF2G  PROC  GOPGM=MAIN,GOREGN=100K,                                  00001000
//             GOF5DD='DDNAME=SYSIN',                                   00002000
//             GOF6DD='SYSOUT=A',                                       00003000
//             GOF7DD='SYSOUT=B'                                        00004000
//*                                                                     00005000
//*            PARAMETER  DEFAULT-VALUE     USAGE                       00007000
//*                                                                     00008000
//*              GOPGM    MAIN              PROGRAM NAME                00009000
//*              GOREGN   100K              GO-STEP REGION              00010000
//*              GOF5DD   DDNAME=SYSIN      GO.FT05F001 DD OPERAND      00011000
//*              GOF6DD   SYSOUT=A          GO.FT06F001 DD OPERAND      00012000
//*              GOF7DD   SYSOUT=B          GO.FT07F001 DD OPERAND      00013000
//*                                                                     00014000
//*                                                                     00015000
//GO     EXEC  PGM=&GOPGM,REGION=&GOREGN,COND=(4,LT)                    00016000
//STEPLIB      DD DSN=CEE.SCEERUN,DISP=SHR                            00017000
//FT05F001     DD &GOF5DD                                             00018000
//FT06F001     DD &GOF6DD                                             00019000
//FT07F001     DD &GOF7DD                                             00020000
// PEND
//GO EXEC VSF2G,GOPGM=CRTFON01,GOREGN=999K
//GO.STEPLIB      DD DSN=CEE.SCEERUN,DISP=SHR                         00017000
// DD DSN=WINDOW.USER.LOAD,DISP=SHR,VOL=SER=VM2TSO,UNIT=3380
```

Pascal Example

```
************************************************************************
*                                                                      *
*                                                                      *
*     PASCAL example. The data object is permanent and already         *
*     allocated.  A scroll area is used.                               *
*                                                                      *
*                                                                      *
*                                                                      *
************************************************************************
program CRTPAN06;
const
  K = 1024;                           (* One kilo-byte              *)
  PAGESIZE = 4 * K;                   (* 4K page boundary           *)
  OFFSET = 0;                         (* Windows starts             *)
  WINDOW_SIZE = 40;                   (* Window size in pages       *)
  NUM_WIN_ELEM = WINDOW_SIZE*K;       (* Num of 4-byte elements     *)
  OBJECT_SIZE = 3*WINDOW_SIZE;        (* Chosen object size in pages*)
  SPACE_SIZE = (WINDOW_SIZE+1)*4*K;   (* Space allocated for window *)
type
  S = space[SPACE_SIZE] of INTEGER;   (* Define byte aligned space  *)
  STR3 = packed array (. 1..3 .) of CHAR;
  STR5 = packed array (. 1..5 .) of CHAR;
  STR6 = packed array (. 1..6 .) of CHAR;
  STR7 = packed array (. 1..7 .) of CHAR;
  STR9 = packed array (. 1..9 .) of CHAR;
  STR44 = packed array (. 1..44 .) of CHAR;
var
  SP : @S;                            (* Declare pointer to space   *)
  ORIG,                               (* Start address of window    *)
  AD, I,                              (* Temporary variables        *)
  VOFFSET,                            (* Offset passed as parameter *)
  VOFSET2,                            (* Offset passed as parameter *)
  VOBJSIZ,                            (* Object size,  as parameter *)
  VWINSIZ,                            (* Window Size,  as parameter *)
  HIGH_OFFSET,                        (* Size of object in pages    *)
  NEW_HI_OFFSET,                      (* New max size of the object *)
```

Pascal Example

```
  RETURN_CODE,                          (* Return code               *)
  REASON_CODE : INTEGER;                (* Reason code               *)
  OBJECT_ID   : REAL;                   (* Identifying token         *)
  CSCROLL     : STR3;                   (* Scroll area YES/NO        *)
  COBSTATE    : STR3;                   (* Object state NEW/OLD      *)
  COPTYPE     : STR5;                   (* Operation type BEGIN/END  *)
  CACCESS     : STR6;                   (* Access RANDOM/SEQ         *)
  CUSAGE      : STR6;                   (* Usage READ/UPDATE         *)
  CDISP       : STR7;                   (* Disposition RETAIN/REPLACE *)
  CSPTYPE     : STR9;         (* Object type DSNAME/DDNAME/TEMPSPACE *)
  COBNAME     : STR44;                  (* Object name               *)
procedure CSRIDAC ( var OP_TYPE : STR5;
                    var OBJECT_TYPE : STR9;
                    var OBJECT_NAME : STR44;
                    var SCROLL_AREA : STR3;
                    var OBJECT_STATE : STR3;
                    var ACCESS_MODE : STR6;
                    var VOBJSIZ : INTEGER;
                    var OBJECT_ID : REAL;
                    var HIGH_OFFSET : INTEGER;
                    var RETURN_CODE : INTEGER;
                    var REASON_CODE : INTEGER);  FORTRAN;
procedure CSRVIEW ( var OP_TYPE : STR5;
                    var OBJECT_ID : REAL;
                    var OFFSET : INTEGER;
                    var WINDOW_SIZE : INTEGER;
                    var WINDOW_NAME : INTEGER;
                    var USAGE : STR6;
                    var DISPOSITION : STR7;
                    var RETURN_CODE : INTEGER;
                    var REASON_CODE : INTEGER);  FORTRAN;
procedure CSRSCOT ( var OBJECT_ID : REAL;
                    var OFFSET : INTEGER;
                    var SPAN : INTEGER;
                    var RETURN_CODE : INTEGER;
                    var REASON_CODE : INTEGER );  FORTRAN;
procedure CSRSAVE ( var OBJECT_ID : REAL;
                    var OFFSET : INTEGER;
                    var SPAN : INTEGER;
                    var NEW_HI_OFFSET : INTEGER;
                    var RETURN_CODE : INTEGER;
                    var REASON_CODE : INTEGER );  FORTRAN;
procedure CSRREFR ( var OBJECT_ID : REAL;
                    var OFFSET : INTEGER;
                    var SPAN : INTEGER;
                    var RETURN_CODE : INTEGER;
                    var REASON_CODE : INTEGER );  FORTRAN;
begin
  TERMOUT(OUTPUT);                      (* Output to terminal        *)
  WRITELN ('<< Begin Data Windowing Services Interface Validation >>');
  WRITELN;
  VOBJSIZ := OBJECT_SIZE;               (* Set object size variable  *)
  VOFFSET := OFFSET;                    (* Set offset variable to 0  *)
  VWINSIZ := WINDOW_SIZE;               (* Set window size variable  *)
  VOFSET2 := OFFSET+WINDOW_SIZE;        (* Set offset variable to 0  *)
  COPTYPE  := 'BEGIN' ;
  CSPTYPE  := 'DDNAME   ' ;
  COBNAME  := 'CSRDD1 ' ;
  CSCROLL  := 'YES' ;
  COBSTATE := 'NEW' ;
  CACCESS  := 'UPDATE' ;
  CSRIDAC (COPTYPE,                     (* Set up access to a        *)
           CSPTYPE,                     (* hiperspace object         *)
           COBNAME,
           CSCROLL,
           COBSTATE,
           CACCESS,
```

```
         VOBJSIZ,
         OBJECT_ID,
         HIGH_OFFSET,
         RETURN_CODE,
         REASON_CODE);
NEW(SP);                              (* Allocate space             *)
AD := ADDR(SP@); (* or ORD(SP) *)    (* Get address of space       *)
ORIG := AD mod PAGESIZE;              (* See where space is in page *)
if ORIG <> 0 then                     (* If not on page boundary    *)
  ORIG := PAGESIZE-ORIG;              (* then locate page boundary  *)
for I := 0 to NUM_WIN_ELEM-1 do       (* Put data into window       *)
  SP@[4*I+ORIG] := 999999;            (* area                       *)
COPTYPE  := 'BEGIN' ;
CUSAGE   := 'RANDOM' ;
CDISP    := 'REPLACE' ;
CSRVIEW (COPTYPE,                     (* Now view data in 1st       *)
         OBJECT_ID,                   (* window                     *)
         VOFFSET,
         VWINSIZ,
         SP@[ORIG],
         CUSAGE,
         CDISP,
         RETURN_CODE,
         REASON_CODE);
for I := 0 to NUM_WIN_ELEM-1 do       (* Calculate a value in 1st  *)
  SP@[4*I+ORIG] := I+1;               (* window                    *)
CSRSCOT( OBJECT_ID,                   (* Capture the view in 1st   *)
         VOFFSET,                     (* window                    *)
         VWINSIZ,
         RETURN_CODE,
         REASON_CODE);
COPTYPE  := 'END' ;
CUSAGE   := 'RANDOM' ;
CDISP    := 'RETAIN' ;
CSRVIEW (COPTYPE,                     (* End the view in 1st window *)
         OBJECT_ID,
         VOFFSET,
         VWINSIZ,
         SP@[ORIG],
         CUSAGE,
         CDISP,
         RETURN_CODE,
         REASON_CODE);
COPTYPE  := 'BEGIN' ;
CUSAGE   := 'RANDOM' ;
CDISP    := 'REPLACE' ;
CSRVIEW (COPTYPE,                     (* Now view other data in the *)
         OBJECT_ID,                   (* 2nd window                 *)
         VOFSET2,
         VWINSIZ,
         SP@[ORIG],
         CUSAGE,
         CDISP,
         RETURN_CODE,
         REASON_CODE);
for I := 0 to NUM_WIN_ELEM-1 do       (* Calculate a new value in   *)
  SP@[4*I+ORIG] := I-101;             (* the window                 *)
CSRSAVE (OBJECT_ID,
         VOFSET2,
         VWINSIZ,
         NEW_HI_OFFSET,
         RETURN_CODE,
         REASON_CODE);
COPTYPE  := 'END' ;
CUSAGE   := 'RANDOM' ;
CDISP    := 'RETAIN' ;
CSRVIEW (COPTYPE,                     (* End the current view in    *)
```

Pascal Example

```
                OBJECT_ID,                  (* window                     *)
                VOFSET2,
                VWINSIZ,
                SP@[ORIG],
                CUSAGE,
                CDISP,
                RETURN_CODE,
                REASON_CODE);
      COPTYPE  := 'BEGIN' ;
      CUSAGE   := 'RANDOM' ;
      CDISP    := 'REPLACE' ;
      CSRVIEW (COPTYPE,                   (* Now go back to the view in *)
                OBJECT_ID,                (* the 1st window             *)
                VOFFSET,
                VWINSIZ,
                SP@[ORIG],
                CUSAGE,
                CDISP,
                RETURN_CODE,
                REASON_CODE);
      CSRREFR (OBJECT_ID,                 (* Refresh the data in 1st   *)
                VOFFSET,                  (* window                    *)
                VWINSIZ,
                RETURN_CODE,
                REASON_CODE);
      COPTYPE  := 'END' ;
      CUSAGE   := 'RANDOM' ;
      CDISP    := 'RETAIN' ;
      CSRVIEW (COPTYPE,                   (* End the view in 1st window *)
                OBJECT_ID,
                VOFFSET,
                VWINSIZ,
                SP@[ORIG],
                CUSAGE,
                CDISP,
                RETURN_CODE,
                REASON_CODE);
      COPTYPE  := 'END' ;
      CSPTYPE  := 'DDNAME   ' ;
      COBNAME  := 'CSRDD1 ' ;
      CSCROLL  := 'YES' ;
      COBSTATE := 'NEW' ;
      CACCESS  := 'UPDATE' ;
      CSRIDAC (COPTYPE,                   (* Terminate access to the    *)
                CSPTYPE,                  (* Hiperspace object          *)
                COBNAME,
                CSCROLL,
                COBSTATE,
                CACCESS,
                VWINSIZ,
                OBJECT_ID,
                HIGH_OFFSET,
                RETURN_CODE,
                REASON_CODE);
end.
*********************************************************************
*                                                                   *
*       JCL to compile and linkedit                                 *
*                                                                   *
*********************************************************************
//PASC1JOB JOB                                                        00010005
//GO   EXEC PAS22CL                                                   00050000
//*                                                                   00050102
//*    Compile and linkedit for PASCAL                                00050202
//*                                                                   00050302
//PASC.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTPAN06),DISP=SHR              00060006
//LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,DISP=SHR,UNIT=3380,            00560000
```

```
// VOL=SER=VM2TSO                                                        00570000
//LKED.SYSIN DD *                                                        00580000
  LIBRARY  IN(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC)          00590000
  NAME CRTPAN06(R)                                                      00600006
/*                                                                      00610000
//*     SYS1.CSSLIB is the source of the CSR stubs                      00620002
//*                                                                     00650002
//LKED.IN     DD DSN=SYS1.CSSLIB,DISP=SHR                               00690000
*********************************************************************
*                                                                   *
*       JCL to execute.  A DD statement, CSRDD1, is needed to define *
*       the permanent object which already exists.                  *
*                                                                   *
*                                                                   *
*********************************************************************
//PASC2JOB JOB MSGLEVEL=(1,1)                                           00010000
//GO     EXEC  PGM=CRTPAN06                                             00020002
//STEPLIB DD DSN=WINDOW.PASCAL22.LINKLIB,                               00030000
// DISP=SHR,UNIT=3380,                                                  00040000
// VOL=SER=VM2TSO                                                       00050000
// DD DSN=WINDOW.USER.LOAD,                                             00060000
// DISP=SHR,UNIT=3380,                                                  00070000
// VOL=SER=VM2TSO                                                       00080000
//CSRDD1   DD  DSN=DIV.TESTDS01,DISP=SHR
//OUTPUT   DD  SYSOUT=A,DCB=(RECFM=VBA,LRECL=133)                       00090000
//SYSPRINT DD  SYSOUT=A,DCB=(RECFM=VBA,LRECL=133)                       00100000
```

PL/I Example

```
*********************************************************************
*                                                                   *
*      PL/I EXAMPLE                                                 *
*      OBJECT IS TEMPORARY                                          *
*                                                                   *
*                                                                   *
*                                                                   *
*********************************************************************
 CRTPLN3: PROCEDURE OPTIONS (MAIN);                                     CSR00010
                                                                        CSR00020
 DCL                                                                    CSR00030
  (                                                                     CSR00040
  K INIT(1024),                       /* ONE KILO-BYTE             */ CSR00050
  PAGESIZE INIT(4096),                /* 4K PAGE BOUNDARY          */ CSR00060
  OFFSET INIT(0),                     /* WINDOWS STARTS            */ CSR00070
  WINDOW_SIZE INIT(20),               /* WINDOW SIZE IN PAGES      */ CSR00080
  NUM_WIN_ELEM INIT (20480),          /* NUM OF 4-BYTE ELEMENTS    */ CSR00090
  OBJECT_SIZE INIT (60))              /* CHOSEN OBJECT SIZE IN PGS */ CSR00100
  FIXED BIN(31);                                                        CSR00110
                                                                        CSR00120
 DCL                                                                    CSR00130
 /* 32767 IS UPPER LIMIT FOR ARRAY BOUND.                          */ CSR00140
  S(32767) BIN(31) FIXED BASED(SP);     /* DEFINE WORD ALIGNED SPACE */ CSR00150
                                                                        CSR00160
 DCL SP PTR;                                                            CSR00170
                                                                        CSR00180
 DCL                                                                    CSR00190
  (                                                                     CSR00200
  ORIG,                               /* START ADDRESS OF WINDOW   */ CSR00210
  AD, I,                              /* TEMPORARY VARIABLES       */ CSR00220
  HIGH_OFFSET,                        /* SIZE OF OBJECT IN PAGES   */ CSR00230
  NEW_HI_OFFSET,                      /* NEW MAX SIZE OF THE OBJECT */ CSR00240
  RETURN_CODE,                        /* RETURN CODE               */ CSR00250
  REASON_CODE) FIXED BIN(31);         /* REASON CODE               */ CSR00260
                                                                        CSR00270
 DCL                                                                    CSR00280
  OBJECT_ID CHAR(8);                  /* IDENTIFYING TOKEN         */ CSR00290
                                                                        CSR00300
```

PL/I Example

```
/*****************************************************************/ CSR00310
                                                                    CSR00320
DCL CSRIDAC ENTRY(CHAR(5),            /* OP_TYPE                */ CSR00330
                  CHAR(9),            /* OBJECT_TYPE            */ CSR00340
                  CHAR(44),           /* OBJECT_NAME            */ CSR00350
                  CHAR(3),            /* SCROLL_AREA            */ CSR00360
                  CHAR(3),            /* OBJECT_STATE           */ CSR00370
                  CHAR(6),            /* ACCESS_MODE            */ CSR00380
                  FIXED BIN(31),      /* OBJECT_SIZE            */ CSR00390
                  CHAR(8),            /* OBJECT_ID              */ CSR00400
                  FIXED BIN(31),      /* HIGH_OFFSET            */ CSR00410
                  FIXED BIN(31),      /* RETURN_CODE            */ CSR00420
                  FIXED BIN(31) )     /* REASON_CODE            */ CSR00430
                  OPTIONS(ASSEMBLER);                               CSR00440
                                                                    CSR00450
                                                                    CSR00460
DCL CSRVIEW ENTRY(CHAR(5),            /* OP_TYPE                */ CSR00470
                  CHAR(8),            /* OBJECT_ID              */ CSR00480
                  FIXED BIN(31),      /* OFFSET                 */ CSR00490
                  FIXED BIN(31),      /* WINDOW_SIZE            */ CSR00500
                  FIXED BIN(31),      /* WINDOW_NAME            */ CSR00510
                  CHAR(6),            /* USAGE                  */ CSR00520
                  CHAR(7),            /* DISPOSITION            */ CSR00530
                  FIXED BIN(31),      /* RETURN_CODE            */ CSR00540
                  FIXED BIN(31) )     /* REASON_CODE            */ CSR00550
                  OPTIONS(ASSEMBLER);                               CSR00560
                                                                    CSR00570
                                                                    CSR00580
DCL CSRSCOT ENTRY(CHAR(8),            /* OBJECT_ID              */ CSR00590
                  FIXED BIN(31),      /* OFFSET                 */ CSR00600
                  FIXED BIN(31),      /* SPAN                   */ CSR00610
                  FIXED BIN(31),      /* RETURN_CODE            */ CSR00620
                  FIXED BIN(31) )     /* REASON_CODE            */ CSR00630
                  OPTIONS(ASSEMBLER);                               CSR00640
                                                                    CSR00650
                                                                    CSR00660
DCL CSRSAVE ENTRY(CHAR(8),            /* OBJECT_ID              */ CSR00670
                  FIXED BIN(31),      /* OFFSET                 */ CSR00680
                  FIXED BIN(31),      /* SPAN                   */ CSR00690
                  FIXED BIN(31),      /* NEW_HI_OFFSET          */ CSR00700
                  FIXED BIN(31),      /* RETURN_CODE            */ CSR00710
                  FIXED BIN(31) )     /* REASON_CODE            */ CSR00720
                  OPTIONS(ASSEMBLER);                               CSR00730
                                                                    CSR00740
                                                                    CSR00750
DCL CSRREFR ENTRY(CHAR(8),            /* OBJECT_ID              */ CSR00760
                  FIXED BIN(31),      /* OFFSET                 */ CSR00770
                  FIXED BIN(31),      /* SPAN                   */ CSR00780
                  FIXED BIN(31),      /* RETURN_CODE            */ CSR00790
                  FIXED BIN(31) )     /* REASON_CODE            */ CSR00800
                  OPTIONS(ASSEMBLER);                               CSR00810
                                                                    CSR00820
/*****************************************************************/ CSR00830
                                                                    CSR00840
                                                                    CSR00850
 PUT SKIP LIST                                                      CSR00860
        ('<< BEGIN DATA WINDOWING SERVICES INTERFACE VALIDATION >>'); CSR00870
 PUT SKIP LIST (' ');                                               CSR00880
                                                                    CSR00890
 CALL                                                               CSR00900
 CSRIDAC ('BEGIN',                    /* SET UP ACCESS TO A HIPER-  */ CSR00910
          'TEMPSPACE',                /* SPACE OBJECT               */ CSR00920
          'MY FIRST HIPERSPACE',                                    CSR00930
          'YES',                                                    CSR00940
          'NEW',                                                    CSR00950
          'UPDATE',                                                 CSR00960
          OBJECT_SIZE,                                              CSR00970
```

```
         OBJECT_ID,                                                CSR00980
         HIGH_OFFSET,                                              CSR00990
         RETURN_CODE,                                              CSR01000
         REASON_CODE);                                             CSR01010
                                                                   CSR01020
ALLOC S;                          /* ALLOCATE SPACE            */ CSR01030
AD = UNSPEC(SP);                  /* GET ADDRESS OF SPACE      */ CSR01040
ORIG = MOD(AD,PAGESIZE);          /* SEE WHERE SPACE IS IN PAGE */ CSR01050
IF ORIG ¬= 0 THEN                 /* IF NOT ON PAGE BOUNDARY   */ CSR01060
  ORIG = (PAGESIZE-ORIG) / 4;     /* THEN LOCATE PAGE BOUNDARY */ CSR01070
ORIG = ORIG + 1;                                                   CSR01080
                                                                   CSR01090
DO  I = 1 TO NUM_WIN_ELEM;        /* PUT SOME DATA INTO WINDOW */ CSR01100
  S(I+ORIG-1) = 99;               /* AREA                      */ CSR01110
END;                                                               CSR01120
                                                                   CSR01130
CALL                                                               CSR01140
CSRVIEW ('BEGIN',                 /* NOW VIEW DATA IN FIRST    */ CSR01150
         OBJECT_ID,               /* WINDOW                    */ CSR01160
         OFFSET,                                                   CSR01170
         WINDOW_SIZE,                                              CSR01180
         S(ORIG),                                                  CSR01190
         'RANDOM',                                                 CSR01200
         'REPLACE',                                                CSR01210
         RETURN_CODE,                                              CSR01220
         REASON_CODE);                                             CSR01230
                                                                   CSR01240
DO  I = 1 TO NUM_WIN_ELEM;        /* CALCULATE VALUE IN 1ST    */ CSR01250
  S(I+ORIG-1) = I+1;              /* WINDOW                    */ CSR01260
END;                                                               CSR01270
                                                                   CSR01280
CALL                                                               CSR01290
CSRSCOT( OBJECT_ID,               /* CAPTURE THE VIEW IN 1ST   */  CSR01300
         OFFSET,                  /* WINDOW                    */  CSR01310
         WINDOW_SIZE,                                              CSR01320
         RETURN_CODE,                                              CSR01330
         REASON_CODE);                                             CSR01340
                                                                   CSR01350
CALL                                                               CSR01360
CSRVIEW ('END ',                  /* END THE VIEW IN 1ST WINDOW */ CSR01370
         OBJECT_ID,                                                CSR01380
         OFFSET,                                                   CSR01390
         WINDOW_SIZE,                                              CSR01400
         S(ORIG),                                                  CSR01410
         'RANDOM',                                                 CSR01420
         'RETAIN ',                                                CSR01430
         RETURN_CODE,                                              CSR01440
         REASON_CODE);                                             CSR01450
                                                                   CSR01460
CALL                                                               CSR01470
CSRVIEW ('BEGIN',                 /* NOW VIEW OTHER DATA IN    */ CSR01480
         OBJECT_ID,               /* 2ND WINDOW                */ CSR01490
         OFFSET+WINDOW_SIZE,                                       CSR01500
         WINDOW_SIZE,                                              CSR01510
         S(ORIG),                                                  CSR01520
         'RANDOM',                                                 CSR01530
         'REPLACE',                                                CSR01540
         RETURN_CODE,                                              CSR01550
         REASON_CODE);                                             CSR01560
                                                                   CSR01570
DO  I = 1 TO NUM_WIN_ELEM;        /* CALCULATE NEW VALUE IN    */ CSR01580
  S(I+ORIG-1) = I-101;            /* WINDOW                    */ CSR01590
END;                                                               CSR01600
                                                                   CSR01610
CALL                                                               CSR01620
CSRSCOT (OBJECT_ID,                                                CSR01630
         OFFSET+WINDOW_SIZE,                                       CSR01640
```

PL/I Example

```
          WINDOW_SIZE,                                              CSR01650
          RETURN_CODE,                                              CSR01670
          REASON_CODE);                                             CSR01680
                                                                    CSR01690
  CALL                                                              CSR01700
  CSRVIEW ('END ',                  /* END THE CURRENT VIEW IN    */ CSR01710
          OBJECT_ID,                /* WINDOW                     */ CSR01720
          OFFSET+WINDOW_SIZE,                                       CSR01730
          WINDOW_SIZE,                                              CSR01740
          S(ORIG),                                                  CSR01750
          'RANDOM',                                                 CSR01760
          'RETAIN ',                                                CSR01770
          RETURN_CODE,                                              CSR01780
          REASON_CODE);                                             CSR01790
                                                                    CSR01800
  CALL                                                              CSR01810
  CSRVIEW ('BEGIN',                 /* NOW GO BACK TO THE VIEW IN */ CSR01820
          OBJECT_ID,                /* THE 1ST WINDOW             */ CSR01830
          OFFSET,                                                   CSR01840
          WINDOW_SIZE,                                              CSR01850
          S(ORIG),                                                  CSR01860
          'RANDOM',                                                 CSR01870
          'REPLACE',                                                CSR01880
          RETURN_CODE,                                              CSR01890
          REASON_CODE);                                             CSR01900
                                                                    CSR01910
  CALL                                                              CSR01920
  CSRREFR (OBJECT_ID,               /* REFRESH THE DATA IN 1ST   */  CSR01930
          OFFSET,                   /* WINDOW                    */  CSR01940
          WINDOW_SIZE,                                              CSR01950
          RETURN_CODE,                                              CSR01960
          REASON_CODE);                                             CSR01970
                                                                    CSR01980
  CALL                                                              CSR01990
  CSRVIEW ('END ',                  /* END THE VIEW IN 1ST WINDOW */ CSR02000
          OBJECT_ID,                                                CSR02010
          OFFSET,                                                   CSR02020
          WINDOW_SIZE,                                              CSR02030
          S(ORIG),                                                  CSR02040
          'RANDOM',                                                 CSR02050
          'RETAIN ',                                                CSR02060
          RETURN_CODE,                                              CSR02070
          REASON_CODE);                                             CSR02080
                                                                    CSR02090
  CALL                                                              CSR02100
  CSRIDAC ('END  ',                 /* TERMINATE ACCESS TO THE    */ CSR02110
          'TEMPSPACE',              /* HIPERSPACE OBJECT          */ CSR02120
          'MY FIRST HIPERSPACE ENDS HERE ',                         CSR02130
          'YES',                                                    CSR02140
          'NEW',                                                    CSR02150
          'UPDATE',                                                 CSR02160
          WINDOW_SIZE,                                              CSR02170
          OBJECT_ID,                                                CSR02180
          HIGH_OFFSET,                                              CSR02190
          RETURN_CODE,                                              CSR02200
          REASON_CODE);                                             CSR02210
                                                                    CSR02220
  FREE S;                                                           CSR02230
 END CRTPLN3;                                                       CSR02260
*********************************************************************
*                                                                   *
*                                                                   *
*      JCL TO COMPILE AND LINKEDIT PL/I PROGRAM.                    *
*                                                                   *
*                                                                   *
*                                                                   *
*********************************************************************
```

