MVS Programming: Callable Services for High-Level Languages
MVS Programming: Callable Services for High-Level Languages
## Contents

<table>
<thead>
<tr>
<th>Figures</th>
<th>ix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>xi</td>
</tr>
</tbody>
</table>

### About this document
- Who should use this document: xiii
- How to use this document: xiii
- Where to find more information: xiii
- Information updates on the web: xiii
- The z/OS Basic Skills Information Center: xiii

### Summary of changes
- xv

## Part 1. Window Services

### Chapter 1. Introduction to Window Services
- Structure of a Data Object: 1-1
- What Does Window Services Provide?: 1-1
  - The Ways That Window Services Can Map an Object: 1-2
  - Access to Permanent Data Objects: 1-6
  - Access to Temporary Data Objects: 1-7

### Chapter 2. Using Window Services
- Obtaining Access to a Data Object: 2-2
  - Identifying the Object: 2-2
  - Specifying the Object’s Size: 2-3
  - Specifying the Type of Access: 2-3
  - Obtaining a Scroll Area: 2-3
- Defining a View of a Data Object: 2-4
  - Identifying the Data Object: 2-4
  - Identifying a Window: 2-4
  - Defining the Disposition of a Window’s Contents: 2-5
  - Defining the Expected Reference Pattern: 2-5
  - Identifying the Blocks You Want to View: 2-6
  - Extending the Size of a Data Object: 2-7
- Defining Multiple Views of an Object:
  - Nonoverlapping Views: 2-7
  - Overlapping Views: 2-7
- Saving Interim Changes to a Permanent Data Object: 2-8
- Updating a Temporary Data Object: 2-8
- Refreshing Changed Data: 2-9
- Updating a Permanent Object on DASD:
  - When There is a Scroll Area: 2-10
  - When There is No Scroll Area: 2-10
- Changing a View in a Window: 2-10
- Terminating Access to a Data Object: 2-12
- Handling Return Codes and Abnormal Terminations: 2-12

### Chapter 3. Window Services
- CSREVW — View an Object and Sequentially Access It: 3-1
  - Abend Codes: 3-3
  - Return Codes and Reason Codes: 3-3
- CSRIDAC — Request or Terminate Access to a Data Object: 3-5
### Chapter 9. Using the Latch Manager Services

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax and Linkage Conventions for Latch Manager Callable Services</td>
<td>9-1</td>
</tr>
<tr>
<td>ISGLCRT — Create a Latch Set</td>
<td>9-2</td>
</tr>
<tr>
<td>ABEND Codes.</td>
<td>9-4</td>
</tr>
<tr>
<td>Return Codes.</td>
<td>9-4</td>
</tr>
<tr>
<td>Examples of Calls to Latch Manager Services</td>
<td>9-4</td>
</tr>
<tr>
<td>ISGLOBT — Obtain a Latch</td>
<td>9-6</td>
</tr>
<tr>
<td>ABEND Codes.</td>
<td>9-8</td>
</tr>
<tr>
<td>Return Codes.</td>
<td>9-8</td>
</tr>
<tr>
<td>Example</td>
<td>9-9</td>
</tr>
<tr>
<td>ISGLREL — Release a Latch</td>
<td>9-9</td>
</tr>
<tr>
<td>ABEND Codes.</td>
<td>9-11</td>
</tr>
<tr>
<td>Return Codes.</td>
<td>9-11</td>
</tr>
<tr>
<td>Example</td>
<td>9-12</td>
</tr>
<tr>
<td>ISGLPRG — Purge a Requestor from a Latch Set</td>
<td>9-12</td>
</tr>
<tr>
<td>ABEND Codes.</td>
<td>9-13</td>
</tr>
<tr>
<td>Return Codes.</td>
<td>9-13</td>
</tr>
<tr>
<td>Example</td>
<td>9-13</td>
</tr>
<tr>
<td>ISGLPBA — Purge a Group of Requestors from a Group of Latch Sets</td>
<td>9-13</td>
</tr>
<tr>
<td>ABEND Codes.</td>
<td>9-15</td>
</tr>
<tr>
<td>Return Codes.</td>
<td>9-15</td>
</tr>
</tbody>
</table>

### Part 4. Resource Recovery Services (RRS)

<table>
<thead>
<tr>
<th>Chapter 10. Using Protected Resources</th>
<th>10-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Recovery Programs</td>
<td>10-1</td>
</tr>
<tr>
<td>Two-Phase Commit Protocol</td>
<td>10-2</td>
</tr>
<tr>
<td>Resource Recovery Process</td>
<td>10-2</td>
</tr>
<tr>
<td>Requesting Resource Protection and Recovery</td>
<td>10-4</td>
</tr>
<tr>
<td>Using Distributed Resource Recovery</td>
<td>10-5</td>
</tr>
<tr>
<td>Application_Backout_UR (SRRBACK)</td>
<td>10-5</td>
</tr>
<tr>
<td>Description</td>
<td>10-5</td>
</tr>
<tr>
<td>Application_Commit_UR (SRRCMIT)</td>
<td>10-9</td>
</tr>
<tr>
<td>Description</td>
<td>10-9</td>
</tr>
</tbody>
</table>

### Part 5. Other Callable Services

<table>
<thead>
<tr>
<th>Chapter 11. IEAAFFN — Assign Processor Affinity for Encryption or Decryption</th>
<th>11-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restrictions and Limitations</td>
<td>11-2</td>
</tr>
<tr>
<td>Requirements</td>
<td>11-2</td>
</tr>
<tr>
<td>Return Codes.</td>
<td>11-2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 12. CSRL16J — Transfer Control to Another Routine</th>
<th>12-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining the Entry Characteristics of the Target Routine</td>
<td>12-1</td>
</tr>
<tr>
<td>Freeing Dynamic Storage Associated with the Caller</td>
<td>12-2</td>
</tr>
<tr>
<td>Programming Requirements</td>
<td>12-2</td>
</tr>
<tr>
<td>Restrictions</td>
<td>12-5</td>
</tr>
<tr>
<td>Performance Implications</td>
<td>12-5</td>
</tr>
<tr>
<td>Syntax Diagram.</td>
<td>12-5</td>
</tr>
<tr>
<td>C/370 Syntax</td>
<td>12-5</td>
</tr>
<tr>
<td>PL/I Syntax</td>
<td>12-6</td>
</tr>
<tr>
<td>Parameters</td>
<td>12-6</td>
</tr>
<tr>
<td>Return Codes</td>
<td>12-6</td>
</tr>
<tr>
<td>Example</td>
<td>12-7</td>
</tr>
</tbody>
</table>
Chapter 13. CSRSI — System Information Service .......................... 13-1
  Description ......................................................... 13-1
  Environment ....................................................... 13-1
  Programming Requirements ....................................... 13-1
  Restrictions ....................................................... 13-1
  Input Register Information ...................................... 13-1
  Output Register Information ..................................... 13-2
  Syntax ............................................................. 13-2
  Parameters ......................................................... 13-2
  Return Codes .......................................................... 13-3
  CSRSIC C/370 Header File ........................................ 13-4

Part 6. Base Control Program Internal Interface (BCPii) Services

Chapter 14. Base Control Program Internal Interface (BCPii) .............. 14-1
  BCPii Setup and Installation ..................................... 14-1
    Setting up Connectivity to the Support Element ................ 14-2
    Setting up Authority to Use BCPii ............................... 14-4
    BCPii Configuration ............................................. 14-6
    Setting up Event Notification for BCPii USS Applications ....... 14-6
    BCPii Startup and Shutdown ...................................... 14-7
  BCPii Callable Services ........................................... 14-9
  Syntax, Linkage and Programming Considerations ..................... 14-9
    Calling formats ................................................ 14-9
    Assembler considerations ....................................... 14-9
    Linkage considerations ......................................... 14-10
  HWICMD — Issue a BCPii Hardware Management Command ............... 14-10
    Description .................................................... 14-10
  HWICONN — Establish a BCPii Connection ........................... 14-21
    Description .................................................... 14-21
  HWIDISC — Release a BCPii Connection ................................ 14-28
    Description .................................................... 14-28
  HWIEVENT — Register for BCPii Events ................................ 14-32
    Description .................................................... 14-33
  HWLILIST — Retrieve HMC and BCPii Configuration-Related Information 14-40
    Description .................................................... 14-41
  HWIQUERY — BCPii Retrieval of SE/HMC-Managed Objects Data ....... 14-49
    Description .................................................... 14-49
  HWISET — BCPii Set SE/HMC-Managed Objects Data .................... 14-69
    Description .................................................... 14-69
  HWIBeginEventDelivery — Begin Delivery of BCPii Event Notifications 14-90
    Description .................................................... 14-90
  HWIEndEventDelivery — End Delivery of BCPii Event Notifications .... 14-92
    Description .................................................... 14-92
  HWIManageEvents — Manage the List of BCPii Events ................. 14-94
    Description .................................................... 14-94
  HWIGetEvent — Retrieve Outstanding BCPii Event Notifications ....... 14-98
    Description .................................................... 14-98