```
//PLIJOB   JOB                                                          00010007
//*                                                                     00041001
//*  PL/I Compile and Linkedit                                          00042001
//*                                                                     00043001
//*  Change all CRTPLNx to CRTPLNy                                      00044001
//*                                                                     00045001
//GO       EXEC PLIXCL                                                 00050000
//PLI.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTPLN3),DISP=SHR                 00060008
//LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,UNIT=3380,VOL=SER=VM2TSO,       00070000
// DISP=SHR                                                           00080000
//LKED.SYSIN DD *                                                     00090000
  LIBRARY  IN(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC)       00100001
  NAME CRTPLN3(R)                                                    00110008
/*                                                                    00120000
//*                                                                   00121001
//*     SYS1.CSSLIB is source of CSR stubs                            00130001
//*                                                                   00190000
//LKED.IN     DD DSN=SYS1.CSSLIB,DISP=SHR                             00200000
**********************************************************************
*                                                                    *
*                                                                    *
*      JCL TO EXECUTE.                                               *
*                                                                    *
*                                                                    *
*                                                                    *
**********************************************************************
//PLIRUN   JOB MSGLEVEL=(1,1)                                         00010000
//*                                                                   00011001
//*   EXECUTE A PL/I TESTCASE                                         00012001
//*                                                                   00013001
//GO       EXEC PGM=CRTPLN3                                           00020000
//STEPLIB  DD  DSN=WINDOW.USER.LOAD,DISP=SHR,                         00030000
// UNIT=3380,VOL=SER=VM2TSO                                           00040000
//SYSLIB   DD  DSN=CEE.SCEERUN,DISP=SHR                               00050000
//SYSABEND DD SYSOUT=*                                                00070000
//SYSLOUT  DD  SYSOUT=*                                               00080000
//SYSPRINT DD  SYSOUT=*                                               00090000
```

PL/I Example

Part 2. Reference Pattern Services

Chapter 5. Introduction to Reference Pattern Services

Reference pattern services allow HLL programs to define a reference pattern for a specified area of virtual storage that the program is about to reference. Additionally, the program specifies how much data it wants the operating system to bring into central storage at one time. Data and instructions in virtual storage must reside in central storage before they can be processed. The system honors the request according to the availability of central storage. By bringing in more data at one time, the system might improve the performance of your program.

The term **reference pattern** refers to the order in which a program's instructions process a range of data, such as an array or part of an array.

Programs that benefit most from reference pattern services are those that reference amounts of data that are greater than one megabyte. The program should reference the data in a sequential manner and in a consistent direction, either forward or backward. In forward direction, the program references data elements in order of ascending addresses. In backward direction, the program references data elements in order of decreasing addresses. In addition, if the program "skips over" certain areas, and these areas are of uniform size and are repeated at regular intervals throughout the area, reference pattern services might provide additional performance improvement.

Two reference pattern services are available through program CALLs:

- CSRIRP identifies the range of data and the reference pattern, and defines the number of bytes that the system is requested to bring into central storage at one time. These activities are called "defining the reference pattern".
- CSRRRP removes the definition; it tells the system that the program has stopped using the reference pattern with the range of data.

A program might have a number of different ways of referencing a particular area. In this case, the program can issue multiple pairs of CSRIRP and CSRRRP services for the area. Only one pattern can be in effect at a time.

Although reference pattern services can be used for data structures other than arrays, for simplicity, examples in this chapter and in the next use the services with arrays.

How Does the System Manage Data?

Before you can evaluate the performance advantage that reference pattern services offer, you must understand some facts about how the operating system handles the data your program references. The system divides the data into 4096-byte chunks; each chunk is called a "page". For the processor to execute an instruction, the page that contains the data that the instruction requires must reside in central storage. Central storage contains pages of data for many programs — your program, plus other programs that the system is working on. The system brings a page of your data into central storage when your program needs data on that page. If the program uses the data in a sequential manner, once the program finishes using the data on that page, it will not immediately use the page again. After your program finishes using that page, the system might remove the page from central storage to make room for another page of your data or maybe a page of some other

program's data. The system allows pages to stay in central storage if they are referenced frequently enough and if the system does not need those pages for other programs.

The process that the system goes through when it pauses to bring a page into central storage is called a "page fault". This interruption causes the system to stop working on your program (or "suspend" your program) while more of your program's data comes into central storage. Then, when the page is in central storage and the system is available to your program again, the system resumes running your program at the instruction where it left off.

Reference pattern services can change the way the system handles your program's data. With direction from reference pattern services, the system moves multiple pages into central storage at a time. By bringing in many pages at a time, the system takes fewer page faults. Fewer page faults mean possible performance gains for your program.

An Example of How the System Manages Data in an Array

To evaluate the performance advantage reference pattern services offers, you need to understand how the system handles a range of data. The best way to describe this is through an example of a simple two-dimensional array. As array A(i,j) of 3 rows and 4 columns illustrates, the system stores arrays in FORTRAN programs in column-major order and stores arrays in COBOL, Pascal, PL/1, and C programs in row-major order.

```
A(1,1)    A(1,2)    A(1,3)    A(1,4)
A(2,1)    A(2,2)    A(2,3)    A(2,4)
A(3,1)    A(3,2)    A(3,3)    A(3,4)
```

The system stores the elements of the arrays in the following order:

Sequence of Element in Storage	FORTRAN Array Element	COBOL, Pascal, PL/1, C Array Element
1	A(1,1)	A(1,1)
2	A(2,1)	A(1,2)
3	A(3,1)	A(1,3)
4	A(1,2)	A(1,4)
5	A(2,2)	A(2,1)
6	A(3,2)	A(2,2)
7	A(1,3)	A(2,3)
8	A(2,3)	A(2,4)
9	A(3,3)	A(3,1)
10	A(1,4)	A(3,2)
11	A(2,4)	A(3,3)
12	A(3,4)	A(3,4)

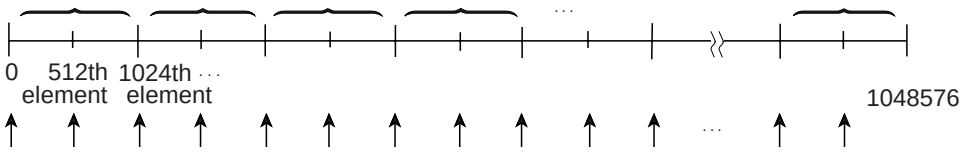
Examples in this chapter and the next depict data as a horizontal string. The elements in the arrays, therefore, would look like the following:

```
                    Location of elements
_____________________________________________________
1   2   3   4   5   6   7   8   9   10  11  12
```

Consider a two-dimensional array, ARRAY1, that has 1024 columns and 1024 rows and each element is eight bytes in size. The size of the array, therefore, is 1048576 elements or 8388608 bytes. For simplicity, assume the array is aligned on a page boundary. Also, assume the data is not in central storage. The program references each element in the array in a forward direction, starting with the first element.

First, consider how the system brings data into central storage without information from reference pattern services. At the first reference of ARRAY1, the system takes a page fault and brings into central storage the page (of 4096 bytes) that contains the first element. After the program finishes processing the 512th (4096 divided by 8) element in the array, the system takes another page fault and brings in a second page. The system takes a page fault every 512 elements, throughout the array.

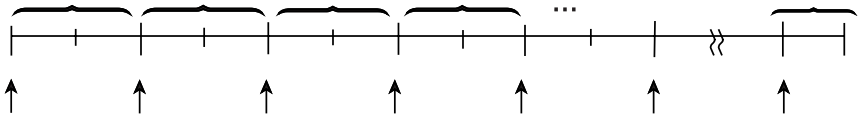
The following linear representation shows the elements in the array and the page faults the system takes as a program processes the array.



1 page fault each 512 elements (1 page)

By bringing in one page at a time, the system takes 2048 page faults (8388608 divided by 4096), each page fault adding to the elapsed time of the program.

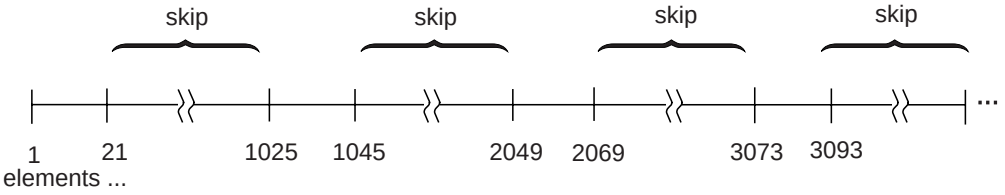
Suppose, through CSRIRP, the system knew in advance that a program would be using the array in a consistently forward direction. The system could then assume that the program's use of the pages of the array would be sequential. To decrease the number of page faults, each time the program requested data that was not in central storage, the system could bring in more than one page at a time. Suppose the system brought the next 20 consecutive pages (81920 bytes) of the array into central storage on each page fault. In this case, the system takes not 2048 page faults, but 103 (8388608 divided by 81920=102.4). Page faults occur in the array as follows:



1 page fault each 10240 elements (20 pages)

The system brings in successive pages only to the end of the array.

Consider another way of referencing ARRAY1. The program references the first twenty elements, then skips over the next 1004 elements, and so forth through the array. CSRIRP allows you to tell the system to bring in only the pages that contain the data the program references. In this case, the reference pattern includes a repeating gap of 8032 bytes (1004×8) every 8192 bytes (1024×8). The pattern looks like this:



The grouping of consecutive bytes that the program references is called a **reference unit**. The grouping of consecutive bytes that the program skips over is

called a **gap**. Reference units and gaps alternate throughout the array at regular intervals. The reference pattern is as follows:

- The reference unit is 20 elements in size — 160 consecutive bytes that the program references.
- The gap is 1004 elements in size — 8032 consecutive bytes that the program skips over.

Figure 5-1 shows this reference pattern and the pages that the system does not bring into central storage.

What Pages Does the System Bring in When a Gap Exists?

When a gap exists, the number of pages the system brings in depends on the size of the gap, the size of the reference unit, and where the page boundary lies in relation to the gap and the reference unit. The following examples illustrate those factors.

Example 1

Figure 5-1 illustrates ARRAY1, the 1024-by-1024 array of eight-byte elements, where the program references 20 elements, then skips over the next 1004, and so forth in a forward direction throughout the array. The reference pattern includes a reference unit of 160 and a gap of 8032 bytes. The reference units begin on every other page boundary.

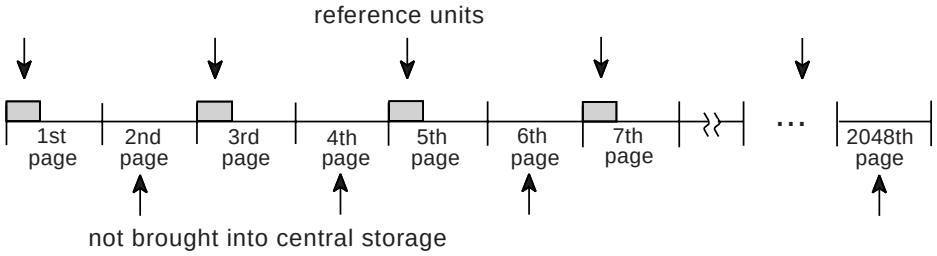
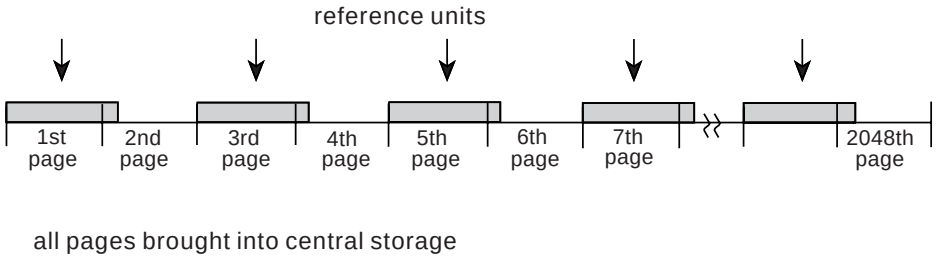


Figure 5-1. Illustration of a Reference Pattern with a Gap

Every other consecutive page of the data does not come into central storage; those pages contain only the “skipped over” data.

Example 2

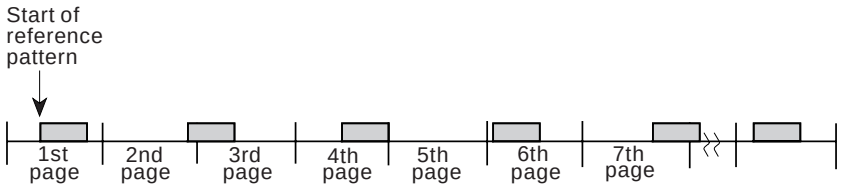
In example 2, the reference pattern includes a reference unit of 4800 bytes and a gap of 3392 bytes. The example assumes that the area to be referenced starts on a page boundary.



Because each page contains data that the program references, the system brings in all pages.

Example 3

In example 3, the area to be referenced does not begin on a page boundary. The reference pattern includes a reference unit of 2000 bytes and a gap of 5000 bytes. When you specify a reference pattern that includes a gap, the reference unit must be at the start of the area, as the following illustration shows:



most pages brought into central storage

Because the gap is larger than 4096 bytes, some pages do not come into central storage. Notice that the system does not bring in the fifth page.

Summary of how the size of the gap affects the number of pages the system brings into central storage:

- If the gap is less than 4096 bytes, the system has to bring into central all pages of the array.
- If the gap is greater than 4095 bytes and less than 8192, the system might not have to bring in certain pages. Pages that contain only data in the gap do not come in.
- If the gap is greater than 8191 bytes, the system definitely does not have to bring in certain pages that contain the gap.

Chapter 6. Using Reference Pattern Services

The two reference pattern services are CSRIRP and CSRRRP. First, you issue CALL CSRIRP to define a reference pattern for an area; then, issue CALL CSRRRP to remove the definition of reference pattern for the area. To avoid unnecessary processing, issue the calls outside of the loops that control processing of the data elements contained in the area.

Defining the Reference Pattern for a Data Area

On CSRIRP, you tell the system:

- The lowest address of the area to be referenced
- The size of the area
- The direction of reference
- The reference pattern, in terms of reference unit and gap (if one exists)
- The number of reference units the system is to bring into central storage on a page fault

The system will not process CSRIRP unless the values you specify can result in a performance gain for your program. To make sure the system processes CSRIRP, ask the system to bring in more than three pages (that is, 12288 bytes) on each page fault.

Your program can have only one pattern defined for that area at one time. If your program will later reference the same area with another reference pattern, use CSRRRP to remove the definition, and then use CSRIRP to define another pattern.

Although the system brings in pages 4096 bytes at a time, you do not have to specify values on CSRIRP or CSRRRP in increments of 4096.

Defining the Range of the Area

On CSRIRP, you define the range of the area to be referenced:

- *low_address* identifies the lowest addressed byte in the range.
- *size* identifies the size, in bytes, of the range.

When reference is forward, *low_address* identifies the first element that the program can reference in the range. When reference is backward, *low_address* identifies the last element that the program can reference in the range: reference proceeds from the high-address end in the range towards *low_address*.

The following parameters define the lowest address and the size of ARRAY1, a 1024-by-1024 array that consists of 8-byte elements. ARRAY1(1,1) identifies the element in the first row and the first column.

```
CSRIRP  with low_address of ARRAY1(1,1)
            size of 1024*1024*8 bytes
```

When a gap exists, define the range according to the following rules:

- If direction is forward, *low_address* must be the first data element in a reference unit.
- If direction is backward, the value you use for *size* must be such that the first data element the program references is the high-address end of a reference unit.

These two rules are described and illustrated in “Using CSRIRP When a Gap Exists” on page 6-3.

Identifying the Direction of the Reference

On *direction*, you specify the direction of reference through the array. Forward reference means instructions start with the element indicated by *low_address* and proceed through the range of data specified by *size*. Backward reference means the program starts processing the high-address end of the range specified by *size* and proceeds toward the *low_address* end.

- “+1” indicates forward direction.
- “−1” indicates backward direction.

An example of forward reference through ARRAY1 is specified as follows:

```
CSRIRP  with direction of +1
```

“Using CSRIRP When a Gap Exists” on page 6-3 contains examples of forward and backward references when a gap exists.

Defining the Reference Pattern

Figure 6-1 identifies two reference patterns that characterize most of the reference patterns that reference pattern services applies to.

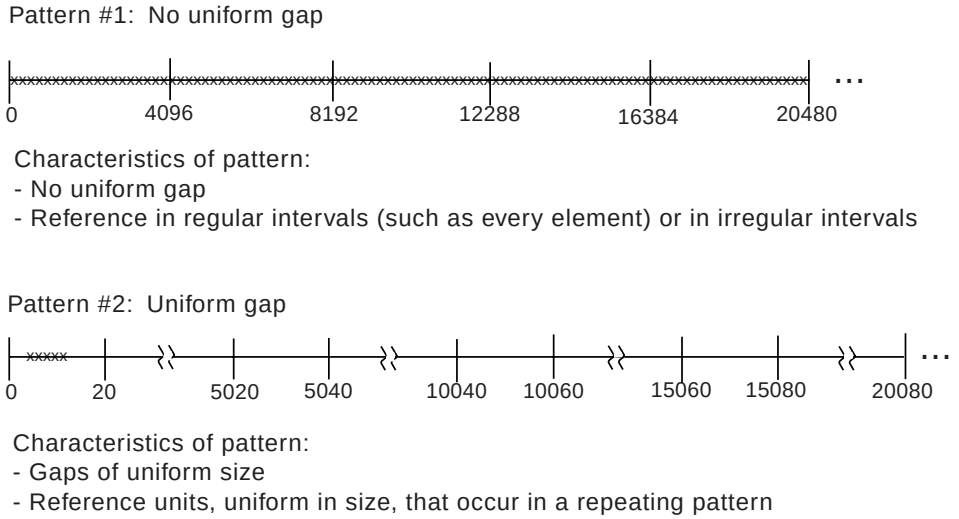


Figure 6-1. Two Typical Reference Patterns

How you define the reference pattern depends on whether your program’s reference pattern is like pattern #1 or pattern #2.

- **With pattern #1 where no uniform gap exists**, the program uses every element, every other element, or at least most elements on each page of array data. No definable gap exists. Do not use reference pattern services if the reference pattern is irregular and includes skipping over many areas larger than a page.
  - The *unitsize* parameter identifies the reference pattern; it indicates the number of bytes you want the system to use as a reference unit. Look at logical groupings of bytes, such as one row, a number of rows, or one element, if the elements are large in size. Or, you might choose to divide the area to be referenced, and bring in that area on a certain number of page faults. Use the value 0 on *gapsize*.

  - The *units* parameter tells the system how many reference units to try to bring in on a page fault. For a reference pattern that begins on a page boundary and has no gaps, the total number of bytes the system tries to bring into central storage at a time is the value on *unitsize* times the number on *units*, rounded up to the nearest multiple of 4096. See “Choosing the Number of Bytes on a Page Fault” on page 6-4 for more information on how to choose the total number of bytes.
- **With pattern #2 where a uniform gap exists**, the pattern includes alternating gaps and reference units. Specify the reference pattern carefully. If you identify a reference pattern and do not adhere to it, the system will work harder than if you had not used the service.
  - The *unitsize* and *gapsize* parameters identify the reference pattern. Pattern #2 in Figure 6-1 on page 6-2 includes a reference unit of 20 bytes and a gap of 5000 bytes. Because the gap is greater than 4095, some pages of the array might not be brought into central storage.
  - The *units* parameter tells the system how many reference units to try to bring into central storage at a time. “What Pages Does the System Bring in When a Gap Exists?” on page 5-4 can help you understand how many bytes come into central storage at one time when a gap exists.

Using CSRIRP When a Gap Exists

When a gap exists, you have to follow one of two rules in coding the two parameters, *low_address* and *size*, that define the range of data. The direction of reference determines which rule you follow:

- **When reference is forward**, *low_address* must identify the beginning of a reference unit.

  Figure 6-2 illustrates forward reference through a range of data that includes gaps. Consider the reference pattern where the program references 2000 bytes and skips the next 5000 bytes, and so forth throughout the array. The range of data starts at *low_address* and ends at the point identified in the figure by **A**. **A** can be any part of a gap or reference unit.

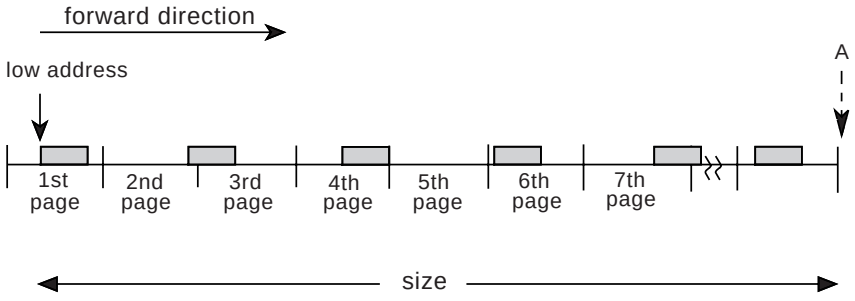


Figure 6-2. Illustration of Forward Direction of Reference

- **When reference is backward**, the value you code on *size* determines the location of the first element the program actually references. Calculate that value so that the first element the program references is the high-address end of a reference unit.

  Figure 6-3 on page 6-4 illustrates backward reference through the same array as in Figure 6-2. Again, the program references 2000 bytes and skips the next 5000 bytes, and so forth throughout the array. The range starts at *low_address* and ends at the point identified in the figure by **B**, where **B** must be the high-address end of a reference unit. *low_address* can be any part of a gap or reference unit.

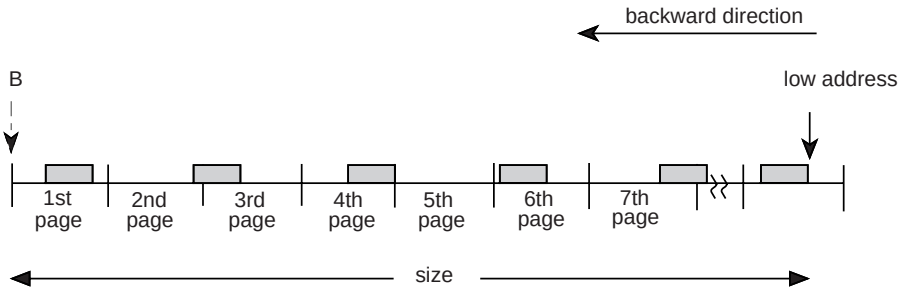


Figure 6-3. Illustration of Backward Direction of Reference

Choosing the Number of Bytes on a Page Fault

An important consideration in using reference pattern services is how many bytes to ask the system to bring in on a page fault. To determine this, you need to understand some factors that affect the performance of your program.

Pages do not stay in central storage if they are not referenced frequently enough and other programs need that central storage. The longer it takes for a program to begin referencing a page in central storage, the greater the chance that the page has been moved out before being referenced. When you tell the system how many bytes it should try and bring into central at one time, you have to consider the following:

1. **Contention for central storage**

   Your program contends for central storage along with all other submitted jobs. The greater the size of central storage, the more bytes you can ask the system to bring in on a page fault. The system responds with as much of the data you request as possible, given the availability of central storage.

2. **Contention for processor time**

   Your program contends for the processor's attention along with all other submitted jobs. The more competition, the less the processor can do for your program and the smaller the number of bytes you should request.

3. **The elapsed time of processing one page of your data**

   How long it takes a program to process a page depends on the number of references per page and the elapsed time per reference. If your program uses only a small percentage of elements on a page and references them only once or twice, the program completes the use of pages quickly. If the processing of each referenced element includes processor-intensive operations or a time-intensive operation, such as I/O, the time the program takes to process a page increases.

Conditions might vary between the peak activity of the daytime period and the low activity of the nighttime. You might be able to request a greater number at night than during the day.

What if you specify too many bytes? What if you ask the system to bring in so many pages that, by the time your program needs to use some of those pages, they have left central storage? The answer is that the system will have to bring them in again. This action causes an extra page fault and extra system overhead and decreases the benefit of reference pattern services.

For example, suppose you ask the system to bring in 204800 bytes, or 50 pages, at a time. But, by the time your program begins referencing the data on the 30th page, the system has moved that page and the ones after it out of central storage. It moved them out because the program did not use them soon enough. In this case, your program has lost the benefit of moving the last 21 pages in. Your program would get more benefit by requesting fewer than 30 pages.

What if you specify too few bytes? If you specify too small a number, the system will take more page faults than it needs to and you are not taking full advantage of reference pattern services.

For example, suppose you ask the system to bring in 40960 bytes (or 10 pages) at a time. Your program's use of each page is not time-intensive, meaning that the program finishes using the pages quickly. The program can request a number greater than 10 without causing additional page faults.

IBM recommends that you use one of the following approaches, depending on whether you want to involve your system programmer in the decision.

- The first approach is the simple one. Choose a conservative number of bytes, around 81920 (20 pages), and run the program. Look for an improvement in the elapsed time. If you like the results, you might increase the number of bytes. If you continue to increase the number, at some point you will notice a diminishing improvement or even an increase in elapsed time. Do not ask for so much that your program or other programs suffer from degraded performance.
- The second approach is for the program that needs very significant performance improvements — those programs that require amounts in excess of 50 pages. If you have such a program, you and your system programmer should examine the program's elapsed time, paging speeds, and processor execution times. In fact, the system programmer can tune the system with your program in mind, providing the needed paging resources. *z/OS MVS Initialization and Tuning Guide* can provide information on tuning the system.

  Reference pattern services affects movement of pages from auxiliary **and** expanded storage to central storage. To gain insight into the effectiveness of your reference patterns, you and your system programmer will need the kind of information that the SMF Type 30 record provides. A Type 30 record includes counts of pages moved in anticipation of your program's use of those pages. The record provides counts of pages moved between expanded and central and between auxiliary and central. It also provides elapsed time values. Use this information to calculate rates of movement in determining whether to specify a very large number of bytes — for example, amounts greater than 204800 bytes (50 pages).

Examples of Using CSRIRP to Define a Reference Pattern

To clarify the relationships between the *unitsize*, *gapsize*, and *units* parameters, this section contains three examples of defining a reference pattern. So that you can compare the three examples with what the system does without information from CSRIRP, the following call approximates the system's normal paging operation:

```
CSRIRP with unitsize of 4096 bytes
            gapsize of 0 bytes
            units of 1 reference unit (that is, one page)
```

Each time the system takes a page fault, it brings in 4096 bytes (one page), the system's reference unit. It brings in one reference unit at a time.

Example 1 The program processes all elements in an array in a forward direction. The processing of each element is fairly simple. The program runs during the peak hours, and many programs compete for processor time and central storage. A reasonable value to choose for the number of bytes to come into central on a page fault might be 80000 bytes (around 20 pages); *unitsize* can be 4000 bytes and *units* can be 20. The following CSRIRP service communicates this pattern to the system:

```
CSRIRP with unitsize of 4000 bytes
            gapsize of 0 bytes
            units of 20
            direction of +1
```

Example 2 The program performs the same process as in Example 1, except the program does not reference every element in the array. The program runs during the night hours when contention for the processor and for central storage is light. In this case, a reasonable value to choose for the number of bytes to come into central storage on a page fault might be 200000 bytes (around 50 pages). *unitsize* can again be 4000 bytes and *units* can be 50. The following CSRIRP service communicates this pattern:

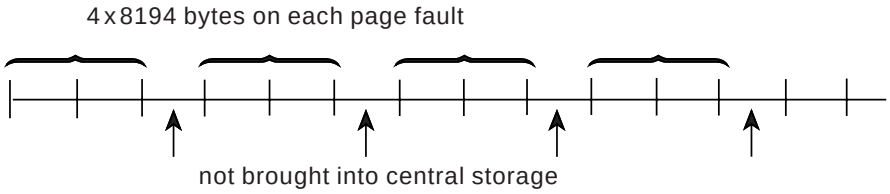
```
CSRIRP with unitsize of 4000 bytes
            gapsize of 0 bytes
            units of 50
            direction of +1
```

Example 3 The program references in a consistently forward direction through the same large array. The pattern of reference in this example includes a gap. The program references 8192 bytes, then skips the next 4096 bytes, references the next 8192 bytes, skips the next 4096 bytes throughout the array. The program chooses to bring in data 8 pages at a time. Because of the placement of reference units and gaps on page boundaries, the system does not bring in the data in the gaps.

The following CSRIRP service reflects this reference pattern:

```
CSRIRP with unitsize of 4096*2 bytes
            gapsize of 4096 bytes
            units of 4
            direction of +1
```

where the system is to bring into central storage 8 pages (4×4096×2 bytes) on a page fault. The system's response to CSRIRP is illustrated as follows:



Removing the Definition of the Reference Pattern

When a program is finished referencing the array in the way you specified on CSRIRP, use CSRRRP to remove the definition. The following example tells the system that the program in "Defining the Range of the Area" on page 6-1 has stopped referencing the array. *low_address* and *size* have the same values you coded on the CSRIRP service that defined the reference pattern for that area.

```
CSRRRP  with low_address of ARRAY1(1,1)
            size of 1024*1024*8 bytes
```

Handling Return Codes

Each time you call CSRIRP or CSRRRP, your program receives a return code and a reason code. These codes indicate whether the service completed successfully or whether the system rejected the service.

When you receive a return code that indicates a problem or an unusual condition, try to correct the problem, and rerun the program. Return codes and reason codes are described in Chapter 7, “Reference Pattern Services” with the description of each reference pattern service.

Chapter 7. Reference Pattern Services

To use reference pattern services, you issue CALLs that invoke the appropriate reference pattern services program. Each service program performs one or more functions and requires a set of parameters coded in a specific order on the CALL statement.

This chapter describes the CALL statements that invoke reference pattern services. Each description includes a syntax diagram, parameter descriptions, and return code and reason code explanations with recommended actions. For examples of how to code the CALL statements, see Chapter 8, "Reference Pattern Services Coding Examples" on page 8-1.

This chapter contains the following topics:

CSRIRP — Define a Reference Pattern

Call CSRIRP to define a reference pattern for a large data area, such as an array, that you are about to reference. Through CSRIRP, you identify the data area and describe the reference pattern. Additionally, you tell the system how many bytes of data you want it to bring into central storage on a page fault (that is, each time the program references data that is not in central storage). This action might significantly improve the performance of the program.

Two parameters define the reference pattern:

- *unitsize* refers to a **reference unit** — a grouping of consecutive bytes that the program references.
- *gapsize* refers to a **gap** — a grouping of consecutive bytes that the program repeatedly skips over; when a pattern has a gap, reference units and gaps alternate throughout the data area.

Reference units and gaps must each be uniform in size and appear throughout the data area at repeating intervals.

Another parameter, *units*, allows you to specify how many reference units you want the system to bring into central storage each time the program references data that is not in central storage.

When you end the reference pattern in that data area, call the CSRRRP service.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown below. For parameters that CSRIRP uses to obtain input values, assign appropriate values.

On entry to CSRIRP, register 1 points to the reference pattern service parameter list. Note that when a FORTRAN program calls CSRIRP, and it is running in access register (AR) mode, register 1 does not point to the reference pattern service parameter list; it points to a list of parameter addresses. Each address in this list points to the data in the corresponding parameter of the reference pattern service parameter list. To use reference pattern services in this environment, the caller must provide an assembler interface routine to convert the FORTRAN parameter list to the form expected by reference services.

Assign values, acceptable to CSRIRP, to *low_address*, *size*, *direction*, *unitsize*, *gapsize*, and *units*. CSRIRP returns values in *return_code* and *reason_code*.

CALL CSRIRP	(low_address ,size ,direction ,unitsize ,gapsize ,units ,return_code ,reason_code)

The parameters are explained as follows:

low_address
Specifies the beginning point of the data to be referenced.

low_address is the name of the data that resides at the beginning of the data area. When the direction is forward and a gap exists, *low_address* must identify the beginning of a reference unit.

,size
Identifies the size, in bytes, of the data area to be accessed. When direction is backward and a gap exists, the value of *size* must be such that the first data element the program references is the high-address end of a reference unit.

Define *size* as integer data of length 4.

,direction
Indicates the direction of reference, either “+1” for forward or “−1” for backward.

Define *direction* as integer data of length 4.

,unitsize
Specifies the size of a reference unit.

If the pattern does not have a gap, define the reference unit as a logical grouping according to the structure of the data array. Examples are: one row, a number of rows, one element, or one page (4096 bytes). If the pattern has a gap, define *unitsize* as the grouping of bytes that the program references and *gap* as the grouping of bytes that the program skips over.

Define *unitsize* as integer data of length 4.

,gapsize
Specifies the size, in bytes, of a gap. If the pattern has a gap, define the gap as the grouping of bytes that the program skips over. If the pattern does not have a gap, use the value “0”.

Define *gapsize* as integer data of length 4.

,units
Indicates how many reference units the system is to bring into central storage each time the program needs data that is not in central storage.

Define *units* as integer data of length 4.

,return_code
When CSRIRP completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

,reason_code
When CSRIRP completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return Codes and Reason Codes

When CSRIRP returns control to your program, *return_code* contains a return and *reason_code* contains a reason code. The following table identifies return code and reason code combinations and tells what each means.

Return and reason codes, in hexadecimal, from CSRIRP are:

Return Code	Reason Code	Meaning
00	None	CSRIRP completed successfully.
04	xx0001xx	CSRIRP completed successfully; however, the system did not accept the reference pattern the caller specified. The system decided that bringing in pages of 4096 bytes would be more efficient.
08	xx0002xx	Unsuccessful completion. The range that the caller specified overlaps the range that a previous request specified.
08	xx0003xx	Unsuccessful completion. The number of CSRIRP requests for the user exceeds 100, the maximum number the system allows.
08	xx0004xx	Unsuccessful completion. Storage is not available for the CSRIRP service.
08	00000004	Unsuccessful completion. The direction that the caller specified is not valid.

CSRRRP — Remove a Reference Pattern

Call CSRRRP to remove the reference pattern for a data area, as specified by the CSRIRP service. On CSRRRP, you identify the beginning of the data area and its size. Code *low_address* and *size* exactly as you coded them on the CSRIRP service that defined the reference pattern.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown below. For parameters that CSRRRP uses to obtain input values, assign values that are acceptable to CSRRRP.

Assign values to CSRRRP, to *low_address* and *size*. CSRRRP returns values in *return_code* and *reason_code*.

CALL CSRRRP	(low_address ,size ,return_code ,reason_code)

The parameters are explained as follows:

low_address
Specifies the beginning point of the data to be referenced.

low_address is the name of the data that resides at the beginning of the data area.

,size
Specifies the size, in bytes, of the data area.

Define *size* as integer data of length 4.

,return_code
When CSRRRP completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

,reason_code
When CSRRRP completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return Codes and Reason Codes

When CSRRRP returns control to your program, *return_code* contains a hexadecimal return code and *reason_code* contains a hexadecimal reason code. The following table identifies return code and reason code combinations and tells what each means.

Return Code	Reason Code	Meaning
00	None	CSRRRP completed successfully.
08	xx0101xx	Unsuccessful completion. No CSRIRP service request was in effect for the specified data area. Check to see if the system rejected the previous CSRIRP request for the data area.

Chapter 8. Reference Pattern Services Coding Examples

The following examples show how to invoke reference pattern services from each of the supported languages. Following each program example is an example of the JCL needed to compile, link edit, and execute the program example. Use these examples to supplement and reinforce information that is presented elsewhere in this book.

Note: Included in the FORTRAN example is the code for a required assembler language program. This program ensures that the reference pattern for the FORTRAN program is aligned on a 4K boundary.

The programs in this chapter are similar. They each process two arrays, A and B. The arrays are 200×200 in size, each element consisting of 4 bytes. Processing is as follows:

- Declare the arrays.
- Define reference patterns for A and B.
- Initialize A and B.
- Remove the definitions of the reference patterns for A and B.
- Define new reference patterns for A and B.
- Multiply A and B, generating array C.
- Remove the definitions of the reference patterns for A and B.

The examples are presented on the following pages:

C/370 Example

The following example is coded in C/370:

```
#include <stdio.h>
#include <stdlib.h>
#include "csrbpc"

#define m 200
#define n 200
#define p 200
#define kelement_size 4
int chk_code(long int ret, long int reason, int linenumber);

main()
{
  long int A[m] [n];
  long int B[m] [n];
  long int C[m] [n];
  long int i;
  long int j;
  long int k;
  long int rc;
  long int rsn;
  long int arraysize;
  long int direction;
  long int unitsize;
  long int gap;
```

C/370 Example

```
long int units;

arraysize = m*n*kelement_size;
direction = csr_forward;
unitsize = kelement_size*n;
gap = 0;
units = 20;


csrirp(A, &amp;arraysize, &direction,;
       &unitsize,;
       &gap,;
       &units,;
       &rc,;
       &rsn);
chk_code(rc,rsn,__LINE__);

arraysize = m*p*kelement_size;

csrirp(B, &amp;arraysize, &direction,;
       &unitsize,;
       &gap,;
       &units,;
       &rc,;
       &rsn);
chk_code(rc,rsn,__LINE__);
for (i=0; i<m; i++) {
  for (j=0; j<n; j++) {
    A[i][j] = i + j;
  }
}
for (i=0; i<n; i++) {
  for (j=0; j<p; j++) {
    B[i][j] = i + j;
  }
}

arraysize = m*n*kelement_size;

csrrrp(A, &arraysize,;
       &rc,;
       &rsn);
chk_code(rc,rsn,__LINE__);

arraysize = m*p*kelement_size;
csrrrp(B, &arraysize,;
       &rc,;
       &rsn);
chk_code(rc,rsn,__LINE__);

arraysize = m*n*kelement_size;
units = 25;
csrirp(A, &arraysize, &direction,;
       &unitsize,;
       &gap,;
       &units,;
       &rc,;
       &rsn);
chk_code(rc,rsn,__LINE__);

arraysize = n*p*kelement_size;
gap = (p-1)*kelement_size;
units = 50;
csrirp(B, &arraysize, &direction,;
       &unitsize,;
       &gap,;
       &units,;
```

```
        &rc,;
        &rsn);
  chk_code(rc,rsn,__LINE__);
  for (i=0; i<m; i++) {
    for (j=0; j<p; j++) {
      C[i][j] = 0;
      for (k=0; k<n; k++) {
        C[i][j] = C[i][j] + A[i][k] * B[k][j];
      }
    }
  }
  arraysize = m*n*kelement_size;
  csrrrp(A, &arraysize,;
        &rc,;
        &rsn);
  chk_code(rc,rsn,__LINE__);
  arraysize = n*p*kelement_size;
  csrrrp(B, &arraysize,;
        &rc,;
        &rsn);
  chk_code(rc,rsn,__LINE__);
}

/* chk_code will check return code and reason code from previous */
/* calls to HLL services. It will print a message if any of the  */

int chk_code(long int ret, long int reason, int linenumber)
{
    if (ret != 0)
        printf("return_code = %ld instead of 0 at line %d\n",
                ret, linenumber);
    if (reason != 0)
        printf("reason_code = %ld instead of 0 at line %d\n",
                reason, linenumber);
}
//*-------------------------------------------------------------------
//* JCL USED TO COMPILE, LINK, THE C/370 PROGRAM
//*-------------------------------------------------------------------
//CJOB JOB
//CCSTEP EXEC EDCCO,
//  CPARM='LIST,XREF,OPTIMIZE,RENT,SOURCE',
//  INFILE='REFPAT.SAMPLE.PROG(C),DISP=SHR'
//COMPILE.SYSLIN DD DSN='TEST.MPS.OBJ(C),DISP=SHR'
//COMPILE.USERLIB DD  DSN=REFPAT.DECLARE.SET,DISP=SHR
//LKSTEP EXEC EDCPLO,
//    LPARM='AMOD=31,LIST,REFR,RENT,RMOD=ANY,XREF'                      00022007
//PLKED.SYSIN DD DSN='TEST.MPS.OBJ(C),DISP=SHR'
//LKED.SYSLMOD DD DSN=REFPAT.USER.LOAD,DISP=SHR,
//     UNIT=3380,VOL=SER=RSMPAK
//LKED.SYSIN DD *
  LIBRARY IN(CSRIRP,CSRRRP)
  NAME BPGC(R)
//LKED.IN  DD DSN=SYS1.CSSLIB,DISP=SHR
//*-------------------------------------------------------------------
//* JCL USED TO EXECUTE THE C/370 PROGRAM
//*-------------------------------------------------------------------
//CGO  JOB  TIME=1440,MSGLEVEL=(1,1),MSGCLASS=A
//RUN        EXEC  PGM=BPGC,TIME=1440                                00110804
//STEPLIB    DD    DSN=REFPAT.USER.LOAD,DISP=SHR,                    00111002
//           UNIT=3380,VOL=SER=VM2TSO                                00111101
//           DD    DSN=CEE.SCEERUN,DISP=SHR                           0111002
//SYSPRINT   DD SYSOUT=*
//PLIDUMP    DD SYSOUT=*
//SYSUDUMP   DD SYSOUT=*
```

COBOL Example

```
//*-------------------------------------------------------------------
//* THE FOLLOWING EXAMPLE IS CODED IN COBOL:
//*-------------------------------------------------------------------

       IDENTIFICATION DIVISION.
      ******************************************************************
      * MULTIPLY ARRAY A TIMES ARRAY B GIVING ARRAY C                  *
      * USE THE REFERENCE PATTERN CALLABLE SERVICES TO IMPROVE THE     *
      * PERFORMANCE.                                                   *
      ******************************************************************

       PROGRAM-ID. TESTCOB.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

      * COPY THE INCLUDE FILE (WHICH DEFINES CSRFORWARD, CSRBACKWARD)
       COPY CSRBPCOB.

      * DIMENSIONS OF ARRAYS - A IS M BY N, B IS N BY P, C IS M BY P
       1   M PIC 9(9) COMP VALUE 200.
       1   N PIC 9(9) COMP VALUE 200.
       1   P PIC 9(9) COMP VALUE 200.

      * ARRAY DECLARATIONS FOR ARRAY A - M = 200, N = 200
       1   A1.
        2   A2 OCCURS 200 TIMES.
         3   A3 OCCURS 200 TIMES.
          4   ARRAY-A PIC S9(8).

      * ARRAY DECLARATIONS FOR ARRAY B - N = 200, P = 200
       1   B1.
        2   B2 OCCURS 200 TIMES.
         3   B3 OCCURS 200 TIMES.
          4   ARRAY-B PIC S9(8).

      * ARRAY DECLARATIONS FOR ARRAY C - M = 200, P = 200
       1   C1.
        2   C2 OCCURS 200 TIMES.
         3   C3 OCCURS 200 TIMES.
          4   ARRAY-C PIC S9(8).

       1   I PIC 9(9) COMP.
       1   J PIC 9(9) COMP.
       1   K PIC 9(9) COMP.
       1   X PIC 9(9) COMP.
       1   ARRAY-A-SIZE PIC 9(9) COMP.
       1   ARRAY-B-SIZE PIC 9(9) COMP.
       1   UNITSIZE PIC 9(9) COMP.
       1   GAP PIC 9(9) COMP.
       1   UNITS PIC 9(9) COMP.
       1   RETCODE PIC 9(9) COMP.
       1   RSNCODE PIC 9(9) COMP.
       PROCEDURE DIVISION.
           DISPLAY " BPAGE PROGRAM START "

      * CALCULATE CSRIRP PARAMETERS FOR INITIALIZING ARRAY A
      * UNITSIZE WILL BE THE SIZE OF ONE ROW.
      * UNITS WILL BE 25
      * SO WE'RE ASKING FOR 25 ROWS TO COME IN AT A TIME
           COMPUTE ARRAY-A-SIZE = M * N * 4
           COMPUTE UNITSIZE = N * 4
           COMPUTE GAP = 0
           COMPUTE UNITS = 25
```

```
     CALL "CSRIRP" USING
         ARRAY-A(1, 1),
         ARRAY-A-SIZE,
         CSRFORWARD,
         UNITSIZE,
         GAP,
         UNITS,
         RETCODE,
         RSNCODE

     DISPLAY "FIRST RETURN CODE IS "
     DISPLAY RETCODE

* CALCULATE CSRIRP PARAMETERS FOR INITIALIZING ARRAY B
* UNITSIZE WILL BE THE SIZE OF ONE ROW.
* UNITS WILL BE 25
* SO WE'RE ASKING FOR 25 ROWS TO COME IN AT A TIME

     COMPUTE ARRAY-B-SIZE = N * P * 4
     COMPUTE UNITSIZE = P * 4
     COMPUTE GAP = 0
     COMPUTE UNITS = 25
     CALL "CSRIRP" USING

         ARRAY-B(1, 1),
         ARRAY-B-SIZE,
         CSRFORWARD,
         UNITSIZE,
         GAP,
         UNITS,
         RETCODE,
         RSNCODE

     DISPLAY "SECOND RETURN CODE IS "
     DISPLAY RETCODE

* INITIALIZE EACH ARRAY A ELEMENT TO THE SUM OF ITS INDICES
     PERFORM VARYING I FROM 1 BY 1 UNTIL I = M
       PERFORM VARYING J FROM 1 BY 1 UNTIL J = N
         COMPUTE X = I + J
         MOVE X TO ARRAY-A(I, J)
         END-PERFORM
       END-PERFORM

* INITIALIZE EACH ARRAY B ELEMENT TO THE SUM OF ITS INDICES
     PERFORM VARYING I FROM 1 BY 1 UNTIL I = N
       PERFORM VARYING J FROM 1 BY 1 UNTIL J = P
         COMPUTE X = I + J
         MOVE X TO ARRAY-B(I, J)
       END-PERFORM
     END-PERFORM

* REMOVE THE REFERENCE PATTERN ESTABLISHED FOR ARRAY A
     CALL "CSRRRP" USING
         ARRAY-A(1, 1),
         ARRAY-A-SIZE,
         RETCODE,
         RSNCODE

     DISPLAY "THIRD RETURN CODE IS "
     DISPLAY RETCODE

* REMOVE THE REFERENCE PATTERN ESTABLISHED FOR ARRAY B
     CALL "CSRRRP" USING
         ARRAY-B(1, 1),
         ARRAY-B-SIZE,
         RETCODE,
```

COBOL Example

```
            RSNCODE

        DISPLAY "FOURTH RETURN CODE IS "
        DISPLAY RETCODE

      * CALCULATE CSRIRP PARAMETERS FOR ARRAY A
      * UNITSIZE WILL BE THE SIZE OF ONE ROW.
      * UNITS WILL BE 20
      * SO WE'RE ASKING FOR 20 ROWS TO COME IN AT A TIME
        COMPUTE ARRAY-A-SIZE = M * N * 4
        COMPUTE UNITSIZE = N * 4
        COMPUTE GAP = 0
        COMPUTE UNITS = 20

        CALL "CSRIRP" USING
            ARRAY-A(1, 1),
            ARRAY-A-SIZE,
            CSRFORWARD,
            UNITSIZE,
            GAP,
            UNITS,
            RETCODE,
            RSNCODE

        DISPLAY "FIFTH RETURN CODE IS "
        DISPLAY RETCODE

      * CALCULATE CSRIRP PARAMETERS FOR ARRAY B
      * UNITSIZE WILL BE THE SIZE OF ONE ELEMENT.
      * GAP WILL BE (N-1)*4 (IE. THE REST OF THE ROW).
      * UNITS WILL BE 50
      * SO WE'RE ASKING FOR 50 ELEMENTS OF A COLUMN TO COME IN
      * AT ONE TIME
        COMPUTE ARRAY-B-SIZE = N * P * 4
        COMPUTE UNITSIZE = 4
        COMPUTE GAP = (N - 1) * 4
        COMPUTE UNITS = 50

        CALL "CSRIRP" USING
            ARRAY-B(1, 1),
            ARRAY-B-SIZE,
            CSRFORWARD,
            UNITSIZE,
            GAP,
            UNITS,
            RETCODE,
            RSNCODE

        DISPLAY "SIXTH RETURN CODE IS "
        DISPLAY RETCODE

      * MULTIPLY ARRAY A TIMES ARRAY B GIVING ARRAY C
        PERFORM VARYING I FROM 1 BY 1 UNTIL I = M
          PERFORM VARYING J FROM 1 BY 1 UNTIL J = P
            COMPUTE ARRAY-C(I, J) = 0
            PERFORM VARYING K FROM 1 BY 1 UNTIL K = N
            COMPUTE X = ARRAY-C(I, J) +
                    ARRAY-A(I, K) * ARRAY-B(K, J)
            END-PERFORM
          END-PERFORM
        END-PERFORM

      * REMOVE THE REFERENCE PATTERN ESTABLISHED FOR ARRAY A
        CALL "CSRRRP" USING
            ARRAY-A(1, 1),
            ARRAY-A-SIZE,
            RETCODE,
```

```
                RSNCODE

            DISPLAY "SEVENTH RETURN CODE IS "
            DISPLAY RETCODE

      * REMOVE THE REFERENCE PATTERN ESTABLISHED FOR ARRAY B
            CALL "CSRRRP" USING
                ARRAY-B(1, 1),
                ARRAY-B-SIZE,
                RETCODE,
                RSNCODE

            DISPLAY "EIGHTH RETURN CODE IS "
            DISPLAY RETCODE

            DISPLAY " BPAGE PROGRAM END "
            GOBACK.
//*-------------------------------------------------------------------
//* JCL USED TO COMPILE, LINK, THE COBOL PROGRAM
//*-------------------------------------------------------------------
//FCHANGC JOB 'D3113P,D31,?','FCHANG6-6756',CLASS=T,
//     MSGCLASS=H,NOTIFY=FCHANG,REGION=0K
//CCSTEP EXEC EDCCO,
//  CPARM='LIST,XREF,OPTIMIZE,RENT,SOURCE',
//  INFILE='FCHANG.PUB.TEST(C)'
//COMPILE.SYSLIN DD DSN='FCHANG.MPS.OBJ(C),DISP=SHR'
//COMPILE.USERLIB DD  DSN='FCHANG.DECLARE.SET,DISP=SHR
//LKSTEP EXEC EDCPLO,
//    LPARM='AMOD=31,LIST,REFR,RENT,RMOD=ANY,XREF'                    00022007
//PLKED.SYSIN DD DSN='FCHANG.MPS.OBJ(C),DISP=SHR'
//LKED.SYSLMOD DD DSN=RSMID.FBB4417.LINKLIB,DISP=SHR,
//     UNIT=3380,VOL=SER=RSMPAK
//LKED.SYSIN DD *
  LIBRARY IN(CSRIRP,CSRRRP)
  NAME BPGC(R)
//LKED.IN  DD DSN=FCHANG.MPS.OBJ,DISP=SHR
//*-------------------------------------------------------------------
//* LINK PROGRAM
//*-------------------------------------------------------------------
//COBOLLK JOB                                                       00010002
//LINKEDIT EXEC PGM=IEWL,                                           00040000
// PARM='MAP,XREF,LIST,LET,AC=1,SIZE=(1000K,100K)'                  00050000
//SYSLIN   DD DDNAME=SYSIN                                          00051000
//SYSLMOD  DD DSN=REFPAT.USER.LOAD,DISP=OLD                         00052002
//SYSLIB   DD DSN=CEE.SCEELKED,DISP=SHR                             00053000
//MYLIB    DD DSN=REFPAT.COBOL.OBJ,DISP=SHR                         00053102
//CSRLIB   DD DSN=SYS1.CSSLIB,DISP=SHR                              00053202
//SYSPRINT DD SYSOUT=H                                              00053300
//*                                                                 00053400
//SYSUT1   DD UNIT=SYSDA,SPACE=(TRK,(20,10))                        00053500
//SYSUT2   DD UNIT=SYSDA,SPACE=(TRK,(20,10))                        00053600
//SYSIN    DD *                                                     00053700
  INCLUDE MYLIB(COBOL)                                              00053802
  LIBRARY CSRLIB(CSRIRP,CSRRRP)                                     00053901
  NAME COBLOAD(R)                                                   00054002
/*                                                                  00055000
//*-------------------------------------------------------------------
//* JCL USED TO EXECUTE THE COBOL PROGRAM
//*-------------------------------------------------------------------
//COB2  JOB  MSGLEVEL=(1,1),TIME=1440                               00010000
//GO     EXEC  PGM=COBLOAD                                          00020001
//STEPLIB  DD  DSNAME=CEE.SCEERUN,DISP=SHR                          00030001
//         DD  DSN=REFPAT.USER.LOAD,DISP=SHR,VOL=SER=RSMPAK,        00040001
//     UNIT=3380                                                    00041001
//SYSABOUT DD  SYSOUT=*                                             00050000
```

COBOL Example

```
//SYSOUT   DD  SYSOUT=A                                                  00051001
//SYSDBOUT DD  SYSOUT=*                                                  00060000
//SYSUDUMP DD  SYSOUT=*                                                  00070000
```

FORTRAN Example

```
*********************************************************************
*                                                                   *
*                                                                   *
*      This is FORTRAN.  Followed by an assembler routine           *
*      called ADDR that has to be linkedited with the object        *
*      code from this testcase, and the CSR stubs.                  *
*                                                                   *
*********************************************************************
@PROCESS DC(BPAGEFOR)
      PROGRAM BPAGEFOR
C
      INCLUDE 'SYS1.SAMPLIB(CSRBPFOR)'
C
C     Multiply two arrays together - testing CSRIRP, CSRRRP services
C
C
      INTEGER M /200/
      INTEGER N /200/
      INTEGER P /200/
      PARAMETER (NKELEMENT_SIZE=4)
      INTEGER RC,RSN
      COMMON /WINCOM/A(200,200)
      COMMON /WINCOM/B(200,200)
      COMMON /WINCOM/C(200,200)
C
C     Initialize the arrays
C
      CALL CSRIRP(A(1,1),
     *            M*N*NKELEMENT_SIZE,
     *            CSR_FORWARD,
     *            M*NKELEMENT_SIZE,
     *            0,
     *            20,
     *            RC,
     *            RSN)
      CALL CSRIRP(B(1,1),
     *            N*P*NKELEMENT_SIZE,
     *            CSR_FORWARD,
     *            N*NKELEMENT_SIZE,
     *            0,
     *            20,
     *            RC,
     *            RSN)
      DO 102 J = 1, N
      DO 100 I = 1, M
        A(I,J) = I + J
  100 CONTINUE
  102 CONTINUE
      DO 106 J = 1, P
      DO 104 I = 1, N
        B(I,J) = I + J
  104 CONTINUE
  106 CONTINUE
C
      CALL CSRRRP(A(1,1),
     *            M*N*NKELEMENT_SIZE,
     *            RC,
     *            RSN)
      CALL CSRRRP(B(1,1),
     *            N*P*NKELEMENT_SIZE,
     *            RC,
```

```
     *            RSN)
C
C Multiply the two arrays together
C
      CALL CSRIRP (A(1,1),
     *             M*N*NKELEMENT_SIZE,
     *             CSR_FORWARD,
     *             N*NKELEMENT_SIZE,
     *             (N-1)*KELEMENT_SIZE,
     *             50,
     *             RC,
     *             RSN)
      CALL CSRIRP (B(1,1),
     *             N*P*NKELEMENT_SIZE,
     *             CSR_FORWARD,
     *             NKELEMENT_SIZE*N,
     *             0,
     *             20,
     *             RC,
     *             RSN)
      DO 112 I = 1, M
      DO 110 J = 1, N
      DO 108 K = 1, P
        C(I,J) = C(I,J) + A(I,K) * B(K,J)
  108 CONTINUE
  110 CONTINUE
  112 CONTINUE
      CALL CSRRRP (A(1,1),
     *             M*N*NKELEMENT_SIZE,
     *             RC,
     *             RSN)
      CALL CSRRRP (B(1,1),
     *             N*P*NKELEMENT_SIZE,
     *             RC,
     *             RSN)

      STOP
      END
//*********************************************************************    00010000
//*                                                                   *    00020000
//*      THIS IS THE JCL THAT COMPILES THE PROGRAM.                   *    00030000
//*                                                                   *    00020000
//*********************************************************************    00080000
//FORTJOB JOB                                                              00090007
//  MSGCLASS=H,RDR=R,                                                      00110007
//  MSGLEVEL=(1,1),CLASS=T                                                 00120000
//*                                                                        00130000
//*                                                                        00140000
//*  COMPILE AND LINKEDIT FOR FORTRAN                                      00150000
//*                                                                        00160000
//*                                                                        00170000
//*                                                                        00180000
//VSF2CL PROC  FVPGM=FORTVS2,FVREGN=2100K,FVPDECK=NODECK,                  00190000
//         FVPOLST=NOLIST,FVPOPT=0,FVTERM='SYSOUT=A',                      00200000
//         PGMNAME=MAIN,PGMLIB='&&GOSET',FVLNSPC='3200,(25,6)'             00210000
//*                                                                        00220000
//* COPYRIGHT:  5668-806                                                   00230000
//*             (C) COPYRIGHT IBM CORP 1985, 1988                          00240000
//*             LICENSED MATERIALS - PROPERTY OF IBM                       00250000
//*             REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083      00260000
//*                                                                        00270000
//* STATUS:     02.03.00 (VV.RR.MM)                                        00280000
//*                                                                        00290000
//*            PARAMETER  DEFAULT-VALUE     USAGE                         00300000
//*                                                                        00310000
//*              FVPGM    FORTVS2           COMPILER NAME                 00320000
//*              FVREGN   2100K             FORT-STEP REGION              00330000
```

FORTRAN Example

```
//*              FVPDECK  NODECK            COMPILER DECK OPTION        00340000
//*              FVPOLST  NOLIST            COMPILER LIST OPTION        00350000
//*              FVPOPT   0                 COMPILER OPTIMIZATION       00360000
//*              FVTERM   SYSOUT=A          FORT.SYSTERM OPERAND        00370000
//*              FVLNSPC  3200,(25,6)       FORT.SYSLIN SPACE           00380000
//*              PGMLIB   &&GOSET           LKED.SYSLMOD DSNAME         00390000
//*              PGMNAME  MAIN              LKED.SYSLMOD MEMBER NAME    00400000
//*                                                                     00410000
//FORT   EXEC  PGM=&FVPGM,REGION=&FVREGN,COND=(4,LT),                   00420000
//             PARM='&FVPDECK,&FVPOLST,OPT(&FVPOPT)'                    00430000
//STEPLIB      DD DSN=D24PP.FORT230.VSF2COMP,DISP=SHR                   00440000
//SYSPRINT     DD SYSOUT=A,DCB=BLKSIZE=3429                             00450000
//SYSTERM      DD &FVTERM                                               00460000
//SYSPUNCH     DD SYSOUT=B,DCB=BLKSIZE=3440                             00470000
//SYSLIN       DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,             00480000
//             SPACE=(&FVLNSPC),DCB=BLKSIZE=3200                        00490000
//LKED   EXEC  PGM=HEWL,REGION=768K,COND=(4,LT),                        00500000
//             PARM='LET,LIST,XREF'                                     00510000
//SYSPRINT     DD SYSOUT=A                                              00520000
//SYSLIB       DD DSN=CEE.SCEELKED,DISP=SHR                             00530000
//SYSUT1       DD UNIT=SYSDA,SPACE=(1024,(200,20))                      00540000
//SYSLMOD      DD DSN=&PGMLIB.(&PGMNAME),DISP=(,PASS),UNIT=SYSDA,       00550000
//             SPACE=(TRK,(10,10,1),RLSE)                               00560000
//SYSLIN       DD DSN=&&LOADSET,DISP=(OLD,DELETE)                       00570000
//             DD DDNAME=SYSIN                                          00580000
// PEND                                                                00590000
//       EXEC  VSF2CL,FVTERM='SYSOUT=H',                               00600000
//         PGMNAME=FORTRAN,PGMLIB='REFPAT.USER.LOAD'                   00680008
//FORT.SYSIN DD DSN=REFPAT.SAMPLE.PROG(FORTRAN),DISP=SHR               00690008
//LKED.SYSLIB       DD DSN=CEE.SCEELKED,DISP=SHR                       00700000
//LKED.SYSLMOD DD DSN=REFPAT.USER.LOAD,DISP=SHR                        00710007
//LKED.SYSIN DD *                                                     00720000
  INCLUDE  IN(CSRIRP,CSRRRP,ADDR)                                     00730000
  NAME BPGFORT(R)                                                     00740006
/*                                                                    00750000
//*     THE CSR STUBS ARE AVAILABLE IN SYS1.CSSLIB,                   00760007
//*     THE OBJ FOR THE ADDR ROUTINE IS IN TEST.OBJ                   00770007
//*                                                                   00780000
//LKED.IN     DD DSN=SYS1.CSSLIB,DISP=SHR                             00790007
//            DD DSN=REFPAT.TEST.OBJ,DISP=SHR                         00mm0007
*********************************************************************   00010000
*                                                                   *   00020000
*      THIS IS THE JCL I USE TO EXECUTE THE PROGRAM.                *   00030000
*                                                                   *   00060000
*********************************************************************   00070000
//FONO1 JOB  MSGLEVEL=(1,1),TIME=1440                                   00080003
//VSF2G  PROC  GOPGM=MAIN,GOREGN=100K,                                  00090000
//*                                                                   00100000
//*                                                                   00110000
//*    EXECUTE A FORTRAN TESTCASE - CHANGE ALL CRTFONXX TO CRTFONZZ   00120000
//*                                                                   00130000
//             GOF5DD='DDNAME=SYSIN',                                 00140000
//             GOF6DD='SYSOUT=A',                                     00150000
//             GOF7DD='SYSOUT=B'                                      00160000
//*                                                                   00170000
//* COPYRIGHT:  5668-806                                              00180000
//*             (C) COPYRIGHT IBM CORP 1985, 1988                     00190000
//*             LICENSED MATERIALS - PROPERTY OF IBM                  00200000
//*             REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083  00210000
//*                                                                   00220000
//* STATUS:     02.03.00 (VV.RR.MM)                                   00230000
//*                                                                   00240000
//*            PARAMETER  DEFAULT-VALUE     USAGE                     00250000
//*                                                                   00260000
//*              GOPGM    MAIN              PROGRAM NAME              00270000
//*              GOREGN   100K              GO-STEP REGION            00280000
//*              GOF5DD   DDNAME=SYSIN      GO.FT05F001 DD OPERAND    00290000
```

```
//*              GOF6DD   SYSOUT=A          GO.FT06F001 DD OPERAND      00300000
//*              GOF7DD   SYSOUT=B          GO.FT07F001 DD OPERAND      00310000
//*                                                                      00320000
//*                                                                      00330000
//GO     EXEC  PGM=&GOPGM,REGION=&GOREGN,COND=(4,LT)                     00340000
//STEPLIB      DD DSN=CEE.SCEERUN,DISP=SHR                             00350004
//FT05F001     DD &GOF5DD                                              00360000
//FT06F001     DD &GOF6DD                                              00370000
//FT07F001     DD &GOF7DD                                              00380000
// PEND                                                                00390000
//GO EXEC VSF2G,GOPGM=BPGFORT,GOREGN=999K                              00400004
//GO.STEPLIB      DD DSN=WINDOW.D24PP.FORTLIB,DISP=SHR,                00410004
//                   VOL=SER=VM2TSO,UNIT=3380                          00410104
//                DD DSN=WINDOW.R40.VSF2LOAD,DISP=SHR,                 00411004
//                   VOL=SER=VM2TSO,UNIT=3380                          00412004
//                DD DSN=REFPAT.USER.LOAD,DISP=SHR,                    00420003
//                   VOL=SER=VM2TSO,UNIT=3380                          00430004
```

Pascal Example

```
*********************************************************************
*                                                                   *
*     PASCAL example. The data object is permanent and already      *
*     allocated.  A scroll area is used.                            *
*                                                                   *
*********************************************************************
 program BPAGEPAS;

   %include CSRBPPAS

   CONST
     m               = 250;
     n               = 250;
     p               = 250;
     kelement_size   = 4;
     a_size          = m*n*kelement_size;
     b_size          = n*p*kelement_size;
     c_size          = m*p*kelement_size;

   VAR
     a               : array (.1..m, 1..n.) of integer;
     b               : array (.1..n, 1..p.) of integer;
     c               : array (.1..m, 1..p.) of integer;
     i               : integer;
     j               : integer;
     k               : integer;
     rc              : integer;
     rsn             : integer;

   BEGIN
          csrirp (a(.1,1.), a_size, csr_forward,
                  kelement_size*m,
                  0,
                  50,
                  rc,
                  rsn);
          csrirp (b(.1,1.), b_size, csr_forward,
                  kelement_size*n,
                  0,
                  20,
                  rc,
                  rsn);
     for i:=1 to m do
       for j:=1 to n do
         a(.i,j.) := i + j;
     for i:=1 to n do
       for j:=1 to p do
```

Pascal Example

```
          b(.i,j.) := i + j;
           csrrrp (a(.1,1.), a_size,
                   rc,
                   rsn);
           csrrrp (b(.1,1.), b_size,
                   rc,
                   rsn);
      /* Multiply the two arrays together */

           csrirp (a(.1,1.), m*n*kelement_size, csr_forward,
                   kelement_size*n,
                   0,
                   20,
                   rc,
                   rsn);
           csrirp (b(.1,1.), n*p*kelement_size, csr_forward,
                   (p-1)*kelement_size,
                   0,
                   50,
                   rc,
                   rsn);
      for i:=1 to m do
        for J:=1 to p do
          begin;
          c(.i,j.) := 0;
          for k:=1 to n do
            c(.i,j.) := c(.i,j.) + a(.i,k.) * b(.k,j.);
          end;

           csrrrp (a(.1,1.), m*n*kelement_size,
                   rc,
                   rsn);
           csrrrp (b(.1,1.), n*p*kelement_size,
                   rc,
                   rsn);
   END.
*********************************************************************  00010000
*                                                                   *  00020000
*       JCL TO COMPILE AND LINKEDIT                                 *  00030000
*                                                                   *  00040000
*********************************************************************  00050000
//PASCJOB JOB                                                           00060008
//GOGO EXEC PAS22CL                                                     00100000
//*                                                                     00110000
//*    COMPILE AND LINKEDIT FOR PASCAL                                  00120000
//*                                                                     00130000
//*    CHANGE THE MEMBER NAME ON THE NEXT LINE AND THE                  00140000
//*    NAME CRTPANXX(R) SIX LINES DOWN                                  00150000
//*                                                                     00160000
//PASC.SYSLIB DD                                                        00161006
//            DD                                                        00162006
//            DD DSN=REFPAT.DECLARE.SET(CSRBPPAS),DISP=SHR              00163008
//PASC.SYSIN DD DSN=REFPAT.SAMPLE.PROG(PASCAL),DISP=SHR                 00170008
//LKED.SYSLMOD DD DSN=REFPAT.USER.LOAD,DISP=SHR,UNIT=3380,              00180008
// VOL=SER=VM2TSO                                                       00190009
//LKED.SYSIN DD *                                                       00200000
  LIBRARY  IN(CSRIRP,CSRRRP)                                           00210005
  NAME BPGPASC(R)                                                      00220003
/*                                                                      00230000
//*     SYS1.CSSLIB IS THE SOURCE OF THE CSR STUBS                      00240008
//*                                                                     00250000
//LKED.IN     DD DSN=SYS1.CSSLIB,DISP=SHR                               00260008
*********************************************************************
*                                                                   *
*       JCL TO EXECUTE PASCAL                                       *
*                                                                   *
*********************************************************************
```

```
//PASC1JOB JOB                                                          00010005
//GO   EXEC PAS22CL                                                     00050000
//*                                                                     00050102
//*    Compile and linkedit for PASCAL                                  00050202
//*                                                                     00050302
//PASC.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTPAN06),DISP=SHR               00060006
//LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,DISP=SHR,UNIT=3380,             00560000
// VOL=SER=VM2TSO                                                      00570000
//LKED.SYSIN DD *                                                     00580000
  LIBRARY  IN(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC)        00590000
  NAME CRTPAN06(R)                                                    00600006
/*                                                                    00610000
//*     SYS1.CSSLIB is the source of the CSR stubs                    00620002
//*                                                                   00650002
//LKED.IN     DD DSN=SYS1.CSSLIB,DISP=SHR                             00690000
*********************************************************************
*                                                                   *
*                                                                   *
*      JCL TO COMPILE AND LINKEDIT.                                 *
*                                                                   *
*                                                                   *
*                                                                   *
*********************************************************************
*********************************************************************  00010000
*                                                                   *  00020000
*   JCL TO EXECUTE.  THIS ONE NEEDS A DD STATEMENT FOR THE          *  00030000
*   PERMANENT DIV OBJECT - CSRDD1.  DATASET ALREADY EXISTS.         *  00040000
*                                                                   *  00060000
*********************************************************************  00070000
//PASCGO JOB MSGLEVEL=(1,1),TIME=1440                                   00080002
//*                                                                     00090000
//*                                                                     00100000
//*   RUN A PASCAL TESTCASE - CHANGE THE NAME ON THE NEXT LINE          00110000
//*                                                                     00/20000
//*                                                                     00130000
//GO     EXEC  PGM=BPGPASC                                            00140000
//STEPLIB DD DSN=REFPAT.USER.LOAD,                                     00150002
// DISP=SHR,UNIT=3380,                                                 00190000
// VOL=SER=VM2TSO                                                      00200003
//CSRDD1   DD  DSN=DIV.TESTDS,DISP=SHR                                 00210000
//OUTPUT   DD  SYSOUT=A,DCB=(RECFM=VBA,LRECL=133)                      00220000
//SYSPRINT DD  SYSOUT=A,DCB=(RECFM=VBA,LRECL=133)                      00230000
---------------------------
```

PL/I Example

```
*********************************************************************
*                                                                   *
*     PLI example                                                   *
*                                                                   *
*********************************************************************
 BPGPLI: PROCEDURE OPTIONS(MAIN);                                     00010023
                                                                      00020002
  %INCLUDE SYSLIB(CSRBPPLI);                                          00020122
                                                                      00020222
     /* INITs */                                                      00021013
 DCL M INIT(512) FIXED BIN(31);                                       00022035
 DCL N INIT(512) FIXED BIN(31);                                       00023035
 DCL P INIT(512) FIXED BIN(31);                                       00024035
                                                                      00025013
     /* Arrays */                                                     00026013
 DCL A (M,N) BIN FIXED(31);           /* First array               */ 00029113
 DCL B (N,P) BIN FIXED(31);           /* Second array              */ 00029213
 DCL C (M,P) BIN FIXED(31);           /* Product of first and second */ 00029313
 DCL KELEMENT_SIZE INIT(4) FIXED BIN(31); /* Size of an element of an  00029416
                                        array. This value is tied     00029513
                                        directly to the data type of  00029613
```

PL/I Example

```
                                        the three arrays (ie. FIXED(31)00029713
                                        is 4 bytes                  */ 00029813
                                                                       00029913
    /* Indices */                                                      00030013
DCL I FIXED BIN(31),                                                   00031013
    J FIXED BIN(31),                                                   00031113
    K FIXED BIN(31);                                                   00031213
                                                                       00032013
    /* Others */                                                       00037013
DCL RC FIXED BIN(31);                                                  00039013
DCL RSN FIXED BIN(31);                                                 00039113
                                                                       00390108
                                                                       00391808
    /* Initialize the first two arrays such that each element          00411013
       equals the sum of the indices for that element (eg.             00412013
       A(4,10) = 14 */                                                 00413013
                                                                       00414013
CALL CSRIRP (A(1,1), M*N*KELEMENT_SIZE, CSR_FORWARD,                   00415013
        KELEMENT_SIZE*N,                                               00416013
        0,                                                             00417013
        20,                                                            00418013
        RC,                                                            00419013
        RSN);                                                          00419113
CALL CSRIRP (B(1,1), N*P*KELEMENT_SIZE, CSR_FORWARD,                   00419913
        KELEMENT_SIZE*P,                                               00420013
        0,                                                             00420113
        20,                                                            00420213
        RC,                                                            00420313
        RSN);                                                          00420413
DO I = 1 TO M;                                                         00421213
  DO J = 1 TO N;                                                       00421313
    A(I,J) = I + J;                                                    00421413
  END;                                                                 00421513
END;                                                                   00421613
                                                                       00421713
DO I = 1 TO N;                                                         00421813
  DO J = 1 TO P;                                                       00421913
    B(I,J) = I + J;                                                    00422013
  END;                                                                 00422113
END;                                                                   00422213
CALL CSRRRP (A(1,1), M*N*KELEMENT_SIZE,                                00422313
        RC,                                                            00422513
        RSN);                                                          00422613
CALL CSRRRP (B(1,1), N*P*KELEMENT_SIZE,                                00423413
        RC,                                                            00423613
        RSN);                                                          00423713
                                                                       00424513
    /* Multiply the two arrays together */                             00424613
                                                                       00424713
CALL CSRIRP (A(1,1), M*N*KELEMENT_SIZE, CSR_FORWARD,                   00424813
        KELEMENT_SIZE*N,                                               00424913
        0,                                                             00425013
        20,                                                            00425133
        RC,                                                            00425213
        RSN);                                                          00425313
CALL CSRIRP (B(1,1), N*P*KELEMENT_SIZE, CSR_FORWARD,                   00426113
        KELEMENT_SIZE,                                                 00426213
        (P-1)*KELEMENT_SIZE,                                           00426313
        50,                                                            00426413
        RC,                                                            00426513
        RSN);                                                          00426613
DO I = 1 TO M;                                                         00427413
  DO J = 1 TO P;                                                       00427513
    C(I,J) = 0;                                                        00427613
    DO K = 1 TO N;                                                     00427713
      C(I,J) = C(I,J) + A(I,K) * B(K,J);                               00427813
    END;                                                               00427913
```