Appendix A. BCPii Communication Error Reason Codes ..................... A-1

Appendix B. General use C/C++ header files .............................. B-1
Appendix C. Accessibility ............................................. C-1
Using assistive technologies ..................................... C-1
Keyboard navigation of the user interface. ................. C-1
z/OS information ...................................................... C-1

Notices ................................................................. D-1
Programming Interface Information ....................... D-3
Policy for unsupported hardware ............................. D-3
Trademarks ............................................................. D-3

Glossary ............................................................... E-1

Index ................................................................. X-1
## Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Structure of a Data Object</td>
<td>1-2</td>
</tr>
<tr>
<td>1-2</td>
<td>Mapping a Permanent Object That Has No Scroll Area</td>
<td>1-3</td>
</tr>
<tr>
<td>1-3</td>
<td>Mapping a Permanent Object That Has a Scroll Area</td>
<td>1-4</td>
</tr>
<tr>
<td>1-4</td>
<td>Mapping a Temporary Object</td>
<td>1-4</td>
</tr>
<tr>
<td>1-5</td>
<td>Mapping an Object to Multiple Windows</td>
<td>1-5</td>
</tr>
<tr>
<td>1-6</td>
<td>Mapping Multiple Objects</td>
<td>1-6</td>
</tr>
<tr>
<td>5-1</td>
<td>Illustration of a Reference Pattern with a Gap</td>
<td>5-4</td>
</tr>
<tr>
<td>6-1</td>
<td>Two Typical Reference Patterns</td>
<td>6-2</td>
</tr>
<tr>
<td>6-2</td>
<td>Illustration of Forward Direction of Reference</td>
<td>6-3</td>
</tr>
<tr>
<td>6-3</td>
<td>Illustration of Backward Direction of Reference</td>
<td>6-4</td>
</tr>
<tr>
<td>10-1</td>
<td>ATM Transaction</td>
<td>10-2</td>
</tr>
<tr>
<td>10-2</td>
<td>Two-Phase Commit Actions</td>
<td>10-3</td>
</tr>
<tr>
<td>10-3</td>
<td>Backout — Application Request</td>
<td>10-4</td>
</tr>
<tr>
<td>10-4</td>
<td>Backout — Resource Manager Votes NO</td>
<td>10-4</td>
</tr>
<tr>
<td>10-5</td>
<td>Transaction — Distributed Resource Recovery</td>
<td>10-5</td>
</tr>
<tr>
<td>12-1</td>
<td>CSRLJC declarations for the L16J parameter list for C/370</td>
<td>12-3</td>
</tr>
<tr>
<td>12-2</td>
<td>CSRLJPLI declarations for return codes for PL/I</td>
<td>12-5</td>
</tr>
<tr>
<td>13-1</td>
<td>CSRSIC from SYS1.SAMPLIB</td>
<td>13-5</td>
</tr>
</tbody>
</table>
### Tables

3-1. CSREVVW Return and Reason Codes ......................................................... 3-4
3-2. CSRIDAC Return and Reason Codes .......................................................... 3-8
3-3. CSRREFR Return and Reason Codes .......................................................... 3-11
3-4. CSRSAVE Return and Reason Codes ......................................................... 3-13
3-5. CSRSCOT Return and Reason Codes .......................................................... 3-15
3-6. CSRVWEE Return and Reason Codes ........................................................ 3-19
9-1. ISGLCRT Return Codes .............................................................................. 9-4
9-2. ISGLOBT Return Codes .............................................................................. 9-8
9-3. ISGLREL Return Codes .............................................................................. 9-11
9-4. ISGLPRG Return Codes ............................................................................. 9-13
9-5. ISGLPBA Return Codes ............................................................................. 9-15
11-1. IEAAFFN Return Codes ............................................................................ 11-2
12-1. CSRL16J Return Codes ............................................................................ 12-6
About this document

This document supports z/OS (5694).

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®, see the online document at:

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/ZDOCAPAR

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

The z/OS Basic Skills Information Center

The z/OS Basic Skills Information Center is a Web-based information resource intended to help users learn the basic concepts of z/OS, the operating system that runs most of the IBM® mainframe computers in use today. The Information Center is designed to introduce a new generation of Information Technology professionals to basic concepts and help them prepare for a career as a z/OS professional, such as a z/OS system programmer.

Specifically, the z/OS Basic Skills Information Center is intended to achieve the following objectives:

• Provide basic education and information about z/OS without charge
• Shorten the time it takes for people to become productive on the mainframe
• Make it easier for new people to learn z/OS.
To access the z/OS Basic Skills Information Center, open your Web browser to the following Web site, which is available to all users (no login required):

http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp
Summary of changes

Summary of changes
for SA22-7613-05
z/OS Version 1 Release 11

This document contains information previously presented in z/OS MVS Programming: Callable Services for High-Level Languages, SA22-7613-04, which supports z/OS Version 1 Release 10.

New information

The HWISET service can be called to change or set various CPC, image (LPAR), and activation profile attributes. See “HWISET — BCPii Set SE/HMC-Managed Objects Data” on page 14-69 for details.

Changed information

The HWICONN service is updated to provide the activation profile support. See “HWICONN — Establish a BCPii Connection” on page 14-21 for details.

The HWIDISC service is updated to provide the activation profile support. See “HWIDISC — Release a BCPii Connection” on page 14-28 for details.

The HWILIST service is updated to provide the activation profile support. See “HWILIST — Retrieve HMC and BCPii Configuration-Related Information” on page 14-40 for details.

The HWIQUERY service is updated to provide the activation profile support. See “HWIQUERY — BCPii Retrieval of SE/HMC-Managed Objects Data” on page 14-49 for details.

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.

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.

Summary of changes
for SA22-7613-04
z/OS Version 1 Release 10
as updated April 2009

This document contains information previously presented in z/OS MVS Programming: Callable Services for High-Level Languages, SA22-7613-03, which supports z/OS Version 1 Release 6.

New information

• You can use the Base Control Program internal interface (BCPii) to allow authorized applications to query, change, and perform basic operational
procedures against the installed System z® hardware base. See Chapter 14, “Base Control Program Internal Interface (BCPii),” on page 14-1. To use BCPii services, issue the following calls from high-level language programs:

- “HWICMD — Issue a BCPii Hardware Management Command” on page 14-10
- “HWICONN — Establish a BCPii Connection” on page 14-21
- “HWIDISC — Release a BCPii Connection” on page 14-28
- “HWIEVENT — Register for BCPii Events” on page 14-32
- “HWILIST — Retrieve HMC and BCPii Configuration-Related Information” on page 14-40
- “HWQUERY — BCPii Retrieval of SE/HMC-Managed Objects Data” on page 14-49
- “HWIBeginEventDelivery — Begin Delivery of BCPii Event Notifications” on page 14-90
- “HWIEndEventDelivery — End Delivery of BCPii Event Notifications” on page 14-92
- “HWIManageEvents — Manage the List of BCPii Events” on page 14-94
- “HWIGetEvent — Retrieve Outstanding BCPii Event Notifications” on page 14-98

- A new parameter, create_option, is added to the Latch_Create service to specify a fullword integer that has one of the correct values as indicated. See “ISGLCRT — Create a Latch Set” on page 9-2.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Summary of changes for SA22-7613-02
z/OS Version 1 Release 6

This document contains information previously presented in z/OS MVS Programming: Callable Services for High-Level Languages, SA22-7613-01, which supports z/OS Version 1 Release 4.

Deleted information
- ISGGRHS0, the Global Resource Serialization RNL hash search routine has been removed and replaced by the new ISGQUERY ReqInfo=RNLSearch service in z/OS MVS Programming: Assembler Services Reference IAR-XCT.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.
Part 1. Window Services

Chapter 1. Introduction to Window Services ........................................ 1-1
Structure of a Data Object ......................................................... 1-1
What Does Window Services Provide? ......................................... 1-2
The Ways That Window Services Can Map an Object ...................... 1-3
   Example 1 — Mapping a Permanent Object that has no Scroll Area ... 1-3
   Example 2 — Mapping a Permanent Object that has a Scroll Area ... 1-3
   Example 3 — Mapping a Temporary Object .................................. 1-4
   Example 4 — Mapping Multiple Windows to an Object .................. 1-4
   Example 5 — Mapping Multiple Objects ..................................... 1-5
Access to Permanent Data Objects ........................................... 1-6
Access to Temporary Data Objects ........................................... 1-7

Chapter 2. Using Window Services .............................................. 2-1
Obtaining Access to a Data Object ........................................... 2-2
   Identifying the Object ...................................................... 2-2
      Permanent Object ....................................................... 2-2
      Temporary Object ...................................................... 2-3
   Specifying the Object's Size ............................................. 2-3
   Specifying the Type of Access ........................................... 2-3
   Obtaining a Scroll Area .................................................... 2-3
   Defining a View of a Data Object ........................................ 2-4
   Identifying the Data Object .............................................. 2-4
   Identifying a Window ...................................................... 2-4
   Defining the Disposition of a Window's Contents ....................... 2-5
      Replace Option ............................................................ 2-5
      Retain Option ............................................................. 2-5
   Defining the Expected Reference Pattern ................................ 2-5
   Identifying the Blocks You Want to View ................................ 2-6
   Extending the Size of a Data Object ..................................... 2-7
   Defining Multiple Views of an Object ................................... 2-7
      Nonoverlapping Views .................................................. 2-7
      Overlapping Views ...................................................... 2-7
   Saving Interim Changes to a Permanent Data Object .................... 2-8
   Updating a Temporary Data Object ....................................... 2-8
   Refreshing Changed Data .................................................. 2-9
   Updating a Permanent Object on DASD ................................... 2-10
      When There is a Scroll Area .......................................... 2-10
      When There is No Scroll Area ........................................ 2-10
   Changing a View in a Window .............................................. 2-10
   Terminating Access to a Data Object .................................... 2-12
   Handling Return Codes and Abnormal Terminations ..................... 2-12

Chapter 3. Window Services ....................................................... 3-1
CSREVW — View an Object and Sequentially Access It ..................... 3-1
   Abend Codes ................................................................. 3-3
   Return Codes and Reason Codes ......................................... 3-3
CSRIDAC — Request or Terminate Access to a Data Object ................ 3-5
   Abend Codes ................................................................. 3-8
   Return Codes and Reason Codes ......................................... 3-8
CSRREFR — Refresh an Object ................................................. 3-9
   Abend Codes ................................................................ 3-10
   Return Codes and Reason Codes ......................................... 3-10
CSRSAVE — Save Changes Made to a Permanent Object ................... 3-11
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.

![Structure of a Data Object](image)

**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.

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.
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.

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.
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 1-5. Mapping an Object to Multiple Windows
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 CSRSRSCOT.

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” on page 2-4
- “Defining a View of a Data Object” on page 2-7
- “Defining Multiple Views of an Object” on page 2-8
- “Saving Interim Changes to a Permanent Data Object” on page 2-8
- “Updating a Temporary Data Object” on page 2-9
- “Refreshing Changed Data” on page 2-9
- “Updating a Permanent Object on DASD” on page 2-10
- “Changing a View in a Window” on page 2-10
- “Terminating Access to a Data Object” on page 2-12
- “Handling Return Codes and Abnormal Terminations” on page 2-12.

### 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 CSRSsave 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.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Span</th>
<th>Resulting view in the window</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>view the entire object</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>view the first block only</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Offset</th>
<th>Span</th>
<th>Resulting view in the window</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>view the second block through the last block</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>view the second block only</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>view the third and fourth blocks only</td>
</tr>
</tbody>
</table>
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 \textit{object\_id}, and a different value for \textit{window\_name}. Define each window's view by specifying values for \textit{offset} and \textit{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 \textit{object\_id}. The value supplied for \textit{object\_id} must be the same value CSRIDAC returned in \textit{object\_id} when you requested access to the object.

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

\begin{itemize}
  \item The value assigned to \textit{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.
  \item The value assigned to \textit{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.
\end{itemize}

Window services replaces each block within the range specified by \textit{offset} and \textit{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 the CSRSAVE 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 \textit{window\_name}. The value supplied for \textit{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 \textit{offset} and \textit{span}. The values you supply must be the same values you supplied for \textit{offset} and \textit{span} when you established the view.

To specify a disposition for the data you are currently viewing, supply a value for \textit{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 \textit{disposition}.
  - To discard the data that is currently in the window, supply a value of REPLACE for \textit{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 \textit{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 \textit{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 \textit{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 CSRS Cot.
- 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 value of 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” on page 3-1
- “CSRIDAC — Request or Terminate Access to a Data Object” on page 3-5
- “CSRREFR — Refresh an Object” on page 3-9
- “CSRSAVE — Save Changes Made to a Permanent Object” on page 3-11
- “CSRSCOT — Save Object Changes in a Scroll Area” on page 3-14
- “CSRVIEW — View an Object” on page 3-16

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 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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 (0)</td>
<td>00000000 (0)</td>
<td><strong>Meaning:</strong> The operation was successful.</td>
<td><strong>Action:</strong> Continue normal program execution.</td>
</tr>
<tr>
<td>00000004 (4)</td>
<td>xxxx0125 (293)</td>
<td><strong>Meaning:</strong> The operation was successful. The service could not retain all the data that was in the scroll area, however.</td>
<td><strong>Action:</strong> Notify your system programmer.</td>
</tr>
<tr>
<td>00000012 (18)</td>
<td>xxxx000A (10)</td>
<td><strong>Meaning:</strong> There is another service currently executing with the specified ID.</td>
<td><strong>Action:</strong> Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0017 (23)</td>
<td><strong>Meaning:</strong> An I/O error has occurred.</td>
<td><strong>Action:</strong> Notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx001A (26)</td>
<td><strong>Meaning:</strong> The specified range does not encompass any mapped area of the object.</td>
<td><strong>Action:</strong> 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.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx001C (28)</td>
<td><strong>Meaning:</strong> The object cannot be accessed at the current time.</td>
<td><strong>Action:</strong> Try running your program at a later time. If the problem persists, notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0040 (64)</td>
<td><strong>Meaning:</strong> 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.</td>
<td><strong>Action:</strong> Change the MAP range you have specified or request your system programmer to increase the installation’s data space limits.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0801 (2049)</td>
<td><strong>Meaning:</strong> System error — Insufficient storage available to build the necessary data-in-virtual control block structure.</td>
<td><strong>Action:</strong> Notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0802 (2050)</td>
<td><strong>Meaning:</strong> System error — I/O driver failure.</td>
<td><strong>Action:</strong> Notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0803 (2051)</td>
<td><strong>Meaning:</strong> System error — A necessary page table could not be read into real storage.</td>
<td><strong>Action:</strong> Notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0804 (2052)</td>
<td><strong>Meaning:</strong> System error — Catalog update failed.</td>
<td><strong>Action:</strong> Notify your system programmer.</td>
</tr>
</tbody>
</table>
Table 3-1. CSREVW Return and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 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. |
| 00000002C (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**

<table>
<thead>
<tr>
<th>operation_type</th>
<th>object_type</th>
<th>object_name</th>
<th>scroll_area</th>
<th>object_state</th>
<th>access_mode</th>
<th>object_size</th>
<th>object_id</th>
<th>high_offset</th>
<th>return_code</th>
<th>reason_code</th>
</tr>
</thead>
</table>