```
  END;                                                                  00428013
END;                                                                    00428113
                                                                        00428213
CALL CSRRRP (A(1,1), M*N*KELEMENT_SIZE,                                 00428313
       RC,                                                              00428513
       RSN);                                                            00428613
CALL CSRRRP (B(1,1), N*P*KELEMENT_SIZE,                                 00429413
       RC,                                                              00429613
       RSN);                                                            00429713
                                                                        00430513
END BPGPLI;                                                             01080024
*********************************************************************
*                                                                   *
*                                                                   *
*      JCL TO COMPILE AND LINKEDIT.                                 *
*                                                                   *
*                                                                   *
*                                                                   *
*********************************************************************
//PLIJOB JOB                                                          00010007
//*                                                                   00041001
//*  PL/I Compile and Linkedit                                        00042001
//*                                                                   00043001
//*  Change all CRTPLNx to CRTPLNy                                    00044001
//*                                                                   00045001
//GO EXEC PLIXCL,PARM.PLI='MACRO'                                     00050000
//PLI.SYSLIB DD DSN=REFPAT.DECLARE.SET,DISP=SHR
//PLI.SYSIN DD DSN=REFPAT.SAMPLE.PROG(PLI),DISP=SHR                   00060008
//LKED.SYSLMOD DD DSN=REFPAT.USER.LOAD,UNIT=3380,VOL=SER=RSMPAK,      00070000
// DISP=SHR                                                           00080000
//LKED.SYSIN DD *                                                     00090000
  INCLUDE  IN(CSRIRP,CSRRRP)                                        00100001
  NAME BPGPLI(R)                                                    00110008
/*                                                                  00120000
//*                                                                 00121001
//*     SYS1.CSSLIB is source of CSR stubs                          00130001
//*                                                                 00190000
//LKED.IN     DD DSN=SYS1.CSSLIB,DISP=SHR                           00200000
//PLIJOB JOB                                                        00010007
*********************************************************************
*                                                                   *
*                                                                   *
*      JCL TO EXECUTE.                                              *
*                                                                   *
*                                                                   *
*                                                                   *
*********************************************************************
//PLIRUN   JOB MSGLEVEL=(1,1),TIME=1440                               00010000
//*                                                                   00011001
//*   EXECUTE A PL/I TESTCASE - CHANGE NAME ON NEXT LINE              00012001
//*                                                                   00013001
//GO       EXEC PGM=CRTPLN3                                           00020000
//STEPLIB  DD  DSN=REFPAT.USER.LOAD,DISP=SHR,                         00030000
//     UNIT=3380,VOL=SER=VM2TSO                                       00040000
//         DD DSN=CEE.SCEERUN,DISP=SHR                                0
//SYSABEND DD SYSOUT=*                                                00070000
//SYSLOUT  DD  SYSOUT=*                                               00080000
//SYSPRINT DD  SYSOUT=*                                               00090000
```

PL/I Example

Part 3. Global Resource Serialization Latch Manager Services

Chapter 9. Using the Latch Manager Services

To use global resource serialization latch manager services, you issue CALLs from high level language programs. Each service requires a set of parameters coded in a specific order on the CALL statement.

This chapter describes the CALL statements that invoke latch manager services. Each description includes a syntax diagram, parameter descriptions, and return and reason code explanations with recommended actions. Return and reason codes are shown in hexadecimal and decimal, along with the associated equate symbol.

This chapter contains the following topics:

For information about the basic function of the latch manager, how to plan to use the latch manager, and how to use the latch manager callable services, see the serialization topic in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Syntax and Linkage Conventions for Latch Manager Callable Services

The latch manager callable services have the following general calling syntax:

CALL *routine_name*(*parameters*)

Some specific calling formats for languages that can invoke the latch manager callable services are:

C routine_name (parm1,parm2,...return_code)

COBOL
CALL "*routine_name*" USING *parm1*,*parm2*,...*return_code*

FORTRAN
CALL *routine_name* (*parm1*,*parm2*,...*return_code*)

PL/I
CALL *routine_name* (*parm1*,*parm2*,...*return_code*)

REXX
ADDRESS LU62 "routine_name parm1 parm2...return_code"

IBM provides files, called interface definition files (IDFs), that define variables and values for the parameters used with latch manager services. IBM provides IDFs for some of the listed languages. See the serialization topic in *z/OS MVS Programming: Authorized Assembler Services Guide* for information about the IDFs that are available on MVS.

ISGLCRT — Create a Latch Set

Call the Latch_Create service to create a set of latches. Your application should call Latch_Create during application initialization, and specify a number of latches that is sufficient to serialize all the resources that the application requires. Programs that run as part of the application can call the following related services:

ISGLOBT Requests exclusive or shared ownership of a latch.

ISGLREL Releases ownership of an owned latch or a pending request to obtain a latch.

ISGLPRG Purges all granted and pending requests for a particular requestor within a specific latch set.

In the following description of Latch_Create, constants defined in the latch manager IDFs are followed by their numeric equivalents; you may specify either when coding calls to Latch_Create.

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:
- number_of_latches
- latch_set_name
- create_option

Latch_Create returns values in the following parameters:
- latch_set_token
- return_code

CALL ISGLCRT	(number_of_latches ,latch_set_name ,create_option ,latch_set_token ,return_code)

The parameters are explained as follows:

number_of_latches
Specifies a fullword integer that indicates the number of latches to be created.

,latch_set_name
Specifies a 48-byte area that contains the name of the latch set. The latch set name must be unique within the current address space. The latch set name can be any value up to 48 characters, but the first character must not be binary zeros or an EBCDIC blank. If the latch set name is less than 48 characters, it must be padded on the right with blanks.

IBM recommends that you use a standard naming convention for the latch set name. To avoid using a name that IBM uses, do not begin the latch set name with the character string **SYS**. It is a good idea to select a latch set name that is readable in output from the DISPLAY GRS command and interactive problem control system (IPCS). Avoid '@', '$', and '#' because those characters do not always display consistently.

,create_option
Specifies a fullword integer that must have a value of ISGLCRT_PRIVATE (or a value of 0).

,latch_set_token
Specifies an 8-byte area to contain the latch set token returned by the Latch_Create service. The latch set token uniquely identifies the latch set. Programs must specify this value on calls to the Latch_Obtain, Latch_Release, and Latch_Purge services.

,return_code
A fullword integer to contain the return code from the Latch_Create service.

ABEND Codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses.

Return Codes

When the Latch_Create service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 9-1. ISGLCRT Return Codes

Return code and Equate symbol	Meaning and Action
00 (0) ISGLCRT_SUCCESS	**Meaning**: The Latch_Create service completed successfully. **Action**: None required.
04 (4) ISGLCRT_DUPLICATE_NAME	**Meaning**: The specified latch_set_name already exists, and is associated with a latch set that was created by a program running in the current primary address space. The latch manager does not create a new latch set. **Action**: To create a new latch set, specify a unique name on the latch_set_name parameter, then call the Latch_Create service again. Otherwise, continue processing with the returned latch set token.
10 (16) ISGLCRT_NO_STORAGE	**Meaning**: Environmental error. Not enough storage was available to contain the requested number of latches. The latch manager does not create a new latch set. **Action**: Specify a smaller value on the number_of_latches parameter.

Examples of Calls to Latch Manager Services

The following is an example of how to call all the latch manager services in C language:

```
/*********************************************************************/
/* C Example                                                         */
/*********************************************************************/
#pragma linkage(setsup, OS)
#pragma linkage(setprob, OS)
#include <ISGLMC.H>                /* Include C language IDF        */

main()
{
  const int numberOfLatches = 16;   /* in this example we create 16
                                       latches                       */
```

```
  ISGLM_LSNM_type latchSetName
                 = "EXAMPLE.ONE_LATCH_SET_NAME                    ";
                                  /* set up 48-byte latch set name */
  ISGLM_LSTK_type latchSetToken;  /* latch set token - output from
                                     create and input to obtain,
                                     release, and purge            */
  int returnCode = 0;             /* return code from services     */

  const int latchNumber = 6;      /* in this example we obtain latch
                                     six                           */
  ISGLM_LRID_type requestorID = "123";/* requestor ID - output from
                                     obtain and input to purge     */
  int ECB = 0;                    /* ECB used for latch obtain
                                     service                       */
  ISGLM_EADDR_type ECBaddress = &ECB;/* pointer to ECB             */
  ISGLM_LTK_type latchToken;      /* latch token - output from
                                     obtain and input to release   */
  union {
          double alignment;       /* force double word alignment   */
          ISGLM_WA_type area;     /* set up work area              */
        } work;


  setsup();                       /* set supervisor state PSW      */

/*********************************************************************/
/* create a latch set with 16 latches                                */
/*********************************************************************/

  isglcrt(numberOfLatches
         ,latchSetName
         ,ISGLCRT_PRIVATE
         ,&latchSetToken;
         ,&returnCode);

/*********************************************************************/
/* obtain latch                                                      */
/*********************************************************************/

  isglobt(latchSetToken
         ,latchNumber
         ,requestorID
         ,ISGLOBT_SYNC            /* suspend until granted         */
         ,ISGLOBT_EXCLUSIVE       /* access option (exclusive)     */
         ,&ECBaddress             /* required, but not used        */
         ,&latchToken             /* identifies request            */
         ,&work.area
         ,&returnCode);

/*********************************************************************/
/* release latch                                                     */
/*********************************************************************/

  isglrel(latchSetToken
         ,latchToken
         ,ISGLREL_UNCOND          /* ABEND if latch not owned      */
         ,&workarea
         ,&returnCode);

/*********************************************************************/
/* purge requestor from latch set                                    */
/*********************************************************************/

  isglprg(latchSetToken
         ,requestorID
         ,&returnCode);

  setprob();                      /* set problem state PSW         */
```

```
}
*********************************************************************
* SETSUP subroutine
*********************************************************************
SETSUP   CSECT
SETSUP   AMODE 31
SETSUP   RMODE ANY
         SAVE  (14,12)             save regs
         SAC   0                   ensure primary mode
         LR    12,15               establish addressability
         USING SETSUP,12
         MODESET MODE=SUP          set supervisor state
         RETURN (14,12),RC=0       restore caller's regs and return
         END SETSUP
*********************************************************************
* SETPROB subroutine
*********************************************************************
SETPROB  CSECT
SETPROB  AMODE 31
SETPROB  RMODE ANY
         SAVE  (14,12)             save regs
         LR    12,15               establish addressability
         USING SETPROB,12
         MODESET MODE=PROB         set problem state
         RETURN (14,12),RC=0       restore caller's regs and return
         END SETPROB
```

ISGLOBT — Obtain a Latch

Call the Latch_Obtain service to request exclusive or shared ownership of a latch. When a requestor owns a particular latch, the requestor can use the resource associated with that latch. The following callable services are related to Latch_Obtain:

ISGLCRT Creates a latch set that an application can use to serialize resources.

ISGLREL Releases ownership of an owned latch or a pending request to obtain a latch.

ISGLPRG Purges all granted and pending requests for a particular requestor within a specific latch set.

In the following description of Latch_Obtain:

- The term *requestor* describes a task or SRB routine that calls the Latch_Obtain service to request ownership of a latch.
- Constants defined in the latch manager IDFs are followed by their numeric equivalents; you may specify either when coding calls to Latch_Obtain. For example, “ISGLOBT_COND (value of 1)” indicates the constant ISGLOBT_COND and its associated value, 1.

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- latch_set_token
- latch_number
- requestor_ID
- obtain_option
- access_option
- ECB_address

Latch_Obtain returns values in the following parameters:
- latch_set_token
- return_code

Latch_Obtain uses the following parameter for temporary storage:
- work_area

CALL ISGLOBT	(latch_set_token ,latch_number ,requestor_ID ,obtain_option ,access_option ,ECB_address ,latch_token ,work_area ,return_code)

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch_set_token that the Latch_Create service returned earlier when it created the latch set.

,latch_number
Specifies a fullword integer that contains the number of the latch to be obtained. The latch_number must be in the range from 0 to the total number of latches in the associated latch set minus one.

,requestor_ID
Specifies an 8-byte area that contains a value that identifies the caller of the Latch_Obtain service. The requestor_ID can be any value except all binary zeros.

Recovery routines can purge all granted and pending requests for a particular requestor (identified by a requestor_id) within a specific latch set. When specifying the requestor_ID on Latch_Obtain, consider which latches would be purged if the Latch_Purge service were to be called with the specified requestor_ID. For more information about the Latch_Purge service, see “ISGLPRG — Purge a Requestor from a Latch Set” on page 9-11.

,obtain_option
A fullword integer that specifies how the system is to handle the Latch_Obtain request if the latch manager cannot immediately grant ownership of the latch to the requestor:

ISGLOBT_SYNC (value of 0)
The system processes the request synchronously. The system suspends the requestor. When the latch manager eventually grants ownership of the latch to the requestor, the system returns control to the requestor.

ISGLOBT_COND (value of 1)
The system processes the request conditionally. The system returns control to the requestor with a return code of ISGLOBT_CONTENTION (value of 4). The latch manager does not queue the request to obtain the latch.

ISGLOBT_ASYNC_ECB (value of 2)
The system processes the request asynchronously. The system returns control to the requestor with a return code of ISGLOBT_CONTENTION

(value of 4). When the latch manager eventually grants ownership of the latch to the requestor, the system posts the ECB pointed to by the value specified on the ECB_address parameter.

When you specify this option, the ECB_address parameter must contain the address of an initialized ECB that is addressable from the home address space (HASN).

,access_option

A fullword or character string that specifies the access required:

- ISGLOBT_EXCLUSIVE (value of 0) - Exclusive (write) access
- ISGLOBT_SHARED (value of 1) - Shared (read) access

,ECB_address

Specifies a fullword that contains the address of an ECB. If you specify an obtain_option of ISGLOBT_SYNC (value of 0) or ISGLOBT_COND (value of 1) on the call to Latch_Obtain, the ECB_address field must be valid (though its contents are ignored). IBM recommends that an address of 0 be used when no ECB is to be processed.

If you specify an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) and the system returns a return code of ISGLOBT_CONTENTION (value of 4) to the caller, the system posts the ECB pointed to by the value specified on the ECB_address parameter when the latch manager grants ownership of the latch to the requestor.

,latch_token

Specifies an 8-byte area to contain the latch token returned by the Latch_Obtain service. You must provide this value as a parameter on a call to the Latch_Release service to release the latch.

,work_area

Specifies a 256-byte work area that provides temporary storage for the Latch_Obtain service. The work area should begin on a doubleword boundary to optimize performance. The work area must be in the same storage key as the caller of Latch_Obtain.

,return_code

Specifies a fullword integer that is to contain the return code from the Latch_Obtain service.

ABEND Codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses for these codes.

Return Codes

When the Latch_Obtain service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 9-2. ISGLOBT Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLOBT_SUCCESS	**Meaning**: The Latch_Obtain service completed successfully. **Action**: None.

Table 9-2. ISGLOBT Return Codes (continued)

Return code and Equate Symbol	Meaning and Action
04 (4) ISGLOBT_CONTENTION	**Meaning**: A requestor called Latch_Obtain with an obtain_option of ISGLOBT_COND (value of 1) or ISGLOBT_ASYNC_ECB (value of 2). The latch is not immediately available. **Action**: If the requestor specified an obtain_option of ISGLOBT_COND (value of 1), no response is required. If the requestor specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2), and the latch is still required, wait on the ECB to be posted when the latch manager grants ownership of the latch to the requestor.

Example

See “Examples of Calls to Latch Manager Services” on page 9-3 for an example of how to call Latch_Obtain in C language.

ISGLREL — Release a Latch

Call the Latch_Release service to release ownership of an owned latch or a pending request to obtain a latch. Requestors should call Latch_Release when the use of a resource associated with a latch is no longer required. The following callable services are related to Latch_Release:

ISGLCRT Creates a latch set that an application can use to serialize resources.

ISGLOBT Requests exclusive or shared control of a latch.

ISGLPRG Purges all granted and pending requests for a particular requestor within a specific latch set.

In the following description of Latch_Release:

- The term *requestor* describes a program that calls the Latch_Release service to release ownership of an owned latch or a pending request to obtain a latch.
- Constants defined in the latch manager IDFs are followed by their numeric equivalents; you may specify either when coding calls to Latch_Obtain. For example, “ISGLREL_COND (value of 1)” indicates the constant ISGLREL_COND and its associated value, 1.

Write the CALL as shown on the syntax diagram, coding all parameters in the specified order.

Assign values to the following parameters:
- latch_set_token
- latch_token
- release_option

Latch_Release returns a value in the following parameter:
- return_code

Latch_Release uses the following parameter for temporary storage:
- work_area

CALL ISGLREL	(latch_set_token ,latch_token ,release_option ,work_area ,return_code)

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch set token returned to the caller of the Latch_Create service. The latch set token identifies the latch set that contains the latch to be released.

,latch_token
Specifies an 8-byte area that contains the latch token returned to the caller of the Latch_Obtain service. The latch token identifies the request to be released.

,release_option
Specifies a fullword integer that tells the latch manager what to do when the requestor either no longer owns the latch to be released or still has a pending request to obtain the latch to be released:

ISGLREL_UNCOND (value of 0)
Abend the requestor:

- If a requestor originally specified an obtain_option of ISGLOBT_SYNC (value of 0) when obtaining the latch, the latch manager does not release the latch. The system abends the caller of Latch_Release with abend X'9C6', reason code xxxx0009.
- If a requestor originally specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) when obtaining the latch, the latch manager does not release the latch. The system abends the caller of Latch_Release with abend X'9C6', reason code xxxx0007.
- If the latch manager does not find a previous Latch_Obtain request for the specified latch, the system abends the caller of Latch_Release with abend X'9C6', reason code xxxx000A.

ISGLREL_COND (value of 1)
Return control to the requestor:

- If a requestor originally specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) when obtaining the latch, the latch manager releases the request for ownership of the latch. The system returns control to the caller of Latch_Release with a return code of ISGLREL_NOT_OWNED_ECB_REQUEST (value of 4).
- If a requestor originally specified an obtain_option of ISGLOBT_SYNC (value of 0) when obtaining the latch, the latch manager does not release the request for ownership of the latch. The system returns control to the caller of Latch_Release with a return code of ISGLREL_STILL_SUSPENDED (value of 8).
- If the latch manager does not find a previous Latch_Obtain request for the specified latch, the system returns control to the caller of Latch_Release with a return code of ISGLREL_INCORRECT_LATCH_TOKEN (value of 12).

,work_area
Specifies a 256-byte work area that provides temporary storage for the

Latch_Release service. The work area should begin on a doubleword boundary to optimize performance. The work area must be in the same storage key as the caller of Latch_Release.

,return_code
Specifies a fullword integer that is to contain the return code from the Latch_Release service.

ABEND Codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses.

Return Codes

When the Latch_Release service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 9-3. ISGLREL Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLREL_SUCCESS	**Meaning**: The Latch_Release service completed successfully. The caller released ownership of the specified latch request. **Action**: None.
04 (4) ISGLREL_NOT_OWNED_ECB_REQUEST	**Meaning**: The requestor that originally called the Latch_Obtain service is still expecting the system to post an ECB (to indicate that the requestor has obtained the latch). The call to the Latch_Release service specified a release_option of ISGLREL_COND (value of 1). The latch manager does not post the ECB at the address specified on the original call to Latch_Obtain. The latch manager releases the latch. **Action**: Validate the integrity of the resource associated with the latch (the requestor might have used the resource without waiting on the ECB). If the resource is undamaged, no action is necessary (a requestor routine may have been in the process of cancelling the request to obtain the latch).
08 (8) ISGLREL_STILL_SUSPENDED	**Meaning**: Program error. The request specified a correct latch token, but the program that originally requested the latch is still suspended and waiting to obtain the latch. The latch requestor originally specified an obtain_option of ISGLOBT_SYNC on the call to the Latch_Obtain service. The call to the Latch_Release service specified a release_option of ISGLREL_COND (value of 1). The latch manager does not release the latch. The latch requestor remains suspended. **Action**: • Wait for the latch requestor to obtain the latch and receive control back from the system; then call the Latch_Release service again, or • End the program that originally requested the latch.

Table 9-3. ISGLREL Return Codes (continued)

Return code and Equate Symbol	Meaning and Action
0C (12) ISGLREL_INCORRECT_LATCH_TOKEN	**Meaning**: The latch manager could not find a granted or pending request associated with the value on the latch token parameter. The latch manager does not release a latch. This return code does not indicate an error if a routine calls Latch_Release to ensure that a latch is released. For example, if an error occurs when a requestor calls the Latch_Obtain service, the requestor's recovery routine might call Latch_Release to ensure that the requested latch is released. If the error prevented the requestor from obtaining the latch, the recovery routine receives this return code. **Action**: If the return code is not expected, validate that the latch token is the same latch token returned to the caller of Latch_Obtain.

Example

See "Examples of Calls to Latch Manager Services" on page 9-3 for an example of how to call Latch_Release in C language.

ISGLPRG — Purge a Requestor from a Latch Set

Call the Latch_Purge service to purge all granted and pending requests for a particular requestor within a specific latch set. Recovery routines should call Latch_Purge when one or more errors prevent requestors from releasing latches. The following callable services are related to Latch_Purge:

ISGLCRT Creates a latch set that an application can use to serialize resources.

ISGLOBT Requests exclusive or shared control of a latch.

ISGLREL Releases control of an owned latch or a pending request to obtain a latch.

In the following description of Latch_Purge, constants defined in the latch manager IDFs are followed by their numeric equivalents; you may specify either when coding calls to Latch_Purge.

Write the CALL as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:
- latch_set_token
- requestor_ID

Latch_Purge returns a value in the return_code parameter.

CALL ISGLPRG	(latch_set_token ,requestor_ID ,return_code)

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch_set_token previously returned by the Latch_Create service. The latch set token identifies the latch set from which latch requests are to be purged.

,requestor_ID
Specifies an 8-byte area that contains the requestor_ID originally specified on one or more previous calls to the Latch_Obtain service. The Latch_Purge service is to release all Latch_Obtain requests that specify this requestor_ID.

,return_code
A fullword integer that contains the return code from the Latch_Purge service.

ABEND Codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses.

Return Codes

When the Latch_Purge service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in paretheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 9-4. ISGLPRG Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLPRG_SUCCESS	**Meaning**: The Latch_Purge service completed successfully. **Action**: None.
04 (4) ISGLPRG_DAMAGE_DETECTED	**Meaning**: Program error. While purging all requests for a particular requestor from a latch set, the latch manager found incorrect data in one or more latches. The latch manager tries to purge the latches that contain incorrect data, but the damage might prevent the latch manager from purging those latches. The latch manager purges the remaining latches (those with *correct* data) for the specified requestor. **Action**: Take a dump and check for a storage overlay. If your application can continue without the resources serialized by the damaged latches, no action is required.

Example

See “Examples of Calls to Latch Manager Services” on page 9-3 for an example of how to call Latch_Purge in C language.

ISGLPBA — Purge a Group of Requestors from a Group of Latch Sets

Call the Latch_Purge_by_Address_Space service to purge all granted and pending requests for a group of requestors for a group of latch sets in the same address space. To effectively use this service, your latch_set_names and your requestor_IDs should be defined such that they have a common portion and a unique portion. Groups of latch sets can then be formed by masking off the unique portion of the latch_set_name, and groups of latch requests in a latch set can then be formed by masking off the unique portion of the requestor_ID. Masking off the unique portion of the requestor_ID allows a single purge request to handle multiple latch sets and

multiple requests in a latch set. Recovery routines should call Latch_Purge_by_Address_Space when one or more errors prevent requestors from releasing latches.

The following callable services are related to Latch_Purge_by_Address_Space:

ISGLCRT Creates a latch set that an application can use to serialize resources.

ISGLOBT Requests exclusive or shared control of a latch.

ISGLREL Releases control of an owned latch or a pending request to obtain a latch.

ISGLPRG Purges all granted and pending requests for a particular requestor within a specific latch set.

In the following description of Latch_Purge_by_Address_Space, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Purge_by_Address_Space.

Write the CALL as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:
- latch_set_token
- requestor_ID
- requestor_ID_mask
- latch_set_name
- latch_set_name_mask

Latch_Purge_by_Address_Space returns a value in the return_code parameter.

CALL ISGLPBA	(latch_set_token ,requestor_ID ,requestor_ID_mask ,latch_set_name ,latch_set_name_mask ,return_code)

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch_set_token previously returned by the Latch_Create service or a value of zero. If the value is not zero, the latch_set_token identifies the latch set from which latch requests are to be purged. If the latch_set_token is set to zero, a group of latch sets, determined by the latch_set_name and latch_set_name_mask, will have their latch requests purged.

,requestor_id
Specifies an 8-byte area that contains a portion of the requestor_ID originally specified on one or more previous calls to the Latch_Obtain service. This operand will be compared to the result of logically ANDing each requestor_ID in the latch set with the requestor_ID_mask. Make sure that any corresponding

bits that are zero in the requestor_ID_mask are also zero in this field, otherwise no ID matches will occur. Each requestor_ID that has a name match will have its Latch_Obtain requests released.

,requestor_id_mask
Specifies an 8-byte area that contains the requestor_ID_mask that will be logically ANDed to each requestor_ID in the latch set and then compared to the requestor_ID operand. Each requestor_ID that has a name match will have its Latch_Obtain requests released.

,latch_set_name
Specifies a 48-byte area that contains the portion of the latch_set_name that will be compared to the result of logically ANDing the latch_set_name_mask with each latch set name in the primary address space. Make sure that any corresponding bits that are zero in the latch_set_name_mask are also zero in this field, otherwise no name matches will occur. Each latch set that has a name match will have its Latch_Obtain requests released. If the latch_set_token operand is non-zero this operand is ignored.

,latch_set_name_mask
Specifies a 48-byte area that contains the mask that will be logically ANDed to each of the latch set names in the primary address apace and then compared to the latch_set_name operand. Each latch set that has a name match will have its Latch_Obtain requests released. If the latch_set_token operand is non-zero this operand is ignored.

,return_code
A fullwprd integer that contains the return code from the Latch_Purge_By_Address_Space service.

ABEND Codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses.

Return Codes

When the Latch_Purge_by_Address_Space service returns control to your program, the return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 9-5. ISGLPBA Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLPRG_SUCCESS	**Meaning**: The Latch_Purge_by_Address_Space service completed successfully. **Action**: None.
04 (4) ISGLPRG_DAMAGE_DETECTED	**Meaning**: Program error. While purging all requests for a particular requestor from a latch set, the latch manager found incorrect data in one or more latches. The latch manager tries to purge the latches that contain incorrect data, but the damage might prevent the latch manager from purging those latches. The latch manager purges the remaining latches (those with *correct* data) for the specified requestor. **Action**: Take a dump and check for a storage overlay. If your application can continue without the resources serialized by the damaged latches, no action is required.

Part 4. Resource Recovery Services (RRS)

Chapter 10. Using Protected Resources

Many computer resources are so critical to a company's work that the integrity of these resources must be guaranteed. If changes to the data in the resources are corrupted by a hardware or software failure, human error, or a catastrophe, the computer must be able to restore the data. These critical resources are called *protected resources* or, sometimes, *recoverable resources*.

The system, when requested, can coordinate changes to one or more protected resources so that all changes are made or no changes are made. Resources that the system can protect are, for example:

- A hierarchical database
- A relational database
- A product-specific resource

Resource recovery is the protection of the resources. Resource recovery consists of the protocols and program interfaces that allow an application program to make consistent changes to multiple protected resources.

Resource Recovery Programs

Three programs work together to protect resources:

- **Application program**: The application program accesses protected resources and requests changes to the resources.
- **Resource manager**: A resource manager is an authorized program that controls and manages access to a resource. A resource manager provides interfaces that allow the application program to read and change a protected resource. The resource manager also takes actions that commit or back out changes to a resource it manages.

  Often an application changes more than one protected resource, so that more than one resource manager is involved.

  A resource manager may be an IBM product, part of an IBM product, or a product from another vendor. A resource manager can be:

  – A database manager, such as DB2®

  – A program, such as IMS/ESA® Transaction Manager, that accepts work from an end user or another system and manages that work

  Note: The resource manager in resource recovery is different from an RTM resource manager, which is related to the operating system's recovery termination management (RTM) and runs during termination processing.
- **Sync-point manager**: The sync-point manager coordinates changes to protected resources, so that all changes are made or no changes are made. The z/OS sync-point manager is recoverable resource management services (RRMS). Three MVS components provide RRMS function; because resource recovery services (RRS) provides the sync-point services, most technical information uses RRS rather than RRMS.

  If your resources are distributed, so that they are on multiple systems, the communication resource manager on one system will coordinate the changes. Each communication resource manager works with RRS on its system.

RRS can enable resource recovery on a single system or, with APPC/MVS, on multiple systems.

The application program, resource manager, and sync-point manager use a two-phase commit protocol to protect resources.

Two-Phase Commit Protocol

The two-phase commit protocol is a set of actions used to make sure that an application program makes all changes to a collection of resources or makes no changes to the collection. The protocol makes sure of the **all-or-nothing changes** even if the system, RRS, or the resource manager fails.

The phases of the protocol are:

- **Phase 1**: In the first phase, each resource manager must be prepared to either commit or backout the changes. They prepare for the commit and tell RRS either YES, the change can be made, or NO, the change cannot be made.

  First, RRS decides the results of the YES or NO responses from the resource managers. If the decision is YES to commit the changes, RRS hardens the decision, meaning that it stores the decision in an RRS log.

  Once a commit decision is hardened, the application changes are considered committed. If there is a failure after this point, the resource manager will make the changes during restart. Before this point, a failure causes the resource manager to back out the changes during restart.
- **Phase 2**: In the second phase, the resource managers commit or back out the changes.

Resource Recovery Process

For a look at the resource recovery process, think of a person who requests an automated teller machine (ATM) to transfer money from a savings account to a checking account. The application program receives the person's input from the ATM. Each account is in a different database. Each database has its own resource manager. The sync-point manager is RRS. Figure 10-1 shows how the ATM application, resource managers, and RRS work together

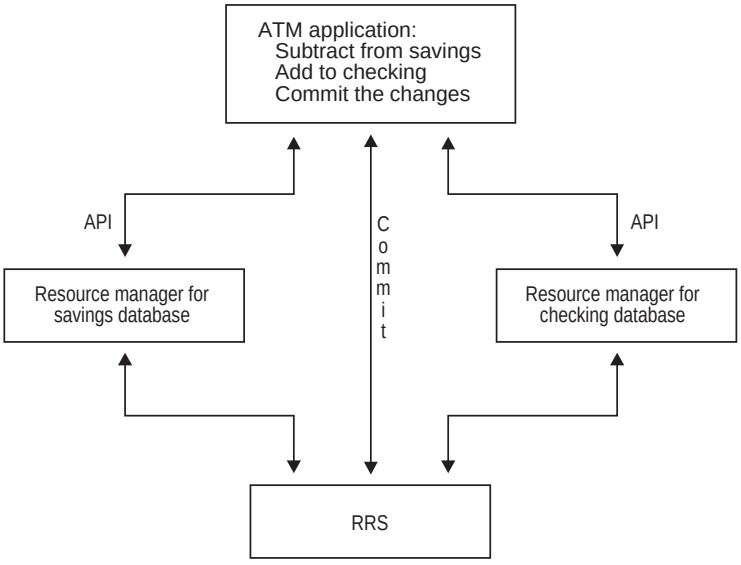


Figure 10-1. ATM Transaction

The actions required to process the ATM transaction are:

1. The ATM user requests transfer of money from a savings account to a checking account.
2. The ATM application program receives the ATM input.

Figure 10-2 shows, for the same transaction, the sequence of the following actions, with time moving from left to right, in the two-phase commit protocol RRS uses to commit the changes. The top line in the figure shows the two phases of the protocol described in “Two-Phase Commit Protocol” on page 10-2.

3. The ATM application requests the savings resource manager to subtract the money from the savings database. For this step, the application uses the resource manager’s application programming interface (API).
4. The ATM application requests the checking resource manager to add the money to the checking database. The application uses this resource manager’s API.
5. The ATM application issues a call to RRS to commit the database changes.
6. RRS asks the resource managers to prepare for the changes.
7. The resource managers indicate whether or not they can make the changes, by voting YES or NO. In Figure 10-2, both resource managers vote YES.
8. In response, RRS notifies the resource managers to commit the changes, that is, to make the changes permanently in the databases.
9. The resource managers complete the commit and return OK to RRS.
10. RRS gives a return code to the application program, indicating that all changes were made in the databases.

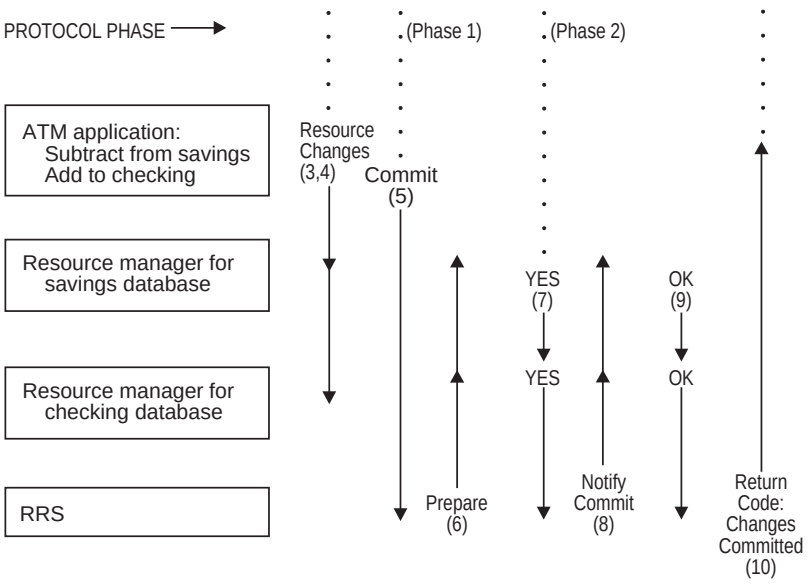


Figure 10-2. Two-Phase Commit Actions

If the ATM user decides not to transfer the money and presses a NO selection, the application requests backout, instead of commit, in step 6. In this case, the changes are backed out and are not actually made in any database. See Figure 10-3 on page 10-4.

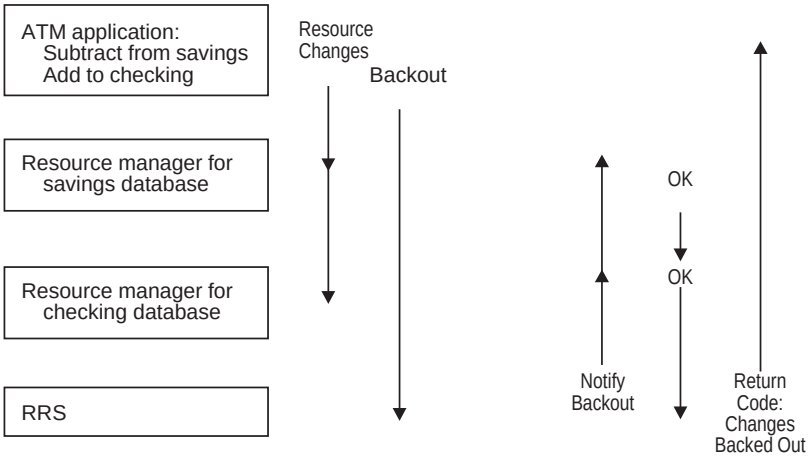


Figure 10-3. Backout — Application Request

Or if a resource manager cannot make the change to its database, the resource manager votes NO during prepare. If **any** resource manager votes NO, all of the changes are backed out. See Figure 10-4.

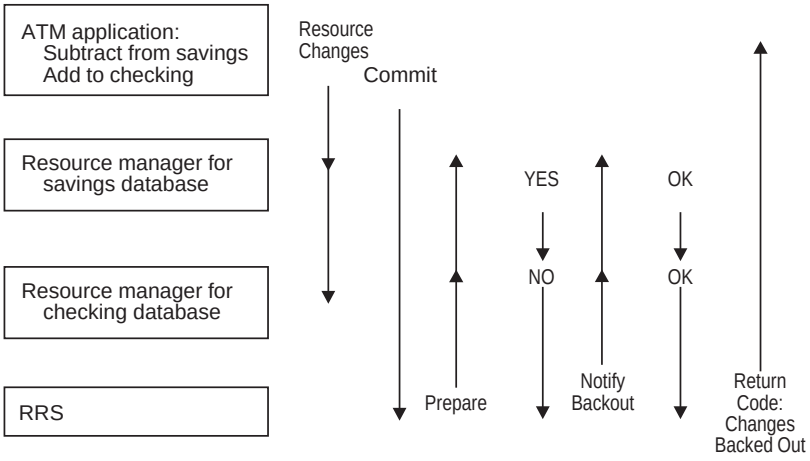


Figure 10-4. Backout — Resource Manager Votes NO

Requesting Resource Protection and Recovery

To request resource protection, your application program must use resource managers that work with RRS to protect resources. The code in your application should do the following:

1. Request one or more accesses to resources for reads, writes, or both.
2. If all of the changes are to be made, request commit by issuing a call to the Application_Commit_UR service.
3. If none of the changes are to be made, request backout by issuing a call to the Application_Backout_UR service.

For details about the calls, see “Application_Backout_UR (SRRBACK)” on page 10-5 and “Application_Commit_UR (SRRCMIT)” on page 10-9.

Using Distributed Resource Recovery

The databases for a work request may be distributed, residing on more than one system. In this case, the application program initiating the work uses a distributed communications manager, such as APPC/MVS, to request changes by an application program on another system. The database resource managers, communication resource managers, and RRS components work together to make or not make all changes of both application programs. Figure 10-5 illustrates distributed resource recovery.

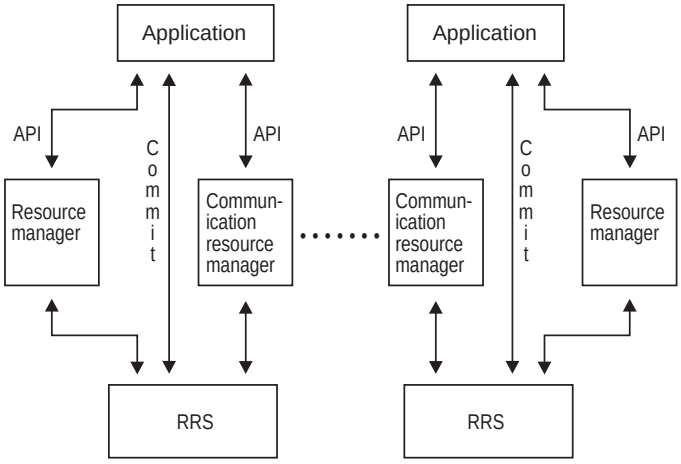


Figure 10-5. Transaction — Distributed Resource Recovery

Application_Backout_UR (SRRBACK)

Call the Application_Backout_UR service to indicate that the changes for the unit of recovery (UR) are not to be made. A UR represents the application's changes to resources since the last commit or backout or, for the first UR, since the beginning of the application. In response to the call, RRS requests that the resource managers return their resources to the values they had before the UR was processed.

An application might need to issue a call to the Application_Backout_UR service if:

- An APPC/MVS call returns a `TAKE_BACKOUT` return code. For example, a CI *send_data* call to a communications manager could return `TAKE_BACKOUT`.
- A resource manager call returns a return code that indicates that a resource manager directly backed out its resource. This situation can occur if the resource manager does not have the capability to return a `TAKE_BACKOUT` code.
- A communications resource manager call returns a return code that indicates that a backout must be done, such as a return code of `COM_RESOURCE_FAILURE_NO_RETRY` from a CI call.

Description

Environment

The requirements for the caller are:

Minimum authorization: Problem state, any PSW key
Dispatchable unit mode: Task
Cross memory mode: Any PASN, any HASN, any SASN

AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming Requirements

The two methods described here can be used to access the callable service.

- Linkedit the stub routine ATRSCSS with the program that uses the service. ATRSCSS resides in SYS1.CSSLIB.
- Code the MVS LOAD macro within a program that uses the service to obtain the entry point address of the service. Use that address to call the service.

Additional language-specific statements may be necessary so that compilers can provide the proper assembler interface. Other programming notations, such as variable declarations, are also language-dependent.

SYS1.CSSLIB contains stubs for all of MVS's callable services including RRS. Other program products like DB2 and IMS™ also provide libraries that contain stubs for their versions of SRRBACK and SRRCMIT.

Because other program products like DB2 and IMS provide their own stubs for SRRBACK or SRRCMIT, you must make sure your program uses the correct stub. You need to take particular care when recompiling and linkediting any application that uses these services. When you linkedit, make sure that the data sets in the syslib concatenation are in the right order. For example, if you want a DB2 application to use the RRS callable service SRRBACK or SRRCMIT, you must ensure that SYS1.CSSLIB precedes the data sets with the stubs that DB2 provides for SRRBACK or SRRCMIT.

If you inadvertently cause your program to use SRRCMIT for RRS when it expects SRRCMIT for another program product like IMS, the application does not run correctly, and your program receives an error return code from the call to SRRCMIT.

For examples of the JCL link edit statements used with high-level languages, see Chapter 4, "Window Services Coding Examples" on page 4-1 or Chapter 8, "Reference Pattern Services Coding Examples" on page 8-1.

High Level Language (HLL) Definitions: The high level language (HLL) definitions for the callable service are:

HLL Definition	Description
ATRSASM	390 Assembler declarations
ATRSC	C/390 declarations
ATRSCOB	COBOL 390 declarations
ATRSPAS	Pascal 390 declarations
ATRSPLI	PL/I 390 declarations

Assembler: If you are an Assembler language caller running in AMODE 24, either use a BASSM instruction in place of the CALL or specify a LINKINST=BASSM parameter on the CALL macro. For example:

```
CALL SRRBACK(RETCODE),LINKINST=BASSM
```

COBOL: The return/reason code names and abend code names in ATRSCOB are truncated at 30 characters.

PL/I: The return/reason code names and abend code names in ATRSPLI are truncated at 31 characters.

Restrictions: The state of the UR must be **in-reset** or **in-flight**. A successful call creates a new UR that is **in-reset**.

The UR cannot be in local transaction mode.

Input Register Information: Before issuing the call, the caller does not have to place any information into any register unless using it in register notation for the parameter, or using it as a base register.

Output Register Information: When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a call. If the system changes the contents of registers on which the caller depends, the caller must save them before calling the service, and restore them after the system returns control.

Performance Implications: None.

Syntax: Write the call as shown in the syntax diagram. You must code the parameters in the CALL statement as shown.

CALL SRRBACK	(return_code)

Parameters: The parameters are explained as follows:

return_code
Returned parameter
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Application_Backout_UR service.

ABEND Codes: The call might result in an abend X'5C4' with a reason code of X'00150000' through X'00150010'. See *z/OS MVS System Codes* for the explanations and actions.

If your application ends abnormally during sync-point processing, the condition is called an asynchronous abend, and you might need to see the programmer at your installation responsible for managing RRS. Under information about working with application programs, *z/OS MVS Programming: Resource Recovery* contains additional details about asynchronous abends.

Issuing SETRRS CANCEL for non-resource manager programs that use the synch-point service results in an abend X'058'. When RRS restarts, transactions that were in progress are resolved.

Return Codes: When the service returns control to your program, GPR 15 and *return_code* contain a hexadecimal return code, shown in the following table. If you need help with a return code, see the programmer at your installation responsible for managing RRS. Under information about working with application programs, *z/OS MVS Programming: Resource Recovery* contains additional details about these return codes.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
0	0	**Code**: RR_OK **Meaning**: Successful completion. The resource managers returned their resources to the values they had before the UR was processed. **Action**: None.
12D	301	**Code**: RR_BACKED_OUT_OUTCOME_PENDING **Meaning**: Environmental error. The backout was not completed, for one of the following reasons: • RRS requested that the resource managers back out the changes to the resources. However, the state of one or more of the resources is not known. • RRS is not active. • The resource manager fails with an incomplete protected interest in the UR, or RRS fails before the UR is complete. **Action**: The action by an application depends on the system environment. Some possible actions are: • Display a warning message to the end user. • Write an exception entry into an output log. • Abnormally end the application because the resource manager will not allow any further changes to the resource until the situation is resolved.
12E	302	**Code**: RR_BACKED_OUT_OUTCOME_MIXED **Meaning**: Environmental error. RRS requested that the resource managers back out the changes to the resources. However, one or more resources were changed. **Action**: Same as the action for return code 12D (301).

Example: In the pseudocode example, the application issues a call to request that RRS back out a UR.

```
.
.
.
CALL SRRBACK(RETCODE)
.
.
.
```

Application_Commit_UR (SRRCMIT)

Call the Application_Commit_UR service to indicate that the changes for the unit of recovery (UR) are to be made permanent. A UR represents the application's changes to resources since the last commit or backout or, for the first UR, since the beginning of the application. In response to the call, RRS requests that the resource managers make the changes permanent.

Certain resource managers, such as a communications manager, can issue a `TAKE_COMMIT` return code to an application that has requested changes to resources. In response to the `TAKE_COMMIT` code from the resource manager, the application should request the changes to the resources:

- If all of the change requests are accepted, call the Application_Commit_UR service again.
- If any of the change requests are not accepted. call the Application_Backout_UR service to back out the changes.

Description

Environment

The requirements for the caller are:

Minimum authorization:	Problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming Requirements

The two methods described here can be used to access the callable service.

- Linkedit the stub routine ATRSCSS with the program that uses the service. ATRSCSS resides in SYS1.CSSLIB.
- Code the MVS LOAD macro within a program that uses the service to obtain the entry point address of the service. Use that address to call the service.

Additional language-specific statements may be necessary so that compilers can provide the proper assembler interface. Other programming notations, such as variable declarations, are also language-dependent.

SYS1.CSSLIB contains stubs for all of MVS's callable services including RRS. Other program products like DB2 and IMS also provide libraries that contain stubs for their versions of SRRBACK and SRRCMIT.

Because other program products like DB2 and IMS provide their own stubs for SRRBACK or SRRCMIT, you must make sure your program uses the correct stub. You need to take particular care when recompiling and linkediting any application

that uses these services. When you linkedit, make sure that the data sets in the syslib concatenation are in the right order. For example, if you want a DB2 application to use the RRS callable service SRRBACK or SRRCMIT, you must ensure that SYS1.CSSLIB precedes the data sets with the stubs that DB2 provides for SRRBACK or SRRCMIT.

If you inadvertently cause your program to use SRRCMIT for RRS when it expects SRRCMIT for another program product like IMS, the application does not run correctly, and your program receives an error return code from the call to SRRCMIT.

For examples of the JCL link edit statements for high-level languages, see Chapter 4, “Window Services Coding Examples” on page 4-1 or Chapter 8, “Reference Pattern Services Coding Examples” on page 8-1.

High Level Language (HLL) Definitions: The high level language (HLL) definitions for the callable service are:

HLL Definition	Description
ATRSASM	390 Assembler declarations
ATRSC	C/390 declarations
ATRSCOB	COBOL 390 declarations
ATRSPAS	Pascal 390 declarations
ATRSPLI	PL/I 390 declarations

Assembler: If you are an Assembler language caller running in AMODE 24, either use a BASSM instruction in place of the CALL or specify a LINKINST=BASSM parameter on the CALL macro. For example:

```
CALL SRRCMIT(RETCODE),LINKINST=BASSM
```

COBOL: The return/reason code names and abend code names in ATRSCOB are truncated at 30 characters.

PL/I: The return/reason code names and abend code names in ATRSPLI are truncated at 31 characters.

Restrictions

The state of the UR that represents the changes must be **in-reset** or **in-flight**.

The UR cannot be in local transaction mode.

Input Register Information

Before issuing the call, the caller does not have to place any information into any register unless using it in register notation for the parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a call. If the system changes the contents of registers on which the caller depends, the caller must save them before calling the service, and restore them after the system returns control.

Performance Implications
None.

Syntax
Write the call as shown in the syntax diagram. You must code the parameter in the CALL statement as shown.

CALL SRRCMIT	(return_code)

Parameters
The parameters are explained as follows:

return_code
Returned parameter
- Type: Integer
- Length: 4 bytes

Contains the return code from the Application_Commit_UR service.

ABEND Codes
The call might result in an abend X'5C4' with a reason code of X'00160000' through X'00160012'. See *z/OS MVS System Codes* for the explanations and actions.

If your application ends abnormally during sync-point processing, the condition is called an asynchronous abend, and you might need to see the programmer at your installation responsible for managing RRS. Under information about working with application programs, *z/OS MVS Programming: Resource Recovery* contains additional details about asynchronous abends.

Issuing SETRRS CANCEL for non-resource manager programs that use the synch-point service results in an abend X'058'. When RRS restarts, transactions that were in progress are resolved.

Return Codes
When the service returns control to your program, GPR 15 and *return_code* contain a hexadecimal return code, shown in the following table. If you need help with a return code, see the programmer at your installation responsible for managing RRS. Under information about working with application programs, *z/OS MVS Programming: Resource Recovery* contains additional details about these return codes.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
0	0	**Code**: RR_OK **Meaning**: Successful completion. The changes to all protected resources have been made permanent. **Action**: None.
65	101	**Code**: RR_COMMITTED_OUTCOME_PENDING **Meaning**: Environmental error. The commit was not completed: • RRS requested that the resource managers make the changes to the resources permanent. However, the state of one or more of the resources is not known. **Action**: The action by an application depends on the system environment. Some possible actions are: • Display a warning message to the end user. • Write an exception entry into an output log. • Abnormally end the application because the resource manager will not allow any further changes to the resource until the situation is resolved.
66	102	**Code**: RR_COMMITTED_OUTCOME_MIXED **Meaning**: Environmental error. RRS requested that the resource managers make the changes to the resources permanent. One or more resources were changed, but one or more were not changed. **Action**: Same as the action for return code 65 (101).
C8	200	**Code**: RR_PROGRAM_STATE_CHECK **Meaning**: Environmental error. The commit failed. The resource managers did not make the changes to the resources because one of the following occurred: • A resource on the same system as the application is not in the proper state for a commit. • A protected conversation is not in the required state: **send**, **send pending**, **defer receive**, **defer allocate**, **sync_point**, **sync_point send**, **sync_point deallocate**. • A protected conversation is in **send** state. The communications manager started sending the basic conversation logical record, but did not finish sending it. **Action**: Initiate an action by a resource manager to get its resource to a committable state, then call Application_Commit_UR again. For example, if the application has allocated a protected conversation through APPPC/MVS, and the conversation is in receive state, the application gets this return code. It then must use APPC/MVS services to change the conversation to send state before issuing the commit request again.
12C	300	**Code**: RR_BACKED_OUT **Meaning**: Environmental error. The commit failed. The resource managers backed out the changes, returning the resources to the values they had before the UR was processed. **Action**: Same as the action for return code 65 (101).

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
12D	301	**Code**: RR_BACKED_OUT_OUTCOME_PENDING **Meaning**: Environmental error. The commit failed for one of the following reasons: • RRS requested that the resource managers back out the changes to the resources. However, the state of one or more of the resources is not known. • RRS is not active. **Action**: Same as the action for return code 65 (101).
12E	302	**Code**: RR_BACKED_OUT_OUTCOME_MIXED **Meaning**: Environmental error. The commit failed. RRS requested that the resource managers back out the changes to the resources. One or more resources were backed out, but one or more were changed. **Action**: Same as the action for return code 65 (101).

Example

In the pseudocode example, the application issues a call to request that RRS commit a UR.