**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” on page 3-8.

`,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” on page 3-8.
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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 (0)</td>
<td>00000000 (0)</td>
<td><strong>Meaning</strong>: The operation was successful.</td>
<td><strong>Action</strong>: Continue normal program execution.</td>
</tr>
<tr>
<td>00000008 (8)</td>
<td>00000118 (280)</td>
<td><strong>Meaning</strong>: The system could not obtain enough storage to create a hiperspace for the temporary object or the scroll area. <strong>Note</strong>: 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.</td>
<td><strong>Action</strong>: 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.</td>
</tr>
<tr>
<td>00000008 (8)</td>
<td>00000119 (281)</td>
<td><strong>Meaning</strong>: The system could not delete or unidentify the temporary object or the scroll area.</td>
<td><strong>Action</strong>: Notify your system programmer.</td>
</tr>
<tr>
<td>00000008 (8)</td>
<td>0000011A (282)</td>
<td><strong>Meaning</strong>: The system was unable to create a new VSAM linear data set. DFP 3.1 must be running and SMS must be active.</td>
<td><strong>Action</strong>: Notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx000A (10)</td>
<td><strong>Meaning</strong>: Another service currently is executing with the specified ID.</td>
<td><strong>Action</strong>: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx001C (28)</td>
<td><strong>Meaning</strong>: The object cannot be accessed at the current time.</td>
<td><strong>Action</strong>: Try running your program at a later time. If the problem persists, notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0037 (55)</td>
<td><strong>Meaning</strong>: 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).</td>
<td><strong>Action</strong>: Notify your system programmer.</td>
</tr>
</tbody>
</table>
### Table 3-2. CSRIDAC Return and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 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. |
| 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 rrr is the return code from dynamic allocation. The value nnn is the two-byte reason code from dynamic allocation. See [z/OS MVS Programming: Authorized Assembler Services Guide](https://www.ibm.com) 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...
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 on page 3-11 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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 (0)</td>
<td>00000000 (0)</td>
<td>Meaning: The operation was successful. Action: Continue normal program execution.</td>
</tr>
<tr>
<td>00000008 (8)</td>
<td>00000152 (338)</td>
<td>Meaning: The system could not refresh all of the temporary object within the specified span. Action: Notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx000A (10)</td>
<td>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.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0017 (23)</td>
<td>Meaning: An I/O error has occurred. Action: Notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx001A (26)</td>
<td>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.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0801 (2049)</td>
<td>Meaning: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. Action: Notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0803 (2051)</td>
<td>Meaning: System error — A necessary page table could not be read into real storage. Action: Notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0805 (2053)</td>
<td>Meaning: System error — A system error of indeterminate origin has occurred. Action: Notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0808 (2056)</td>
<td>Meaning: System error — I/O from a previous request has not completed. Action: Notify your system programmer.</td>
</tr>
<tr>
<td>000002C (44)</td>
<td>00000004 (4)</td>
<td>Meaning: Window services have not been defined to your system or the link to the service failed. Action: Notify your system programmer.</td>
</tr>
</tbody>
</table>

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`.

<table>
<thead>
<tr>
<th>CALL CSRSAVE</th>
<th>(object_id, offset, span, new_hi_offset, return_code, reason_code)</th>
</tr>
</thead>
</table>

`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."

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."

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 (0)</td>
<td>00000000 (0)</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Continue normal program execution.</td>
</tr>
<tr>
<td>00000004 (4)</td>
<td>xxxx0807 (2055)</td>
<td>Meaning: Media damage may be present in allocated DASD space. The damage is beyond</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the currently saved portion of the object. The SAVE operation completed successfully.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Notify your system programmer.</td>
</tr>
<tr>
<td>00000008 (8)</td>
<td>xxxx0143 (323)</td>
<td>Meaning: You cannot use the SAVE service for a temporary object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Use the scrollout (CSRSCOT) service.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx000A (10)</td>
<td>Meaning: There is another service currently executing with the specified ID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0017 (23)</td>
<td>Meaning: An I/O error has occurred.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Notify your system programmer.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx001A (26)</td>
<td>Meaning: The specified range does not encompass any mapped area of the object.</td>
</tr>
</tbody>
</table>
|             |               | 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.
Table 3-4. CSRSAVE Return and Reason Codes  (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 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. |

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.

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

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>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 (0)</td>
<td>00000000 (0)</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td>Action: Continue normal program execution.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3-5. CSRSCOT Return and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000A (4)</td>
<td>xxxx0008 (2055)</td>
<td>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.</td>
<td>Notify your system programmer.</td>
</tr>
<tr>
<td>0000000A (10)</td>
<td>xxxx000A (10)</td>
<td>There is another service currently executing with the specified ID.</td>
<td>Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.</td>
</tr>
<tr>
<td>0000001A (23)</td>
<td>xxxx0017 (23)</td>
<td>An I/O error has occurred.</td>
<td>Notify your system programmer.</td>
</tr>
<tr>
<td>0000001A (26)</td>
<td>xxxx001A (26)</td>
<td>The specified range does not encompass any mapped area of the object.</td>
<td>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.</td>
</tr>
<tr>
<td>0000001A (2049)</td>
<td>xxxx0801 (2049)</td>
<td>System error — Insufficient storage available to build the necessary data-in-virtual control block structure.</td>
<td>Notify your system programmer.</td>
</tr>
<tr>
<td>0000001A (2050)</td>
<td>xxxx0802 (2050)</td>
<td>System error — I/O driver failure.</td>
<td>Notify your system programmer.</td>
</tr>
<tr>
<td>0000001A (2051)</td>
<td>xxxx0803 (2051)</td>
<td>System error — A necessary page table could not be read into real storage.</td>
<td>Notify your system programmer.</td>
</tr>
<tr>
<td>0000001A (2052)</td>
<td>xxxx0804 (2052)</td>
<td>System error — Catalog update failed.</td>
<td>Notify your system programmer.</td>
</tr>
<tr>
<td>0000001A (2054)</td>
<td>xxxx0806 (2054)</td>
<td>System error — I/O error.</td>
<td>Notify your system programmer.</td>
</tr>
<tr>
<td>0000001A (2056)</td>
<td>xxxx0808 (2056)</td>
<td>System error — I/O from a previous request has not completed.</td>
<td>Notify your system programmer.</td>
</tr>
<tr>
<td>0000001A (2054)</td>
<td>xxxx0804 (2054)</td>
<td>System error — Catalog update failed.</td>
<td>Notify your system programmer.</td>
</tr>
<tr>
<td>0000001A (2054)</td>
<td>xxxx0806 (2054)</td>
<td>System error — I/O error.</td>
<td>Notify your system programmer.</td>
</tr>
<tr>
<td>0000002C (44)</td>
<td>00000004 (4)</td>
<td>Window services have not been defined to your system or the link to the service failed.</td>
<td>Notify your system programmer.</td>
</tr>
</tbody>
</table>

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 unmapped 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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 (0)</td>
<td>00000000 (0)</td>
<td>Meaning: The operation was successful. Action: Continue normal program execution.</td>
</tr>
<tr>
<td>00000004 (4)</td>
<td>xxxx0125 (293)</td>
<td>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.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx000A (10)</td>
<td>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.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td>xxxx0017 (23)</td>
<td>Meaning: An I/O error has occurred. Action: Notify your system programmer.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Reason Code</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| 0000000C (12) | xxx001A (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) | xxx001C (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) | xxx0040 (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) | xxx0801 (2049) | **Meaning:** System error — Insufficient storage available to build the necessary data-in-virtual control block structure.  
**Action:** Notify your system programmer. |
| 0000000C (12) | xxx0802 (2050) | **Meaning:** System error — I/O driver failure.  
**Action:** Notify your system programmer. |
| 0000000C (12) | xxx0803 (2051) | **Meaning:** System error — A necessary page table could not be read into real storage.  
**Action:** Notify your system programmer. |
| 0000000C (12) | xxx0804 (2052) | **Meaning:** System error — Catalog update failed.  
**Action:** Notify your system programmer. |
| 0000000C (12) | xxx0806 (2054) | **Meaning:** System error — I/O error.  
**Action:** Notify your system programmer. |
| 0000000C (12) | xxx0808 (2056) | **Meaning:** System error — I/O from a previous request has not completed.  
**Action:** Notify your system programmer. |
| 000002C (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"
- "C/370 Example" on page 4-6
- "COBOL Example" on page 4-9
- "FORTRAN Example" on page 4-13
- "Pascal Example" on page 4-17
- "PL/I Example" on page 4-21

ADA Example

```ada
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
    Voffset2, -- Offset passed as parameter
    Vobjsz, -- Object size, as parameter
    Winisz, -- Window size, as parameter
    High_Offset, -- Size of object in pages
    New_Hi_Offset, -- New max size of the object
```

© Copyright IBM Corp. 1994, 2009
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
Coctype : Str5; -- Operation type BEGIN/END
Cacess : Str6; -- Access RANDOM/SEQ
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
  Wwinsiz := Int_To_32(Window_Size); -- Set window size in variable
  Vofset2 := Offset+Wwinsiz; -- Set offset to 40 for 2nd map
  Coptype := "BEGIN";
  Csptype := "DDNAME ";
  Cobname := "CSRDD1 ";
  Cscroll := "YES";
  Cobstate := "OLD";
  Caccess := "UPDATE";

  -- Set up access to the permanent object and Cobname,
  -- request a scroll area Cscroll, Cobstate, Caccess,
  -- CSRIDAC(Coptype,
  -- permanent access to the Csptype, Cobname, Caccess,
  -- Coptype, Cusage, Cdisp,
  -- 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
  CSRIDAC (Coptype,
          Csptype,
          Cobname,
          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
  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,
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, 
Object_Id, 
Voffset, 
Vwinsiz, 
Sp(Orig..Num_Sp_Elem), 
Cusage, 
Cdisp, 
Return_Code, 
Reason_Code);

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

CSRSCOT (Object_Id, 
Voffset, 
Vwinsiz, 
Return_Code, 
Reason_Code);

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

CSRVIEW (Coptype, 
Object_Id, 
Voffset, 
Vwinsiz, 
Sp(Orig..Num_Sp_Elem), 
Cusage, 
Cdisp, 
Return_Code, 
Reason_Code);

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

CSRVIEW (Coptype, 
Object_Id, 
Voffset, 
Vwinsiz, 
Sp(Orig..Num_Sp_Elem), 
Cusage, 
Cdisp, 
Return_Code, 
Reason_Code);

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

CSRSAVE (Object_Id, 
Voffset2, 
Vwinsiz, 
New_HI_Offset, 
Reason_Code);
Return_Code,
Reason_Code);

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

CSRVIEW (Coptype,
Object_Id, -- End the current view in
Voffset2,
Wwinsiz,
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
Wwinsiz,
Sp(Orig..Num_Sp_ELEM),
Cusage,
Cdisp,
Return_Code,
Reason_Code);

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

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

CSRVIEW (Coptype, -- End the view in window
Object_Id,
Voffset,
Wwinsiz,
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,
Cobname,
Cscroll,
Cobstate,
Caccess,
Wwinsiz,
Object_Id,
High_Offset,
Return_Code,
Reason_Code);
**ADA Example**

```assembly
crtimpan6;
//adiajob    job
//** jcl used to compile, link, and execute theada program crtpan6
//** that uses data window services
//**
//*jobparm   t=2,l=99
//adiaobj    exec pgm=ikjeft01,dynamnbr=133
//sysstpr    dd sysout=* 00055813
//sysstsin   dd *
//syslib     ds('sys1.csslib') shr 00056147
//exec 'hlq.sevgexe1(ada)' 'userid.dws.ada'' (mai cre' 00056251
/*
//adiajob    exec pgm=crtpan06,dynamnbr=133
//steplib    dd disp=shr,dsn=hlq.sevmod1 00100051
//csrodii    dd dsn=userid.ada.dwstest.data,disp=shr 00120051
//conout     dd sysout=*, dcn=(lrecl=133, recfm=f) 00130013
```

---

**C/370 Example**

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

```c
#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 ret, long reason, int linenumber);

int 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);
```

---

4-6  z/OS V1R11.0 MVS Callable Services for HLL
chk_code(return_code, reason_code, __LINE__); /* Define a window in a 4K region and initialize */ 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_type2, &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_id, &offset, &high_offset, 
 &return_code, &reason_code);
chk_code(return_code, reason_code, __LINE__);
/* Report the status of the test. */
if (errflag) {  
}
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 ret, long 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;
}
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 NWING 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 9(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,
      RETRNI,
      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
      RETRNI,
      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,
   RETRNI,
   REASON
* END THE VIEW IN THE WINDOW
   CALL "CSRVIEW" USING
   BY CONTENT
      "END",
   BY REFERENCE
      TOKEN,
      WINOFF,
      NWINPG,
      WINDOW,
   BY CONTENT
"RANDOM",
"RETAI N",
BY REFERENCE
RETN1,
REASON
* NOW VIEW SOMETHING ELSE (2ND WINDOW'S WORTH OF DATA) IN WINDOW
ADD NWINPG TO WINOFF
CALL "CSRVIE W" USING
BY CONTENT
"BEGIN",
BY REFERENCE
TOKEN,
WINOFF
NWINPG,
WINDOW,
BY CONTENT
"RANDOM",
"RETAI N",
BY REFERENCE
RETN1,
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 "CSRSCT" USING
TOKEN,
WINOFF,
NWINPG,
RETN1,
REASON
* NOW END THE CURRENT VIEW IN WINDOW
CALL "CSRVIEW" USING
BY CONTENT
"END",
BY REFERENCE
TOKEN,
WINOFF
NWINPG,
WINDOW,
BY CONTENT
"RANDOM",
"RETAI N",
BY REFERENCE
RETN1,
REASON
* NOW GO BACK TO THE FIRST VIEW IN THE WINDOW
MOVE 0 TO WINOFF
CALL "CSRVIE W" USING
BY CONTENT
"BEGIN",
BY REFERENCE
TOKEN,
WINOFF
NWINPG,
WINDOW,
BY CONTENT
"RANDOM",
"REPLACE",
BY REFERENCE
RETN1,
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',
  'A.A.USER',RD=R, 00010000
// MSGCLASS='H',NOTIFY='AAUSER' , 00020000
// MSGLEVEL=(1,1),CLASS=7 00030000
//LKED EXEC PGM=IEWL,PARM='SIZE=(1024K,512K),LIST,XREF,LET,MAP', 00040000
// REGION=1024K 00050000
//SYSIN DD DDNAME=SYSIN 00060000
//SYSMOD DD DSNAME=AAUSER.USER_LOAD(CRTCON01),DISP=SHR 00070000
//SYSLIB DD DSNAME=CEE.SCEELED,DISP=SHR 00080000
//* 00090000
//* FF310.OBJ HOLDS OBJECT CODE FROM THE COMPIL 00100000
//* 00110000
//* THE CSR STUBS ARE IN SYS1.CSSLIB 00120000
//* 00130000
//* INLIB DD DSN=SYS1.CSSLIB,DISP=SHR 00140000
//* 00150000
//* 00160000
//* SYSSPRINT DD SYSOUT=* 00170000
//* 00180000
#include MYLIB(DWSCBSAM,DWSCB4K,DWSCB2)
library INLIB(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC)
name CRTCON01(R) 00200000
00210000
00220000
00230000
00240000
00250000

4-12  z/OS V1R11.0 MVS Callable Services for HLL
**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 Calculate a value in the window area
DO 101 K = 1, NWINEL
   A(K+PAGE-1) = K
101 CONTINUE

C Capture the view in the window
CALL CSRSCOT( TOKEN,
   * WINOFF,
   * NWINPG,
   * RETRN1,
   * REASON )

C End the view in the window
CALL CSRVIEW('END ',
   * TOKEN,
   * WINOFF,
   * NWINPG,
   * A(PAGE),
   * 'RANDOM',
   * 'RETAI N',
   * RETRN1,
   * REASON )

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 Calculate a new value in the window
DO 102 K = 1, NWINEL
   A(K+PAGE-1) = -K
102 CONTINUE

C Capture the view in the window
CALL CSRSCOT( TOKEN,
   * WINOFF + NWINPG,
   * NWINPG,
   * RETRN1,
   * REASON )

C Now end the current view in window
CALL CSRVIEW('END ',
   * TOKEN,
   * WINOFF + NWINPG,
   * NWINPG,
   * A(PAGE),
   * 'RANDOM',
   * 'RETAI N',
   * RETRN1,
   * REASON )

C Now go back to the first view in the window
CALL CSRVIEW('BEGIN',
   * TOKEN,
C Refresh the data in the window for this view
CALL CSRREFR( TOKEN,
*   WINOFF,
*   NWINPG,
*   RETRNI,
*   REASON )

C Now end the view in the window
CALL CSRVIEW('END ',
*   TOKEN,
*   WINOFF,
*   NWINPG,
*   A(PAGE),
*   'RANDOM',
*   'REPLACE',
*   'RETAIL ',
*   RETRNI,
*   REASON )

C Terminate access to the Hiperspace object
CALL CSRIDAC('END ',
*   'TEMSPACE',
*   'MY FIRST HIPERSPACE ENDS HERE ',
*   'YES',
*   'NEW',
*   'UPDATE',
*   NOBJPG,
*   TOKEN,
*   OBSIZ,
*   RETRNI,
*   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 COMPIL AND LINKEDIT THE ASSEMBLER PROGRAM, THE          *
*   FORTRAN PROGRAM, AND THE STUBS.                               *
************************************************************************