```
⋮
CALL SRRCMIT(RETCODE)
⋮
```

Part 5. Other Callable Services

Chapter 11. IEAAFFN — Assign Processor Affinity for Encryption or Decryption

Call IEAAFFN when the only function performed by your program is to encrypt or decrypt data. Encryption and decryption take place on processors that have Integrated Cryptographic Features (ICRFs) associated with them. IEAAFFN assigns a program affinity to processors with an ICRF; that is, IEAAFFN makes sure the system runs your program on a processor that has an ICRF associated with it.

You do **not** have to use the IEAAFFN service to ensure the system runs a program on a processor with an ICRF; the system ensures that automatically. However, you can avoid some of the system overhead involved in the selection process by using the IEAAFFN service. IBM recommends that you use the service in programs whose **only** function is encryption or decryption.

Note: When you use this service to either establish or remove processor affinity for a program, the program permanently loses any processor affinity that the system programmer assigned to it in the SCHEDxx member of SYS1.PARMLIB.

Code the CALL following the syntax of the high level language you are using and specifying all parameters in the order shown below.

CALL IEAAFFN	(feature ,operation_type ,return_code)

The parameters are explained as follows:

feature
Specifies the feature required by your program. Specify CRYPTO to indicate an ICRF.

Define *feature* as character data of length 10. Pad the string on the right with 4 blanks.

,operation_type
Specifies the type of action you want to take. The types are:

GRANT Establish affinity for the program to processors with an ICRF.

REMOVE Remove affinity for the program to processors with an ICRF.

Note: After you issue a REMOVE request, the program has no processor affinity; it can run on any processor.

Define *operation_type* as character data of length 6. If you specify GRANT, pad the string on the right with 1 blank.

,return_code
When IEAAFFN completes, *return_code* contains the return code from the service. The return code value is also in register 15.

Define *return_code* as integer data of length 4. The return codes are explained under “Return Codes” on page 11-2.

Restrictions and Limitations

Use the IEAAFFN service to request affinity to processors with an ICRF only for sections of a program that require an ICRF and not other features, such as a Vector Facility.

Requirements

Authorization:	Supervisor state or Problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	You can be either in cross memory mode or not
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	None held
Control parameters:	Must be in the primary address space

Return Codes

When IEAAFFN returns control to your program, *return_code* and register 15 contain a return code. The following table identifies the return codes in hexadecimal and decimal (in parentheses), tells what each means, and recommends an action that you should take.

Table 11-1. IEAAFFN Return Codes

Return code	Meaning and Action
00000000 (0)	**Meaning**: The operation was successful. **Action**: None required.
00000004 (4)	**Meaning**: The program already had processor affinity assigned to it by the system programmer. The system replaces that affinity with the affinity you requested in this service. **Action**: None required.
0000000C (12)	**Meaning**: Your program was not running in task mode. **Action**: This service is not available to SRB mode programs. See the FEATURE= option on the SCHEDULE macro for the use of this function in SRB mode.
00000010 (16)	**Meaning**: The feature you specified was not a valid feature. **Action**: Specify a valid feature name.
00000014 (20)	**Meaning**: The operation type you specified was not valid. **Action**: Specify a valid operation type.
00000018 (24)	**Meaning**: The feature you specified is not installed on any of the processors in the system. **Action**: To the system programmer: See that the program runs on a system with the feature installed.
0000001C (28)	**Meaning**: A system error has occurred. **Action**: To the system programmer: The error is recorded in LOGREC. Look for a record with a subcomponent of “IEAAFFN CSS”; then call your IBM Support Center.

Chapter 12. CSRL16J — Transfer Control to Another Routine

The CSRL16J service allows you to transfer control to another routine running under the same request block (RB) as the calling program. The CSRL16J service will transfer control with the contents of all 16 registers intact. When you transfer control to the other routine, use the CSRL16J service to:

- Define the entry characteristics and register contents for the target routine.
- Optionally free dynamic storage associated with the calling program.

When the service is successful, control transfers to the target routine. After the target routine runs, it can transfer control to any program running under the same request block (RB), including the calling program.

The CSRL16J service returns control to the calling program **only** when it cannot transfer control successfully to the target because of an error.

Defining the Entry Characteristics of the Target Routine

Specify the entry characteristics for the target in data area L16J, which forms the parameter list passed from the calling program to CSRL16J. Use the CSRYL16J mapping macro to see the format of the L16J parameter list. To build the L16J parameter list, first initialize the parameter list with zeroes and then fill in the desired fields. This ensures that all fields requiring zeroes are correct. You can specify the following characteristics for the target in L16J:

- Length of the L16J parameter list, L16JLENGTH field in mapping macro CSRYL16J.
- Contents of the general purpose registers (GPRs) 0-15, L16JGRS field in mapping macro CSRYL16J.
- Contents of the access registers (ARs) 0-15, L16JARS field in mapping macro CSRYL16J.
- PSW information for the target routine, field L16JPSW field in mapping macro CSRYL16J.
  - PSW address and AMODE
  - PSW ASC mode - primary or AR
  - PSW program mask
  - PSW condition code

  Authorized callers, (callers in supervisor state, with PSW key 0-7, or with a PKM that allows any key 0-7) can specify:
  - PSW state - problem or supervisor
  - PSW key.

  For unauthorized callers, the system uses the PSW state and key of the calling program for the target routine.

  See *Principles of Operation* for more information about the contents of the PSW.
- Bit indicating whether or not you want to specify the contents of the access registers (ARs) for the target routine. This is the L16JPROCESSARS bit in mapping macro CSRYL16J.

  Set the bit on if you want to specify the contents of the ARs. If you set the bit off, the system determines the contents of the ARs.

If the bit is set on when CSRL16J passes control to the target routine, the access registers (ARs) contain:

Register	Contents
0-15	Specified by the caller

If the bit is set off when CSRL16J passes control to the target routine, the access registers (ARs) contain:

Register	Contents
0-1	Do not contain any information for use by the routine
2-13	The contents are the same as they were when the caller issued the CSRL16J service.
14-15	Do not contain any information for use by the routine

Freeing Dynamic Storage Associated with the Caller

If the calling program has a dynamic storage area associated with it, you can specify that some or all of this storage area be freed before CSRL16J transfers control to the target. In the L16J parameter list, specify:

- The subpool of the area that you want the system to free. L16JSUBPOOL field in mapping macro CSRYL16J.
- The length, in bytes, of the dynamic storage area you want the system to free. L16JLENGTHTOFREE field in mapping macro CSRYL16J.
- The address of the dynamic storage area you want the system to free. L16JAREATOFREE field in mapping macro CSRYL16J.

  Make sure that the address is on a double-word boundary. Otherwise the service ends with an abend code X'978'. See *z/OS MVS System Codes* for information on abend code X'978'.

The system frees the storage only when the CSRL16J service is successful.

Programming Requirements

These are the requirements:

- The calling program must be in 31-bit addressing mode.
- Before you use the CSRL16J service, you must build a parameter list, L16J, to pass to the service. The parameter list includes the entry characteristics and environment for the target.

  If you are coding in C/370, you can include the CSRLJC macro to provide declarations in the calling program for the L16J parameter area and return codes. See Figure 12-1 on page 12-3.

  If you are coding in PL/I, you can include the CSRLJPLI macro to provide declarations for the return codes only. See Figure 12-2 on page 12-5 for the CSRLJPLI macro. Use the data area, mapped by the CSRYL16J mapping macro, as a model for the structure of your parameter list when coding in PL/I.

CSRLJC provides the following declarations for use in your C/370 program:

```
/*********************************************************************
 *         Type Definitions for User Specified Parameters            *
 *********************************************************************/

/*  Type for user supplied L16J                                      */
typedef struct ??<
  int Version;     /* Must be 0                                      */
  int Length;      /* Initialize to CSRL16J_LENGTH                   */
  int SubPool;     /* Subpool of storage to be freed                 */
  union ??<
    char GRs??(64??);  /* General registers                         */
    int  GR??(16??);   /* General register 0-15                     */
  ??> u1;
  union ??<
    char ARs??(64??);  /* Access registers                          */
    int  AR??(16??);   /* Access register 0-15                      */
  ??> u2;
  union ??<
    char PSW??(8??);   /* PSW: the processing will use the address,
                          AMODE, ASC mode, CC, and program mask. For a
                          supervisor state or PKM 0-7 or key 0-7
                          caller, it will use the state and key from
                          the PSW. Otherwise, it will set to caller
                          key and state.                            */
    struct ??<
      int PSWByte0to3 : 32;  /* First 4 bytes                       */
      union ??<
        void *PSWAddr;  /* Address and AMODE                        */
        struct ??<
          int PSWAmode : 1;  /* AMODE                              */
          int Rsvd0 : 31;
        ??> s2;
      ??> u4;
    ??> s1;
  ??> u3;
  union ??<
    struct ??<
      int Flags : 8;    /* Flags                                   */
      int Rsvd0     : 24;   /* Reserved                            */
    ??> s3;
    struct ??<
      int ProcessARs : 1;  /* If on, ARs will be processed. Otherwise
                not. If not processed, ARs 0, 1, 14, and 15 are
                unpredictable. ARs 2-13 are taken from the values
                present when the service is entered.                */
      int Rsvd0  : 31;     /* Reserved                             */
    ??> s4;
  ??> u5;
```

Figure 12-1. CSRLJC declarations for the L16J parameter list for C/370 (Part 1 of 2)

```
  void *AreaToFree;   /* Address of area to free. If this is non-0
                         then the area will be freed using the subpool
                         specified in L16J.Subpool. This can be used
                         to free the caller's entire dynamic area if
                         so desired. When this option is specified, it
                         is necessary that the area begin on a
                         doubleword boundary.                          */
  int LengthToFree;   /* Length of area to free, in bytes.            */
  char Rsvd??(8??);   /* Reserved                                     */
??> L16J;
/*********************************************************************
 *       Fixed Service Parameter and Return Code Defines             *
 *********************************************************************/

#define CSRL16J_LENGTH            168    /* Length of L16J           */

/*  Service Return Codes                                             */
#define CSRL16J_OK                0
#define CSRL16J_BAD_VERSION       4
#define CSRL16J_BAD_AMODE         8
#define CSRL16J_BAD_RESERVED      12
#define CSRL16J_BAD_LENGTH        16
#define CSRL16J_BAD_PSW           24

/*********************************************************************
 *           Function Prototypes for Service Routines                *
 *********************************************************************/

extern void csrl16j(
     L16J *__L16J,      /* Input  - User supplied L16J block         */
     int *__RC);        /* Output - Return code                      */


/*********************************************************************

#endif
```

Figure 12-1. CSRLJC declarations for the L16J parameter list for C/370 (Part 2 of 2)

CSRLJPLI provides the following declarations for use in your PL/I program:

```
/*****************************************************************
 *      Constants for Fixed Return Codes                         *
 *****************************************************************/

/*  Load 16 and Jump Service Return Codes                        */

%DCL CSRL16J_OK FIXED;
%CSRL16J_OK           = 0;

%DCL CSRL16J_BAD_VERSION FIXED;
%CSRL16J_BAD_VERSION  = 4;

%DCL CSRL16J_BAD_AMODE FIXED;
%CSRL16J_BAD_AMODE    = 8;

%DCL CSRL16J_BAD_RESERVED FIXED;
%CSRL16J_BAD_RESERVED = 12;

%DCL CSRL16J_BAD_LENGTH FIXED;
%CSRL16J_BAD_LENGTH   = 16;

%DCL CSRL16J_BAD_PSW FIXED;
%CSRL16J_BAD_PSW      = 24;

/*****************************************************************
 *              Service Entry Declarations                       *
 *****************************************************************/

 DCL CSRL16J ENTRY
      (CHAR(168),     /* Input  - L16J                           */
       FIXED BIN(31)) /* Output - Return code                    */
      OPTIONS(INTER ASSEMBLER);


/*  End of Load 16 and Jump Service Declares                     */
```

Figure 12-2. CSRLJPLI declarations for return codes for PL/I

Restrictions

None.

Performance Implications

None.

Syntax Diagram

Code the invocation following the syntax of the language you are using. Specify parameters in the order shown.

C/370 Syntax

csrl16j	(&L16J ,&return_code)

PL/I Syntax

CALL CSRL16J	(L16J ,return_code)

Parameters

The parameters are explained as follows:

L16J
Specifies a parameter list that the service uses to define the entry characteristics and environment for the target.

return_code
When the service completes, *return_code* contains the return code.

Return Codes

If the CSRL16J service returns control to the caller, an error has occurred and the service was unable to transfer control to the target routine. In this case, the return code is always nonzero. When the service successfully transfers control to the target routine, the return code is zero.

Return codes from the CSRL16J service are as follows:

Table 12-1. CSRL16J Return Codes

Return Code (hexadecimal)	Meaning and Action
00	**Meaning**: Successful completion. The calling program will never see this return code because it indicates that the target routine received control. **Action**: None.
04	**Meaning**: The value specified in the L16JVERSION field of the L16J data area was not a zero. The L16JVERSION field must contain a value of zero. **Action**: When you build the L16J data area, first zero the entire L16J data area and then fill in the required fields. This process ensures that all fields that must contain zeroes are correct.
08	**Meaning**: The calling program was not in 31-bit addressing mode, which is required. **Action**: Make sure the calling program is in 31-bit addressing mode.
0C	**Meaning**: One of the fields in the L16J data area that is reserved for IBM use contained a nonzero value. Any field reserved for IBM use must contain a value of zero. **Action**: When you build the L16J data area, first zero the entire L16J data area and then fill in the required fields. This process ensures that all fields that must contain zeroes are correct
10	**Meaning**: The value specified in field L16JLENGTH in the L16J data area was less than the actual length of the L16J. **Action**: Make sure that the value in the L16JLENGTH field reflects the actual length of the L16J data area.
18	**Meaning**: The PSW provided in field L16JPSW of the L16J data area specified an incorrect ASC mode. **Action**: In the L16JPSW field, specify either primary or AR ASC mode.

Example

The following example, coded in C/370 uses CSRL16J to transfer control to a C/370 program. The target routine executes in the mode and with the register contents specified by the calling program in the L16J parameter list.

This example performs the following operations:

- Fills in L16J parameter list with PSW and execution mode data.
- Calls an assembler routine to obtain the current register contents of registers 0 through 13 and copies them to the L16J parameter list.
- Defines the contents of registers 14 and 15 for the target routine.
- Issues setjmp to allow return from the target routine.
- Invokes the C/370 function L16JPrg through CSRL16J.
- CSRL16J issues longjmp to return to caller and complete processing.

To use this example, you must also use the assembler program following the C/370 example.

C/370 Example Program

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <setjmp.h>
#include "CSRLJC.H"

#define FALSE 0
#define TRUE  1

/* REG0TO13 is the assembler assist routine (below) to extract
   registers 0 through 13, for C/370 addressability */
#pragma linkage(REG0TO13,OS)

int          rcode;
int          i;
unsigned int regs??(14??);  /* Register save area */
jmp_buf      JumpBuffer;    /* Buffer for setjmp/longjmp */
L16J         L16JParmArea;  /* L16J parameter list structure */

/* Function prototype for function to be called via L16J */
void L16JPrg();

/* Invoke a C/370 function via L16J Callable Services */
main()
{
  /* Start by initializing the entire L16J parameter list */
  memset(&L16JParmArea,'\0',sizeof(L16J));

  /* The following fields were implicitly initialized to zero
     by the preceding statement:
       L16JParmArea.Version
       L16JParmArea.SubPool
       L16JParmArea.AreaToFree
       L16JParmArea.LengthToFree
     These field do not need to be explicitly set unless a value
     other than zero is required */

  /* Place parameter list length size into parameter list */
  L16JParmArea.Length = sizeof(L16J);
```

```
  /* Create a Problem State/Key 8 PSW */
  L16JParmArea.u3.s1.PSWByte0to3 = 0x078D1000;
  L16JParmArea.u3.s1.u4.PSWAddr = (void *) &L16JPrg;

  /* Mode data */
  L16JParmArea.u3.s1.u4.s2.PSWAmode = 1;
  L16JParmArea.u5.s4.ProcessARs = 1;

  /* Call assembler assist routine to obtain current register
     values */
  REG0TO13(&regs);

  /* Place register values into parameter list */
  for (i=0;i<14;i++)
     L16JParmArea.u1.GR??(i??)= regs??(i??);

  /* Register 14 is not being used in this linkage, but we
     have set it to zero for this example */
  L16JParmArea.u1.GRAddr??(14??) = 0;

  /* Set register 15 for entry to routine */
  L16JParmArea.u1.GRAddr??(15??) = (void *) &L16JPrg;

  printf("L16JC - Call L16J to invoke L16JPrg\n");

  /* Use setjmp to allow return to this point in program. If
     setjmp is being called for the first time, invoke L16JPrg
     via L16J Callable Services.  If returning from longjmp,
     skip call to L16J services and complete processing. */
  if (!setjmp(JumpBuffer))
  {
    csrl16j (&L16JParmArea,&rcode);

    /* Demonstrate use of L16J C/370 declares */
    switch (rcode)
    {
      /* Select on a particular return code value */
      case CSRL16J_BAD_PSW:
        printf("L16JC - L16J unsuccessful, bad PSW\n");
        break;
      /* Default error processing */
      default:
        printf("L16JC - L16J unsuccessful, RC = %d\n",rcode);
        break;
    }
  }
  printf("L16JC - Returned from L16JPrg\n");
}

/* The routine below receives control via L16J Callable Services.
   control is passed back to main via longjmp. */
void L16JPrg(void)
{
  printf("L16JC - L16JPrg got control\n");
  longjmp(JumpBuffer,1);
}
```

Assembler program for use with the C/370 example

To use this example you must assemble the following program and linkedit it with the C/370 program above.

```
SR0T013 CSECT
SR0T013 AMODE 31
SR0T013 RMODE ANY
*
* Assembler assist routine to save contents of registers 0 through 13
```

```
* to the area pointed to by register 1.
*
REG0TO13  DS    0H
          ENTRY REG0TO13
* Get address of the save area
          L     15,0(,1)
* Save registers 0 to 13
          STM   0,13,0(15)
* Return to the caller
          BR    14
          END   SR0TO13
```

Chapter 13. CSRSI — System Information Service

Use the CSRSI service to retrieve system information. You can request information about the machine itself, the logical partition (LPAR) in which the machine is running, or the virtual machine hypervisor (VM) under which the system is running. The returned information is mapped by DSECTs in macro CSRSIIDF (for assembler language callers) or structures in header file CSRSIC (for C language callers).

The information available depends upon the availability of the Store System Information (STSI) instruction. When the STSI instruction is not available (which would be indicated by receiving the return code 4 (equate symbol CSRSI_STSINOTAVAILABLE), only the SI00PCCACPID, SI00PCCACPUA, and SI00PCCACAFM fields within the returned infoarea are valid. When the STSI instruction is available, the validity of the returned infoarea depends upon the system:

- If the system is running neither under LPAR nor VM, then only the CSRSI_Request_V1CPC_Machine data are valid.
- If the system is running under a logical partition (LPAR), then both the CSRSI_Request_V1CPC_Machine data and CSRSI_Request_V2CPC_LPAR data are valid.
- If the system is running under a virtual machine hypervisor (VM), then all of the data (CSRSI_Request_V1CPC_Machine, CSRSI_Request_V2CPC_LPAR, and CSRSI_Request_V3CPC_VM) are valid.

You can request any or all of the information regardless of your system, and validity bits will indicate which returned areas are valid.

Description

Environment

The requirements for the caller are:

Minimum authorization:	Problem state, key 8–15
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit when using the CALL CSRSI form (or csrsi in C), 31-bit when using an alternate form
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold a LOCAL lock, the CMS lock, or the CPU lock but is not required to hold any locks.

Programming Requirements

The caller should include the CSRSIIDF macro to map the returned information and to provide equates for the service.

Restrictions

None.

Input Register Information

The caller is not required by the system to set up any registers.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Syntax

CALL CSRSI,	(Request ,Infoarealen ,Infoarea ,Returncode)

In C: the syntax is similar. You can use either of the following techniques to invoke the service:

1. `CSRSI (Request,...Returncode);`
   When you use this technique, you must link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB.
2. `CSRSI_byaddr (Request,...Returncode);`
   This second technique requires AMODE=31, and, before you issue the CALL, you must verify that the CSRSI service is available (in the CVT, both CVTOSEXT and CVTCSRSI bits are set on).

In Assembler: Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use either of the following techniques as an alternative to CALL CSRSI:

1.
```
LOAD EP=CSRSI
Save the entry point address
...
Put the saved entry point address into R15
Issue CALL (15),...
```

2.
```
L   15,X'10'                Get CVT
L   15,X'220'(,15)
L   15,X'30'(,15)           Get address of CSRSI
CALL (15),(...)
```

Both of these techniques require AMODE=31. If you use the second technique, before you issue the CALL, you must verify that the CSRSI service is available (in the CVT, both CVTOSEXT and CVTCSRSI bits are set on).

Parameters

Request

Supplied parameter:
- Type: Integer
- Length: Full word

Request identifies the type of system information to be returned. The field must contain a value that represents one or more of the possible request types. You add the values to create the full word. Do not specify a request more than once. The possible requests, and their meanings, are:

CSRSI_Request_V1CPC_Machine
The system is to return information about the machine.

CSRSI_Request_V2CPC_LPAR
The system is to return information about the logical partition (LPAR).

CSRSI_Request_V3CPC_VM
The system is to return information about the virtual machine (VM).

,Infoarealen
Supplied parameter:
- Type: Integer
- Range: X'1040', X'2040', X'3040', X'4040'
- Length: Full word

Infoarealen specifies the length of the infoarea parameter.

,Infoarea
Returned parameter:
- Type: Character
- Length: X'1040', X'2040', X'3040', X'4040' bytes

Infoarea is to contain the retrieved system information. (Infoarealen specifies the length of the provided area.) The infoarea must be of the proper length to hold the requested information. This length depends on the value of the Request parameter.
- When the Request parameter is CSRSI_Request_V1CPC_Machine, the returned infoarea is mapped by SIV1 and the infoarealen parameter must be X'2040'.
- When the Request parameter is CSRSI_Request_V1CPC_Machine plus CSRSI_Request_V2CPC_LPAR, the returned infoarea is mapped by SIV1V2 and the infoarealen parameter must be X'3040'.
- When the Request parameter is CSRSI_Request_V1CPC_Machine plus CSRSI_Request_V2CPC_LPAR plus CSRSI_Request_V3CPC_VM, the returned infoarea is mapped by SIV1V2V3 and the infoarealen parameter must be X'4040'.
- When the Request parameter is CSRSI_Request_V1CPC_Machine plus CSRSI_Request_V3CPC_VM, the returned infoarea is mapped by SIV1V3 and the infoarealen parameter must be X'3040'.
- When the Request parameter is CSRSI_Request_V2CPC_LPAR, the returned infoarea is mapped by SIV2 and the infoarealen parameter must be X'1040'.
- When the Request parameter is CSRSI_Request_V2CPC_LPAR plus CSRSI_Request_V3CPC_VM, the returned infoarea is mapped by SIV2V3 and the infoarealen parameter must be X'2040'.
- When the Request parameter is CSRSI_Request_V3CPC_VM, the returned infoarea is mapped by SIV3 and the infoarealen parameter must be X'1040'.

,Returncode
Returned parameter:
- Type: Integer
- Length: Full word

Returncode contains the return code from the CSRSI service.

Return Codes

When the CSRSI service returns control to the caller, Returncode contains the return code. To obtain the equates for the return codes:

- If you are coding in assembler, include mapping macro CSRSIIDF, described in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.
- If you are coding in C, use include file CSRSIC. See Figure 13-1 on page 13-5.

The following table describes the return codes, shown in decimal.

Return Code and Equate Symbol	Meaning and Action
00 (0) CSRSI_SUCCESS	**Meaning**: The CSRSI service completed successfully. All information requested was returned. **Action**: Check the si00validityflags field to determine the validity of each returned area.
04 (4) CSRSI_STSINOTAVAILABLE	**Meaning**: The CSRSI service completed successfully, but since the Store System Information (STSI) instruction was not available, only the SI00PCCACPID, SI00PCCACPUA, and SI00PCCACAFM fields are valid. **Action**: None required.
08 (8) CSRSI_SERVICENOTAVAILABLE	**Meaning**: Environmental error: The CSRSI service is not available on this system. **Action**: Avoid calling the CSRSI service unless running on a system on which it is available.
12 (C) CSRSI_BADREQUEST	**Meaning**: User error: The request parameter did not specify a word formed from any combination of CSRSI_Request_V1CPC_Machine, CSRSI_Request_V2CPC_LPAR, and CSRSI_Request_V3CPC_VM. **Action**: Correct the parameter.
16 (10) CSRSI_BADINFOAREALEN	**Meaning**: User error: The Infoarealen parameter did not match the length of the area required to return the requested information. **Action**: Correct the parameter.
20 (14) CSRSI_BADLOCK	**Meaning**: User error: The service was called while holding a system lock other than CPU, LOCAL/CML, or CMS. **Action**: Avoid calling in this environment.