//FORTJOB    JOB
00003100
//       * Compile and linkedit for FORTRAN
00003100
//       * FORTRAN 00003100
//       * VSF2CL PROC FVPGM=FORTVS2,FVREGN=2100K,FVPDECK=NODECK,
00001000
//       // FVPOLST=NOLIST,FVPOPT=0,FVTERM='SYSOUT=A',
00002000
//       // PGMNAME=MAIN,PGLIB='&GOSET',FVLNSPC='3200,(25,6)'        00003000
//       *                  00003100
//       *                  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.SYSTEM OPERAND
00100000
//       FVLNSPC 3200,(25,6) FORT.SYSLIN SPACE
00110000
//       PGLIB &GOSET LKED.SYSLMOD MEMBER NAME
00120000
//       PGMNAME MAIN LKED.SYSLMOD MEMBER NAME
00130000
//                 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
//       // SYSTEM 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.SCEELKD,DISP=SHR                 00026000
//       // SYSU1 DD UNIT=SYSDA,SPACE=(1024,(200,20))         00027000
//       // SYSLMOD DD DSN=&PGLIB.(&PGMNAME),DISP=(,PASS),UNIT=SYSDA,
00028000
//            SPACE=(TRK,(10,10,1),RLSE)                  00029000
//       // SYSLIN DD DSN=&LOADSET,DISP=(OLD,DELETE)        00030000
//            DD DDNAME=SYSIN                                00031000
//            // DD DDNAME=SYSIN                            00032000
//       // PEND
//       EXEC VSF2CL,FVTERM='SYSOUT=H',
00003000
//          PGMNAME=CRTFON01,PGLIB='WINDOW.USER.LOAD'
//       // FORT.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTFON01),DISP=SHR 00003000
//       // LKED.SYSLIB DD DSN=CEE.SCEELKD,DISP=SHR              00003000
//       // LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,DISP=SHR,UNIT=3380,
00003000
//            VOL=SER=VM2TSO                                00003000
//       // 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
00030000
//            DD DSN=WINDOW.TEST.OBJ,DISP=SHR
//            //
* JCL TO EXECUTE THE FORTRAN PROGRAM. *
* **************************************************
//FON01 JOB MSGLEVEL=(1,1)
//VSF2G PROC GOPGM=MAIN,GOREGN=100K,
// GOPGM5DD='DDNAME=SYSIN',
// GOPF6DD='SYSOUT=A',
// GOPF7DD='SYSOUT=B'
// PARAMETER DEFAULT-VALUE USAGE
// GOPGM MAIN PROGRAM NAME
// GOREGN 100K GO-STEP REGION
//&GOF5DD DDNAME=SYSIN
//&GOF6DD SYSOUT=A
//&GOF7DD SYSOUT=B
GO EXEC PGM=&GOPGM,REGION=&GOREGN,COND=(4,LT)
STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
&GOF5DD DD
&GOF6DD DD
&GOF7DD DD
//PEND
GO EXEC VSF2G,GOPGM=CRTFON01,GOREGN=999K
//STEP LIB DD DSN=CEE.SCEERUN,DISP=SHR
// 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 = (WINDOW_SIZE+1)*4*K; (* 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 *)
VOFSET, (* Offset passed as parameter *)
VOFSET2, (* Offset passed as parameter *)
VOBJSIZ, (* Object size, as parameter *)
WIN SIZ, (* Window Size, as parameter *)
HIGH OFFSET, (* Size of object in pages *)
NEW_HI_OFFSET, (* New max size of the object *)

Chapter 4. Window Services Coding Examples  4-17
**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 *)
USAGE : STR6;  (* Usage READ/UPDATE *)
DISP : STR7;  (* Disposition RETAIN/REPLACE *)
CSPTYPE : STR9;  (* Object type DSNAME/DDNAME/TEMPSPACE *)
COBNAME : STR44;  (* Object name *)

procedure CSRIDAC ( var OP_TYPE : STR5;
  var SCROLL_AREA : STR6;
  var OBJECT_TYPE : STR9;
  var OBJECT_STATE : STR3;
  var ACCESS_MODE : STR6;
  var OBJECT_ID : REAL;
  var RETURN_CODE : INTEGER;
  var REASON_CODE : INTEGER);

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 );

procedure CSRSCOT ( var OBJECT_ID : REAL;
  var OFFSET : INTEGER;
  var SPAN : INTEGER;
  var RETURN_CODE : INTEGER;
  var REASON_CODE : INTEGER );

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 );

procedure CSRREFR ( var OBJECT_ID : REAL;
  var OFFSET : INTEGER;
  var SPAN : INTEGER;
  var RETURN_CODE : INTEGER;
  var REASON_CODE : INTEGER );

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 *)
  VOFFSET2 := OFFSET+WINDOW_SIZE;  (* Set offset variable to 0 *)
  COPTYPE := 'BEGIN';
  CSPTYPE := 'DDNAME';
  COBNAME := 'CSRDD1';
  CSCROLL := 'YES';
  COBSTATE := 'NEW';
  CACCESS := 'UPDATE';
  CSRIDAC (COPTYPE,
    CSPTYPE,
    COBNAME,
    CSCROLL,
    COBSTATE,
    CACCESS,
    RETURN_CODE,
    REASON_CODE,
    OBJECT_ID,
    CSCROLL,
    COBSTATE,
    COPTYPE,
    CSPTYPE,
    OBJECT_NAME,
    SCROLL_AREA,
    OBJECT_TYPE,
    OBJECT_NAME,
    OBJECT_ID,
    HIGH_OFFSET,
    RETURN_CODE,
    REASON_CODE;
    FORTRAN;
```
VOBJSIZ,
OBJECT_ID,
HIGH_OFFSET,
RETURN_CODE,
REASON_CODE);

NEW(SP); (* Allocate space *)
AD := ADDR(SP); (* or ORD(SP) *)
ORIG := AD mod PAGESIZE; (* Get address of space *)
if ORIG <> 0 then (* See where space is in page *)
ORIG := PAGESIZE-ORIG; (* If not on page boundary *)
for I := 0 to NUM_WIN_ELEM-1 do (* then locate page boundary *)
SP[4*I+ORIG] := 999999; (* Put data into window *)
COPTYPE := 'BEGIN';
CUSAGE := 'RANDOM';
CDISP := 'REPLACE';
CSRVIEW (COPTYPE,
OBJECT_ID,
VOFFSET,
VWINSIZ,
SP[ORIG],
CUSAGE,
CDISP,
RETURN_CODE,
REASON_CODE);

for I := 0 to NUM_WIN_ELEM-1 do (* Now view data in 1st *)
SP[4*I+ORIG] := I+I; (* window *)
CSRCSCOT( OBJECT_ID,
VOFFSET,
VWINSIZ,
RETURN_CODE,
REASON_CODE);

COPTYPE := 'END';
CUSAGE := 'RANDOM';
CDISP := 'RETAIN';
CSRVIEW (COPTYPE,
OBJECT_ID,
VOFFSET,
VWINSIZ,
RETURN_CODE,
REASON_CODE);

for I := 0 to NUM_WIN_ELEM-1 do (* End the view in 1st window *)
SP[4*I+ORIG] := I-101; (* window *)
CSRSAVE (OBJECT_ID,
VOFSET2,
VWINSIZ,
NEW_HI_OFFSET,
RETURN_CODE,
REASON_CODE);

COPTYPE := 'END';
CUSAGE := 'RANDOM';
CDISP := 'RETAIN';
CSRVIEW (COPTYPE,
OBJECT_ID,
VOFSET2,
VWINSIZ,
NEW_HI_OFFSET,
RETURN_CODE,
REASON_CODE);

for I := 0 to NUM_WIN_ELEM-1 do (* Now view other data in the *)
SP[4*I+ORIG] := I-101; (* 2nd window *)
CSRSAVE (OBJECT_ID,
VOFSET2,
VWINSIZ,
NEW_HI_OFFSET,
RETURN_CODE,
REASON_CODE);

COPTYPE := 'END';
CUSAGE := 'RANDOM';
CDISP := 'RETAIN';
CSRVIEW (COPTYPE,
OBJECT_ID,
VOFSET2,
VWINSIZ,
NEW_HI_OFFSET,
RETURN_CODE,
REASON_CODE);

for I := 0 to NUM_WIN_ELEM-1 do (* End the current view 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,
Pascal Example

```pascal
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 *)
VOFSET,
VWINSIZ,
SP@[ORIG],
CUSAGE,
CDISP,
RETURN_CODE,
REASON_CODE);
CSRREFR (OBJECT_ID, (* Refresh the data in 1st *)
VOFSET,
VWINSIZ,
RETURN_CODE,
REASON_CODE);
COPTYPE := 'END';
CUSAGE := 'RANDOM';
CDISP := 'REtain';
CSRVIEW (COPTYPE, (* End the view in 1st window *)
OBJECT_ID,
VOFSET,
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 00060000
/LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,DISP=SHR,UNIT=3380, 00560000
```

4-20  z/OS V1R11.0 MVS Callable Services for HLL
PL/I Example

******************************************************************************
* PL/I EXAMPLE
* OBJECT IS TEMPORARY
* *
******************************************************************************

CRTPLN3: PROCEDURE OPTIONS (MAIN);

DCL
  (CSR00010
    INIT(1024), /* ONE KILO-BYTE */ CSR0050
    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);

DCL /* 32767 IS UPPER LIMIT FOR ARRAY Bound. */ CSR00140
  S(32767) BIN(31) FIXED BASED(SP); /* DEFINE WORD ALIGNED SPACE */ CSR00150

DCL
  (CSR00170
    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

DCL CSR00190
  (CSR00190
    OBJECT_ID CHAR(8); /* IDENTIFYING TOKEN */ CSR00290

Chapter 4. Window Services Coding Examples 4-21
PL/I Example

******************************************************************************

DCL CSRIDAC ENTRY(CHAR(5), /* OP_TYPE */ CSR00310
     CHAR(9), /* OBJECT_TYPE */ CSR00330
     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
     OPTIONS(ASSEMBLER);

DCL CSRVIEW ENTRY(CHAR(5), /* OP_TYPE */ CSR00440
     CHAR(8), /* OBJECT_ID */ CSR00480
     FIXED BIN(31), /* OFFSET */ CSR00510
     FIXED BIN(31), /* WINDOW_SIZE */ CSR00540
     CHAR(6), /* USAGE */ CSR00570
     CHAR(7), /* DISPOSITION */ CSR00590
     FIXED BIN(31), /* RETURN_CODE */ CSR00620
     OPTIONS(ASSEMBLER);

DCL CSRSCOT ENTRY(CHAR(8), /* OBJECT_ID */ CSR00690
     FIXED BIN(31), /* OFFSET */ CSR00720
     FIXED BIN(31), /* SPAN */ CSR00750
     FIXED BIN(31), /* RETURN_CODE */ CSR00780
     OPTIONS(ASSEMBLER);

DCL CSRSAVE ENTRY(CHAR(8), /* OBJECT_ID */ CSR00810
     FIXED BIN(31), /* OFFSET */ CSR00840
     FIXED BIN(31), /* SPAN */ CSR00860
     FIXED BIN(31), /* NEW_HI_OFFSET */ CSR00890
     FIXED BIN(31), /* RETURN_CODE */ CSR00920
     OPTIONS(ASSEMBLER);

DCL CSRRERF ENTRY(CHAR(8), /* OBJECT_ID */ CSR00930
     FIXED BIN(31), /* OFFSET */ CSR00960
     FIXED BIN(31), /* SPAN */ CSR00980
     FIXED BIN(31), /* RETURN_CODE */ CSR00990
     OPTIONS(ASSEMBLER);

******************************************************************************

PUT SKIP LIST
('<< BEGIN DATA WINDOWING SERVICES INTERFACE VALIDATION >>');

CALL CSRIDAC ('BEGIN', /* SET UP ACCESS TO A HIPER- */ CSR00830
   'TEMPSPACE', /* SPACE OBJECT */ CSR00840
   'MY FIRST HIPERSPACE', 'YES', 'NEW', 'UPDATE', /* OBJECT_SIZE, */ CSR00850
   CSR00860
   'YES', /* OBJECT_NAME */ CSR00870
   'TEMPSPACE', /* SPACE OBJECT */ CSR00880
   'YES', /* OBJECT_NAME */ CSR00890
   'MY FIRST HIPERSPACE', /* SPACE OBJECT */ CSR00900
   'YES', /* OBJECT_NAME */ CSR00910
   'TEMPSPACE', /* SPACE OBJECT */ CSR00920
   'YES', /* OBJECT_NAME */ CSR00930
   'NEW', /* OBJECT_NAME */ CSR00940
   'UPDATE', /* OBJECT_NAME */ CSR00950
   'YES', /* OBJECT_NAME */ CSR00960
   'MY FIRST HIPERSPACE', /* SPACE OBJECT */ CSR00970

4-22 z/OS V1R11.0 MVS Callable Services for HLL
ALLOC S; /* ALLOCATE SPACE */
AD = UNSPEC(SP); /* GET ADDRESS OF SPACE */
ORIG = MOD(AD,PAGESIZE); /* SEE WHERE SPACE IS IN PAGE */
IF ORIG = 0 THEN /* IF NOT ON PAGE BOUNDARY */
  ORIG = (PAGESIZE-ORIG) / 4;
  /* THEN LOCATE PAGE BOUNDARY */
ORIG = ORIG + 1;
DO I = 1 TO NUM_WIN_ELEM; /* PUT SOME DATA INTO WINDOW */
  S(I+ORIG-1) = 99; /* AREA */
END;
CALL CSRVIEW ('BEGIN', /* NOW VIEW DATA IN FIRST */
  OBJECT_ID, /* WINDOW */
  OFFSET,
  WINDOW_SIZE,
  S(ORIG),
  'RANDOM',
  'REPLACE',
  RETURN_CODE,
  REASON_CODE);
DO I = 1 TO NUM_WIN_ELEM; /* CALCULATE VALUE IN 1ST */
  S(I+ORIG-1) = I+1; /* WINDOW */
END;
CALL CSRSCOT( OBJECT_ID, /* CAPTURE THE VIEW IN 1ST */
  OFFSET, /* WINDOW */
  WINDOW_SIZE,
  RETURN_CODE,
  REASON_CODE);
CALL CSRVIEW ('END ', /* END THE VIEW IN 1ST WINDOW */
  OBJECT_ID,
  OFFSET,
  WINDOW_SIZE,
  S(ORIG),
  'RANDOM',
  'RETAII ',
  RETURN_CODE,
  REASON_CODE);
CALL CSRVIEW ('BEGIN', /* NOW VIEW OTHER DATA IN */
  OBJECT_ID, /* 2ND WINDOW */
  OFFSET+WINDOW_SIZE,
  WINDOW_SIZE,
  S(ORIG),
  'RANDOM',
  'REPLACE',
  RETURN_CODE,
  REASON_CODE);
DO I = 1 TO NUM_WIN_ELEM; /* CALCULATE NEW VALUE IN */
  S(I+ORIG-1) = I-101; /* WINDOW */
END;
CALL CSRSCOT (OBJECT_ID, /* CAPTURE THE VIEW IN 2ND */
  OFFSET+WINDOW_SIZE,
CALL CSRVIEW ('END ', /* END THE CURRENT VIEW IN */ CSR01710
  OBJECT_ID, /* WINDOW */ CSR01720
  OFFSET+WINDOW_SIZE,
  WINDOW_SIZE,
  S(ORIG),
  'RANDOM',
  'RETAIN ',
  RETURN Code,
  REASON Code);

CALL CSRVIEW ('BEGIN', /* NOW GO BACK TO THE VIEW IN */ CSR01820
  OBJECT_ID, /* THE 1ST WINDOW */ CSR01830
  OFFSET,
  WINDOW_SIZE,
  S(ORIG),
  'RANDOM',
  'REPLACE',
  RETURN Code,
  REASON Code);

CALL CSRREFR (OBJECT_ID, /* REFRESH THE DATA IN 1ST */ CSR01930
  OFFSET, /* WINDOW */ CSR01940
  WINDOW_SIZE,
  RETURN Code,
  REASON Code);

CALL CSRVIEW ('END ', /* END THE VIEW IN 1ST WINDOW */ CSR02000
  OBJECT_ID,
  OFFSET,
  WINDOW_SIZE,
  S(ORIG),
  'RANDOM',
  'RETAIN ',
  RETURN Code,
  REASON Code);

CALL CSRIDAC ('END ', /* TERMINATE ACCESS TO THE */ CSR02110
  'TEMPSPACE', /* HIPERSPACE OBJECT */ CSR02120
  'MY FIRST HIPERSPACE ENDS HERE ',
  'YES',
  'NEW',
  'UPDATE',
  WINDOW_SIZE,
  OBJECT_ID, CSR02170
  HIGH_OFFSET,
  RETURN Code,
  REASON Code);

FREE S;
END CRTPLN3;

***********************************************************************
*                                                                    *
* JCL TO COMPILE AND LINKEDIT PL/I PROGRAM.                          *
*                                                                    *
***********************************************************************

4-24  z/OS V1R11.0 MVS Callable Services for HLL
/* PL/I Compile and Linkedit */
/* Change all CRTPLNx to CRTPLNy */
GO EXEC PLIXCL
PL/IXCL DD DSN=WINDOW.XAMPLE.LIB(CRTPLN3),DISP=SHR
LKD.MOD DD DSN=WINDOW.USER.LOAD,UNIT=3380,VOL=SER=VM2T5O,
    DISP=SHR
LKD.SYSLMOD DD DSN=WINDOW.USER.LOAD,UNIT=3380,VOL=SER=VM2T5O,
    DISP=SHR
LKD.IN DD DSN=SYS1.CSSLIB,DISP=SHR
*/

EXECUTE A PL/I TESTCASE
GO EXEC PGM=CRTPLN3
STEPLIB DD DSN=WINDOW.USER.LOAD,DISP=SHR,
    UNIT=3380,VOL=SER=VM2T5O
SYSLIB DD DSN=CEE.SCEERUN,DISP=SHR
SYSABEND DD SYSOUT=* SYSOUT=*
PL/I Example
Part 2. Reference Pattern Services

<table>
<thead>
<tr>
<th>Chapter 5. Introduction to Reference Pattern Services</th>
<th>5-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>How Does the System Manage Data?</td>
<td>5-1</td>
</tr>
<tr>
<td>An Example of How the System Manages Data in an Array</td>
<td>5-2</td>
</tr>
<tr>
<td>What Pages Does the System Bring in When a Gap Exists?</td>
<td>5-4</td>
</tr>
<tr>
<td>Example 1</td>
<td>5-4</td>
</tr>
<tr>
<td>Example 2</td>
<td>5-4</td>
</tr>
<tr>
<td>Example 3</td>
<td>5-5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 6. Using Reference Pattern Services</th>
<th>6-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining the Reference Pattern for a Data Area</td>
<td>6-1</td>
</tr>
<tr>
<td>Defining the Range of the Area</td>
<td>6-1</td>
</tr>
<tr>
<td>Identifying the Direction of the Reference</td>
<td>6-2</td>
</tr>
<tr>
<td>Defining the Reference Pattern</td>
<td>6-2</td>
</tr>
<tr>
<td>Using CSRIRP When a Gap Exists</td>
<td>6-3</td>
</tr>
<tr>
<td>Choosing the Number of Bytes on a Page Fault</td>
<td>6-4</td>
</tr>
<tr>
<td>Examples of Using CSRIRP to Define a Reference Pattern</td>
<td>6-5</td>
</tr>
<tr>
<td>Removing the Definition of the Reference Pattern</td>
<td>6-6</td>
</tr>
<tr>
<td>Handling Return Codes</td>
<td>6-7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 7. Reference Pattern Services</th>
<th>7-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSRIRP — Define a Reference Pattern</td>
<td>7-1</td>
</tr>
<tr>
<td>Return Codes and Reason Codes</td>
<td>7-3</td>
</tr>
<tr>
<td>CSRRRP — Remove a Reference Pattern</td>
<td>7-3</td>
</tr>
<tr>
<td>Return Codes and Reason Codes</td>
<td>7-4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 8. Reference Pattern Services Coding Examples</th>
<th>8-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/370 Example</td>
<td>8-1</td>
</tr>
<tr>
<td>COBOL Example</td>
<td>8-4</td>
</tr>
<tr>
<td>FORTRAN Example</td>
<td>8-8</td>
</tr>
<tr>
<td>Pascal Example</td>
<td>8-11</td>
</tr>
<tr>
<td>PL/I Example</td>
<td>8-13</td>
</tr>
</tbody>
</table>
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.

\[
\begin{array}{cccc}
A(1,1) & A(1,2) & A(1,3) & A(1,4) \\
A(2,1) & A(2,2) & A(2,3) & A(2,4) \\
A(3,1) & A(3,2) & A(3,3) & A(3,4)
\end{array}
\]

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

<table>
<thead>
<tr>
<th>Sequence of Element in Storage</th>
<th>FORTRAN Array Element</th>
<th>COBOL, Pascal, PL/1, C Array Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A(1,1)</td>
<td>A(1,1)</td>
</tr>
<tr>
<td>2</td>
<td>A(2,1)</td>
<td>A(1,2)</td>
</tr>
<tr>
<td>3</td>
<td>A(3,1)</td>
<td>A(1,3)</td>
</tr>
<tr>
<td>4</td>
<td>A(1,2)</td>
<td>A(1,4)</td>
</tr>
<tr>
<td>5</td>
<td>A(2,2)</td>
<td>A(2,1)</td>
</tr>
<tr>
<td>6</td>
<td>A(3,2)</td>
<td>A(2,2)</td>
</tr>
<tr>
<td>7</td>
<td>A(1,3)</td>
<td>A(2,3)</td>
</tr>
<tr>
<td>8</td>
<td>A(2,3)</td>
<td>A(2,4)</td>
</tr>
<tr>
<td>9</td>
<td>A(3,3)</td>
<td>A(3,1)</td>
</tr>
<tr>
<td>10</td>
<td>A(1,4)</td>
<td>A(3,2)</td>
</tr>
<tr>
<td>11</td>
<td>A(2,4)</td>
<td>A(3,3)</td>
</tr>
<tr>
<td>12</td>
<td>A(3,4)</td>
<td>A(3,4)</td>
</tr>
</tbody>
</table>

Examples in this chapter and in 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.

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:

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.

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:

![Diagram showing page boundaries and gap](chart.png)

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.