CSRSIC C/370 Header File

For the C programmer, include file CSRSIC provides equates for return codes and data constants, such as Register service request types. To use CSRSIC, copy the file from SYS1.SAMPLIB to the appropriate local C library. The contents of the file are displayed in Figure 13-1 on page 13-5.

```
#ifndef __CSRSI

#define __CSRSI

/*********************************************************************
 *         Type Definitions for User Specified Parameters            *
 *********************************************************************/

/*  Type for Request operand of CSRSI                                */
typedef int  CSRSIRequest;

/*  Type for InfoAreaLen operand of CSRSI                            */
typedef int  CSRSIInfoAreaLen;

/*  Type for Return Code                                             */
typedef int  CSRSIReturnCode;



/*********************************************************************
 *           Function Prototypes for Service Routines                *
 *********************************************************************/

#ifdef __cplusplus
   extern "OS" ??<
#else
  #pragma linkage(CSRSI_calltype,OS)
#endif
typedef void CSRSI_calltype(
   CSRSIRequest      __REQUEST,   /* Input  - request type           */
   CSRSIInfoAreaLen  __INFOAREALEN,  /* Input - length of infoarea   */
   void             *__INFOAREA,  /* Input  - info area              */
   CSRSIReturnCode  *__RC);       /* Output - return code            */

extern CSRSI_calltype csrsi;


#ifdef __cplusplus
   ??>
#endif

#ifndef __cplusplus
#define csrsi_byaddr(Request, Flen, Fptr, Rcptr)                    \
??<                                                                 \
 struct CSRSI_PSA* CSRSI_pagezero = 0;                              \
   CSRSI_pagezero->CSRSI_cvt->CSRSI_cvtcsrt->CSRSI_addr             \
           (Request,Flen,Fptr,Rcptr);                               \
??>;
#endif
  ??>;
struct CSRSI_CSRT ??<
   unsigned char CSRSI_csrt_filler1  ??(48??);
   CSRSI_calltype* CSRSI_addr;
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 1 of 14)

```
struct CSRSI_CVT ??<
   unsigned char CSRSI_cvt_filler1  ??(116??);
  struct ??<
    int CSRSI_cvtdcb_rsvd1 : 4;      /* Not needed                   */
    int CSRSI_cvtosext : 1;          /* If on, indicates that the
                  CVTOSLVL fields are valid                          */
    int CSRSI_cvtdcb_rsvd2 : 3;      /* Not needed                   */
         ??> CSRSI_cvtdcb;
   unsigned char CSRSI_cvt_filler2  ??(427??);
   struct CSRSI_CSRT * CSRSI_cvtcsrt;
   unsigned char CSRSI_cvt_filler3  ??(716??);
   unsigned char CSRSI_cvtoslv0;
   unsigned char CSRSI_cvtoslv1;
   unsigned char CSRSI_cvtoslv2;
   unsigned char CSRSI_cvtoslv3;
  struct ??<
    int CSRSI_cvtcsrsi : 1;          /* If on, indicates that the
                                        CSRSI service is available   */
    int CSRSI_cvtoslv1_rsvd1 : 7;    /* Not needed                   */
         ??> CSRSI_cvtoslv4;
   unsigned char CSRSI_cvt_filler4 ??(11??);        /*               */
??>;


struct CSRSI_PSA ??<
   char CSRSI_psa_filler??(16??);
   struct CSRSI_CVT* CSRSI_cvt;
??>;

/*  End of CSRSI Header                                               */

#endif

/*********************************************************************/
/* si11v1 represents the output for a V1 CPC when general CPC        */
/* information is requested                                          */
/*********************************************************************/

typedef struct ??<
  unsigned char  _filler1??(32??);  /* Reserved                      */
  unsigned char  si11v1cpcmanufacturer??(16??); /*
                                     The 16-character (0-9
                                     or uppercase A-Z) EBCDIC name
                                     of the manufacturer of the V1
                                     CPC. The name is
                                     left-justified with trailing
                                     blank characters if necessary.
                                                                   */
  unsigned char  si11v1cpctype??(4??); /* The 4-character (0-9) EBCDIC
                                     type identifier of the V1 CPC.
                                                                   */
  unsigned char  _filler2??(12??);  /* Reserved                      */
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 2 of 14)

```
  unsigned char  si11v1cpcmodel??(16??); /* The 16-character (0-9 or
                                        uppercase A-Z) EBCDIC model
                                        identifier of the V1 CPC. The
                                        identifier is left-justified
                                        with trailing blank characters
                                        if necessary.                */
  unsigned char  si11v1cpcsequencecode??(16??); /*
                                        The 16-character (0-9
                                        or uppercase A-Z) EBCDIC
                                        sequence code of the V1 CPC.
                                        The sequence code is
                                        right-justified with leading
                                        EBCDIC zeroes if necessary.
                                                                     */
  unsigned char  si11v1cpcplantofmanufacture??(4??); /* The 4-character
                                        (0-9 or uppercase A-Z) EBCDIC
                                        plant code that identifies the
                                        plant of manufacture for the
                                        V1 CPC. The plant code is
                                        left-justified with trailing
                                        blank characters if necessary.
                                                                     */
  unsigned char  _filler3??(3996??); /* Reserved                   */
??> si11v1;

/*********************************************************************/
/* si22v1 represents the output for a V1 CPC when information        */
/* is requested about the set of CPUs                                */
/*********************************************************************/

typedef struct ??<
  unsigned char  _filler1??(32??);  /* Reserved                     */
  unsigned char  si22v1cpucapability??(4??); /*
                                        An unsigned binary integer
                                        that specifies the capability
                                        of one of the CPUs contained
                                        in the V1 CPC. It is used as
                                        an indication of the
                                        capability of the CPU relative
                                        to the capability of other CPU
                                        models.                      */
  unsigned int   si22v1totalcpucount             : 16; /* A 2-byte
                                        unsigned integer
                                        that specifies the
                                        total number of CPUs contained
                                        in the V1 CPC. This number
                                        includes all CPUs in the
                                        configured state, the standby
                                        state, and the reserved state.
                                                                     */
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 3 of 14)

```
    unsigned int   si22v1configuredcpucount        : 16; /* A 2-byte
                                                   unsigned binary
                                                   integer that specifies
                                                   the total number of CPUs that
                                                   are in the configured state. A
                                                   CPU is in the configured state
                                                   when it is described in the
                                                   V1-CPC configuration
                                                   definition and is available to
                                                   be used to execute programs.
                                                                          */
    unsigned int   si22v1standbycpucount           : 16; /* A 2-byte
                                                   unsigned integer
                                                   that specifies the
                                                   total number of CPUs that are
                                                   in the standby state. A CPU is
                                                   in the standby state when it
                                                   is described in the V1-CPC
                                                   configuration definition, is
                                                   not available to be used to
                                                   execute programs, but can be
                                                   used to execute programs by
                                                   issuing instructions to place
                                                   it in the configured state.
                                                                          */
    unsigned int   si22v1reservedcpucount          : 16; /* A 2-byte
                                                   unsigned binary
                                                   integer that specifies
                                                   the total number of CPUs that
                                                   are in the reserved state. A
                                                   CPU is in the reserved state
                                                   when it is described in the
                                                   V1-CPC configuration
                                                   definition, is not available
                                                   to be used to execute
                                                   programs, and cannot be made
                                                   available to be used to
                                                   execute programs by issuing
                                                   instructions to place it in
                                                   the configured state, but it
                                                   may be possible to place it in
                                                   the standby or configured
                                                   state through manually
                                                   initiated actions             */
    struct ??<
      unsigned char  _si22v1mpcpucapaf??(2??); /* Each individual
                                                   adjustment factor.            */
      unsigned char  _filler2??(4050??);
    ??> si22v1mpcpucapafs;
  ??> si22v1;

  #define si22v1mpcpucapaf  si22v1mpcpucapafs._si22v1mpcpucapaf
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 4 of 14)

```
/*********************************************************************/
/* si22v2 represents the output for a V2 CPC when information        */
/* is requested about the set of CPUs                                */
/*********************************************************************/
              typedef struct ??<
  unsigned char  _filler1??(32??);  /* Reserved                     */
  unsigned int   si22v2cpcnumber                : 16; /* A 2-byte
                                     unsigned integer
                                     which is the number of
                                     this V2 CPC. This number
                                     distinguishes this V2 CPC from
                                     all other V2 CPCs provided by
                                     the same logical-partition
                                     hypervisor                   */
  unsigned char  _filler2;          /* Reserved                     */
  struct ??<
    unsigned int   _si22v2lcpudedicated           : 1; /*
                                     When one, indicates that
                                     one or more of the logical
                                     CPUs for this V2 CPC are
                                     provided using V1 CPUs that
                                     are dedicated to this V2 CPC
                                     and are not used to provide
                                     logical CPUs for any other V2
                                     CPCs. The number of logical
                                     CPUs that are provided using
                                     dedicated V1 CPUs is specified
                                     by the dedicated-LCPU-count
                                     value. When zero, bit 0
                                     indicates that none of the
                                     logical CPUs for this V2 CPC
                                     are provided using V1 CPUs
                                     that are dedicated to this V2
                                     CPC.                         */
    unsigned int   _si22v2lcpushared              : 1; /*
                                     When one, indicates that
                                     or more of the logical CPUs
                                     for this V2 CPC are provided
                                     using V1 CPUs that can be used
                                     to provide logical CPUs for
                                     other V2 CPCs. The number of
                                     logical CPUs that are provided
                                     using shared V1 CPUs is
                                     specified by the
                                     shared-LCPU-count value. When
                                     zero, it indicates that none
                                     of the logical CPUs for this
                                     V2 CPC are provided using
                                     shared V1 CPUs.              */
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 5 of 14)

```
  unsigned int   _si22v2lcpuulimit              : 1; /*
                                    Utilization limit. When one,
                                    indicates that the amount of
                                    use of the V1-CPC CPUs that
                                    are used to provide the
                                    logical CPUs for this V2 CPC
                                    is limited. When zero, it
                                    indicates that the amount of
                                    use of the V1-CPC CPUs that
                                    are used to provide the
                                    logical CPUs for this V2 CPC
                                    is unlimited.                */
  unsigned int   _filler3                       : 5; /* Reserved
                                                               */
??> si22v2lcpuc;                  /* Characteristics            */
unsigned int   si22v2totallcpucount           : 16; /*
                                    A 2-byte unsigned
                                    integer that specifies the
                                    total number of logical CPUs
                                    that are provided for this V2
                                    CPC. This number includes all
                                    of the logical CPUs that are
                                    in the configured state, the
                                    standby state, and the
                                    reserved state.              */
unsigned int   si22v2configuredlcpucount      : 16; /*
                                    A 2-byte unsigned
                                    binary integer that specifies
                                    the total number of logical
                                    CPUs for this V2 CPC that are
                                    in the configured state. A
                                    logical CPU is in the
                                    configured state when it is
                                    described in the V2-CPC
                                    configuration definition and
                                    is available to be used to
                                    execute programs.            */
unsigned int   si22v2standbylcpucount         : 16; /*
                                    A 2-byte unsigned
                                    binary integer that specifies
                                    the total number of logical
                                    CPUs that are in the standby
                                    state. A logical CPU is in the
                                    standby state when it is
                                    described in the V2-CPC
                                    configuration definition, is
                                    not available to be used to
                                    execute programs, but can be
                                    used to execute programs by
                                    issuing instructions to place
                                    it in the configured state.
                                                               */
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 6 of 14)

```
unsigned int   si22v2reservedlcpucount         : 16; /*
                                       A 2-byte unsigned
                                       binary integer that specifies
                                       the total number of logical
                                       CPUs that are in the reserved
                                       state. A logical CPU is in the
                                       reserved state when it is
                                       described in the V2-CPC
                                       configuration definition, is
                                       not available to be used to
                                       execute programs, and cannot
                                       be made available to be used
                                       to execute programs by issuing
                                       instructions to place it in
                                       the configured state, but it
                                       may be possible to place it in
                                       the standby or configured
                                       state through manually
                                       initiated actions             */
  unsigned char  si22v2cpcname??(16??); /*
                                       The 8-character EBCDIC name of
                                       this V2 CPC. The name is
                                       left-justified with trailing
                                       blank characters if necessary.
                                                                     */
  unsigned char  si22v2cpccapabilityaf??(4??); /* Capability Adjustment
                                       Factor (CAF). An unsigned
                                       binary integer of 1000 or
                                       less. The adjustment factor
                                       specifies the amount of the
                                       V1-CPC capability that is
                                       allowed to be used for this V2
                                       CPC by the logical-partition
                                       hypervisor. The fraction of
                                       V1-CPC capability is
                                       determined by dividing the CAF
                                       value by 1000.                */
  unsigned char  _filler4??(16??);  /* Reserved                     */
  unsigned int   si22v2dedicatedlcpucount        : 16; /*
                                       A 2-byte unsigned
                                       binary integer that specifies
                                       the number of configured-state
                                       logical CPUs for this V2 CPC
                                       that are provided using
                                       dedicated V1 CPUs. (See the
                                       description of bit
                                       si22v2lcpudedicated.)         */
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 7 of 14)

```
 unsigned int   si22v2sharedlcpucount          : 16; /*
                                     A 2-byte unsigned
                                     integer that specifies the
                                     number of configured-state
                                     logical CPUs for this V2 CPC
                                     that are provided using shared
                                     V1 CPUs. (See the description
                                     of bit si22v2lcpushared.)
                                                                    */
   unsigned char  _filler5??(4012??); /* Reserved                  */
    ??> si22v2;

#define si22v2lcpudedicated       si22v2lcpuc._si22v2lcpudedicated
#define si22v2lcpushared          si22v2lcpuc._si22v2lcpushared
#define si22v2lcpuulimit          si22v2lcpuc._si22v2lcpuulimit

/*********************************************************************/
/* si22v3db is a description block that comprises part of the        */
/* si22v3 data.                                                      */
/*********************************************************************/

typedef struct ??<
  unsigned char  _filler1??(4??);   /* Reserved                    */
  unsigned int   si22v3dbtotallcpucount           : 16; /*
                                     A 2-byte unsigned
                                     binary integer that specifies
                                     the total number of logical
                                     CPUs that are provided for
                                     this V3 CPC. This number
                                     includes all of the logical
                                     CPUs that are in the
                                     configured state, the standby
                                     state, and the reserved state.
                                                                    */
  unsigned int   si22v3dbconfiguredlcpucount      : 16; /*
                                     A 2-byte unsigned
                                     binary integer that specifies
                                     the number of logical CPUs for
                                     this V3 CPC that are in the
                                     configured state. A logical
                                     CPU is in the configured state
                                     when it is described in the
                                     V3-CPC configuration
                                     definition and is available to
                                     be used to execute programs.
                                                                    */
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 8 of 14)

```
unsigned int   si22v3dbstandbylcpucount          : 16; /*
                                        A 2-byte unsigned
                                        binary integer that specifies
                                        the number of logical CPUs for
                                        this V3 CPC that are in the
                                        standby state. A logical CPU
                                        is in the standby state when
                                        it is described in the V3-CPC
                                        configuration definition, is
                                        not available to be used to
                                        execute programs, but can be
                                        used to execute programs by
                                        issuing instructions to place
                                        it in the configured state.
                                                                    */
unsigned int   si22v3dbreservedlcpucount         : 16; /*
                                        A 2-byte unsigned
                                        binary integer that specifies
                                        the number of logical CPUs for
                                        this V3 CPC that are in the
                                        reserved state. A logical CPU
                                        is in the reserved state when
                                        it is described in the V2-CPC
                                        configuration definition, is
                                        not available to be used to
                                        execute programs, and cannot
                                        be made available to be used
                                        to execute programs by issuing
                                        instructions to place it in
                                        the configured state, but it
                                        may be possible to place it in
                                        the standby or configured
                                        state through manually
                                        initiated actions           */
 unsigned char  si22v3dbcpcname??(8??); /* The 8-character EBCDIC name
                                        of this V3 CPC. The name is
                                        left-justified with trailing
                                        blank characters if necessary.
                                                                    */
 unsigned char  si22v3dbcpccaf??(4??); /* A 4-byte unsigned binary
                                         integer that specifies an
                                         adjustment factor. The
                                         adjustment factor specifies
                                         the amount of the V1-CPC or
                                         V2-CPC capability that is
                                         allowed to be used for this V3
                                         CPC by the
                                         virtual-machine-hypervisor
                                         program.                   */
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 9 of 14)

```
 unsigned char  si22v3dbvmhpidentifier??(16??); /* The 16-character
                                     EBCDIC identifier of the
                                     virtual-machine-hypervisor
                                     program that provides this V3
                                     CPC. (This identifier may
                                     include qualifiers such as
                                     version number and release
                                     level). The identifier is
                                     left-justified with trailing
                                     blank characters if necessary.
                                                                    */
  unsigned char  _filler2??(24??);  /* Reserved                     */
??> si22v3db;
/*********************************************************************/
/* si22v3 represents the output for a V3 CPC when information        */
/* is requested about the set of CPUs                                */
/*********************************************************************/

typedef struct ??<
  unsigned char  _filler1??(28??);  /* Reserved                     */
  unsigned char  _filler2??(3??);   /* Reserved                     */
  struct ??<
      unsigned int   _filler3               : 4; /* Reserved
                                                                    */
      unsigned int   _si22v3dbcount         : 4; /*
                                     Description Block Count. A
                                     4-bit unsigned binary integer
                                     that indicates the number (up
                                     to 8) of V3-CPC description
                                     blocks that are stored in the
                                     si22v3dbe array.             */
  ??> si22v3dbcountfield;           /*                            */
  si22v3db  si22v3dbe??(8??);    /* Array of entries. Only the number
                                    indicated by si22v3dbcount
                                    are valid                     */
  unsigned char  _filler5??(3552??); /* Reserved                  */
??> si22v3;

#define si22v3dbcount      si22v3dbcountfield._si22v3dbcount


/*********************************************************************/
/* SI00 represents the "starter" information. This structure is      */
/* part of the information returned on every CSRSI request.          */
/*********************************************************************/
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 10 of 14)

```
typedef struct ??<
  char           si00cpcvariety;    /* SI00CPCVariety_V1CPC_MACHINE,
                                       SI00CPCVariety_V2CPC_LPAR, or
                                       SI00CPCVariety_V3CPC_VM       */
    struct ??<
              int   _si00validsi11v1  : 1; /* si11v1 was requested and
                                      the information returned is valid
                                                                     */
              int   _si00validsi22v1  : 1; /* si22v2 was requested and
                                      the information returned is valid
                                                                     */
              int   _si00validsi22v2  : 1; /* si22v2 was requested and
                                      the information returned is valid
                                                                     */
              int   _si00validsi22v3  : 1; /* si22v3 was requested and
                                      the information returned is valid
                                                                     */
              int   _filler1          : 4; /* Reserved              */
    ??> si00validityflags;
  unsigned char  _filler2??(2??);   /* Reserved                     */
  unsigned char  si00pccacpid??(12??); /* PCCACPID value for this CPU
                                                                     */
  unsigned char  si00pccacpua??(2??); /* PCCACPUA value for this CPU
                                                                     */
  unsigned char  si00pccacafm??(2??); /* PCCACAFM value for this CPU
                                                                     */
  unsigned char  _filler3??(4??);   /* Reserved                     */
  unsigned char  si00lastupdatetimestamp??(8??); /* Time of last STSI
                                      update, via STCK               */
  unsigned char  _filler4??(32??);  /* Reserved                     */
 ??> si00;

#define si00validsi11v1         si00validityflags._si00validsi11v1
#define si00validsi22v1         si00validityflags._si00validsi22v1
#define si00validsi22v2         si00validityflags._si00validsi22v2
#define si00validsi22v3         si00validityflags._si00validsi22v3

/*********************************************************************/
/* siv1 represents the information returned when V1CPC_MACHINE       */
/* data is requested                                                 */
/*********************************************************************/

typedef struct ??<
  si00 siv1si00;                                  /* Area mapped by
                                     struct si00                     */
  si11v1 siv1si11v1;                                    /* Area
                                     mapped by struct si11v1         */
  si22v1 siv1si22v1;                                    /* Area
                                     mapped by struct si22v1         */
??> siv1;
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 11 of 14)

```
/*********************************************************************/
/* siv1v2 represents the information returned when V1CPC_MACHINE     */
/* data and V2CPC_LPAR data is requested                             */
/*********************************************************************/

typedef struct ??<
  si00 siv1v2si00;                                  /* Area mapped by
                                        by struct si00               */
  si11v1 siv1v2si11v1;                                /* Area
                                        mapped by struct si11v1      */
  si22v1 siv1v2si22v1;                                /* Area
                                        mapped by struct si22v2      */
  si22v2 siv1v2si22v2;                                /* Area
                                        mapped by struct si22v2      */
??> siv1v2;

/*********************************************************************/
/* siv1v2v3 represents the information returned when V1CPC_MACHINE   */
/* data, V2CPC_LPAR data and V3CPC_VM data is requested              */
/*********************************************************************/

 typedef struct ??<
  si00 siv1v2v3si00;                                 /* Area
                                        mapped by struct si00        */
  si11v1 siv1v2v3si11v1;                                /* Area
                                        mapped by struct si11v1      */
  si22v1 siv1v2v3si22v1;                                /* Area
                                        mapped by struct si22v1      */
  si22v2 siv1v2v3si22v2;                                /* Area
                                        mapped by struct si22v2      */
  si22v3 siv1v2v3si22v3;                                /* Area
                                        mapped by struct si22v3      */
 ??> siv1v2v3;

/*********************************************************************/
/* siv1v3 represents the information returned when V1CPC_MACHINE     */
/* data and V3CPC_VM data is requested                               */
/*********************************************************************/

typedef struct ??<
  si00 siv1v3si00;                                  /* Area mapped
                                        by struct si00               */
  si11v1 siv1v3si11v1;                                /* Area
                                        mapped by struct si11v1      */
  si22v1 siv1v3si22v1;                                /* Area
                                        mapped by struct si22v1      */
  si22v3 siv1v3si22v3;                                /* Area
                                        mapped by struct si22v3      */
??> siv1v3;
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 12 of 14)

```
/*********************************************************************/
/* siv2 represents the information returned when V2CPC_LPAR          */
/* data is requested                                                 */
/*********************************************************************/

typedef struct ??<
  si00 siv2si00;                    /* Area mapped by
                                       struct si00                  */
  si22v2 siv2si22v2;                /* Area
                                       mapped by struct si22v2      */
??> siv2;

/*********************************************************************/
/* siv2v3 represents the information returned when V2CPC_LPAR        */
/* and V3CPC_VM data is requested                                    */
/*********************************************************************/

typedef struct ??<
  si00 siv2v3si00;                  /* Area mapped
                                       by struct si00               */
  si22v2 siv2v3si22v2;              /* Area
                                       mapped by struct si22v2      */
  si22v3 siv2v3si22v3;              /* Area
                                       mapped by struct si22v3      */
??> siv2v3;

/*********************************************************************/
/* siv3 represents the information returned when V3CPC_VM            */
/* data is requested                                                 */
/*********************************************************************/

typedef struct ??<
  si00 siv3si00;                    /* Area mapped by
                                       struct si00                  */
  si22v3 siv3si22v3;                /* Area
                                       mapped by struct si22v3      */
??> siv3;
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 13 of 14)

```
/*************************************************************/
 *       Fixed Service Parameter and Return Code Defines     *
 ************************************************************************/

/*  SI00 Constants                                                      */

#define SI00CPCVARIETY_V1CPC_MACHINE   1
#define SI00CPCVARIETY_V2CPC_LPAR      2
#define SI00CPCVARIETY_V3CPC_VM        3

/*  CSRSI Constants                                                     */

#define CSRSI_REQUEST_V1CPC_MACHINE    1
#define CSRSI_REQUEST_V2CPC_LPAR       2
#define CSRSI_REQUEST_V3CPC_VM         4

/*  CSRSI Return codes                                                  */

#define CSRSI_SUCCESS                  0
#define CSRSI_STSINOTAVAILABLE         4
#define CSRSI_SERVICENOTAVAILABLE      8
#define CSRSI_BADREQUEST               12
#define CSRSI_BADINFOAREALEN           16
#define CSRSI_BADLOCK                  20
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 14 of 14)

Part 6. Appendixes

Appendix A. General use C/C++ header files

Programming Interface information

C/C++ header files are shipped in z/OS V1R4 SYS1.SAMPLIB. These header files are analogous to traditional z/OS MVS mapping macros and are provided for general use. The following table lists the members and describes the interface. Descriptions of the data areas referenced can be found in *z/OS MVS Data Areas, Vol 1 (ABEP-DALT)*.

Member	Description
BLSCADPL	Describes same dara areas as assembler macro BLSABDPL. Depends on BLSCDESC.
BLSCADSY	Describes same data areas as assembler macro BLSADSY.
BLSCCBSP	Describes same data areas as assembler macro BLSACBSP. Depends on BLSCDESC.
BLSCDESC	Describes same data areas as assembler macros BLSRDATC, BLSRDATS, BLSRDATT, BLSRESSY, and BLSRSASY. Many of the other members require that this header file be included before they are included.
BLSCDRPX	Describes same data areas as assembler macro BLSRDRPX. Depends on BLSCDESC.
BLSCNAMP	Describes same data areas as assembler macro BLSRNAMP. Depends on BLSCDESC.
BLSCPCQE	Describes same data areas as assembler macro BLSRPCQE. Depends on BLSCDESC.
BLSCPPR2	Describes same data areas as assembler macro BLSUPPR2.
BLSCPWHS	Describes same data areas as assembler macro BLSRPWHS. Depends on BLSCDESC.
BLSCXMSP	Describes same data areas as assembler macro BLSRXMSP. Depends on BLSCDESC.
BLSCXSSP	Describes same data areas as assembler macro BLSRXSSP. Depends on BLSCDESC.

End of Programming Interface information

Appendix B. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen-readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen-readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using it to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Volume I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Notices

This information was developed for products and services offered in the USA.

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
USA

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 to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who 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 Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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 information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Programming Interface Information

This book is intended to help the customer to write applications that use operating system services. This book documents General-use Programming Interface and Associated Guidance Information provided by z/OS.

General-use programming interfaces allow the customer to write programs that obtain the services of z/OS.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

- C/370
- DB2

- Hiperspace
- IBM
- IBMLink
- IMS
- IMS/ESA
- MVS
- MVS/DFP
- MVS/SP
- OS/390
- Resource Link
- RETAIN
- z/OS
- z/OS.e

Other company, product, and service names may be trademarks or service marks of others.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Glossary

This glossary defines technical terms and abbreviations used in z/OS MVS documentation. If you do not find the term you are looking for, view IBM Glossary of Computing Terms, located at: http://www.ibm.com/ibm/terminology

D

data object. (1) A VSAM linear data set. (2) A storage area, outside the user's storage, that window services defines as a temporary object.

data-in-virtual. An MVS facility that enables a user to access a data object as though that data object resided in the user's storage.

G

gap. The grouping of consecutive bytes that the program repeatedly skips over. When a reference pattern has a gap, gaps and reference units alternate throughout the data area. See also *reference pattern* and *reference unit*.

H

hiperspace. A range of up to two gigabytes of virtual storage that a program can use like a buffer.

L

linear data set. A type of VSAM data set where data is stored as a linear string of bytes.

M

mapping. A process where window services makes a data object or part of a data object accessible to a user program through a scroll area or through a window.

O

object. See data object.

P

permanent data object. A virtual storage access method (VSAM) linear data set that resides on DASD (also called a data-in-virtual object).

R

reference pattern. The order in which a program's instructions process a data structure, such as an array. A reference pattern can be sequential or random and can contain gaps.

reference unit. A grouping of consecutive bytes that the program references. If the reference pattern has a gap, the reference unit is the grouping of bytes between gaps; gaps and reference units alternate throughout the data area. If the reference pattern does not have gaps, the reference unit is a logical grouping according to the structure of the data.

S

scroll area. An area of expanded storage that window services obtains. For a permanent object, window services maps a window to the scroll area and maps the scroll area to the permanent data object. You can use the scroll area to make interim changes to a permanent data object. For a temporary data object, the scroll area is the data object. Window services maps the window to the scroll area.

scrolling. A process where window services saves changes that a user has made in a window. For a permanent data object, window services saves the changes in the scroll area, without updating the permanent object. For a temporary object, window services updates the temporary object.

T

temporary data object. An area of expanded storage that window services provides for use by your program. You can use this storage to hold temporary data instead of using a DASD workfile. Window services provides no means for you to save a temporary data object.

V

VSAM. Virtual storage access method.

W

window. An area in the user's storage where the user can view or change data in a data object that window services has made available.

Index

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Readers' Comments — We'd Like to Hear from You

z/OS
MVS Programming: Callable Services for High-Level Languages

Publication No. SA22-7613-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

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.

Name _____ Address _____

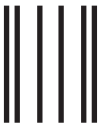
Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SA22-7613-01

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY
12601-5400

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Program Number: 5694-A01, 5655-G52

Printed in U.S.A.

SA22-7613-01