```c
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:

```plaintext
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.

#### Pattern #1: No uniform gap

```
0 4096 8192 12288 16384 20480 ...
```

Characteristics of pattern:
- No uniform gap
- Reference in regular intervals (such as every element) or in irregular intervals

#### Pattern #2: Uniform gap

```
0 20 5020 5040 10040 10060 15060 15080 20080 ...
```

Characteristics of pattern:
- Gaps of uniform size
- Reference units, uniform in size, that occur in a repeating pattern

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`.

---

6-2 z/OS V1R11.0 MVS Callable Services for HLL
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](image)

- **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.
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 MV 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); \textit{unitsize} can be 4000 bytes and \textit{units} can be 20. The following CSRIRP service communicates this pattern to the system:

\begin{verbatim}
CSRIRP with unitsize of 4000 bytes
  gapsize of 0 bytes
  units of 20
  direction of +1
\end{verbatim}

**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). \textit{unitsize} can again be 4000 bytes and \textit{units} can be 50. The following CSRIRP service communicates this pattern:

\begin{verbatim}
CSRIRP with unitsize of 4000 bytes
  gapsize of 0 bytes
  units of 50
  direction of +1
\end{verbatim}

**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:

\begin{verbatim}
CSRIRP with unitsize of 4096*2 bytes
  gapsize of 4096 bytes
  units of 4
  direction of +1
\end{verbatim}

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:

4×8194 bytes on each page fault

\[\text{not brought into central storage}\]

---

**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. \textit{low_address} and \textit{size} have the same values you coded on the CSRIRP service that defined the reference pattern for that area.

\begin{verbatim}
CSRRRP with low_address of ARRAY1(1,1)
  size of 1024×1024×8 bytes
\end{verbatim}
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”
- “CSRRRP — Remove a Reference Pattern” on page 7-3.

**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:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None</td>
<td>CSRIRP completed successfully.</td>
</tr>
<tr>
<td>04</td>
<td>xx0001xx</td>
<td>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.</td>
</tr>
<tr>
<td>08</td>
<td>xx0002xx</td>
<td>Unsuccessful completion. The range that the caller specified overlaps the range that a previous request specified.</td>
</tr>
<tr>
<td>08</td>
<td>xx0003xx</td>
<td>Unsuccessful completion. The number of CSRIRP requests for the user exceeds 100, the maximum number the system allows.</td>
</tr>
<tr>
<td>08</td>
<td>xx0004xx</td>
<td>Unsuccessful completion. Storage is not available for the CSRIRP service.</td>
</tr>
<tr>
<td>08</td>
<td>00000004</td>
<td>Unsuccessful completion. The direction that the caller specified is not valid.</td>
</tr>
</tbody>
</table>

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.
CSRRRP

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

`siz`e

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.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None</td>
<td>CSRRRP completed successfully.</td>
</tr>
<tr>
<td>08</td>
<td>xx0101xx</td>
<td>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.</td>
</tr>
</tbody>
</table>
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" on page 8-4
- "COBOL Example" on page 8-8
- "FORTRAN Example" on page 8-12
- "Pascal Example" on page 8-11
- "PL/I Example" on page 8-13

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;
  long int units;
```c
arraysize = m*n*kelement_size;
direction = csr_forward;
unitsize = kelement_size*n;
gap = 0;
units = 20;

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

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

csrirp(A, &arraysize, &direction,;
&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;
    }
}

csrirp(A, &arraysize, &direction,;
&arraysize,;
&rc,;
&rsn);
chk_code(rc,rsn,LINE__);

csrirp(B, &arraysize, &direction,;
&arraysize,;
&direction,;
&unitsize,;
&gap,;
&units,;
&rc,;
&rsn);
chk_code(rc,rsn,LINE__);
arraysize = m*p*kelement_size;
csrirp(B, &arraysize, &direction,;
&arraysize,;
&direction,;
&unitsize,;
&gap,;
&units,;
&rc,;
&rsn);
chk_code(rc,rsn,LINE__);
arraysize = m*n*kelement_size;
units = 25;
csrirp(A, &arraysize, &direction,;
&arraysize,;
&direction,;
&unitsize,;
&gap,;
&units,;
&rc,;
&rsn);
chk_code(rc,rsn,LINE__);
arraysize = m*n*kelement_size;
gap = (p-1)*kelement_size;
units = 50;
csrirp(B, &arraysize, &direction,;
&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*element_size;
csrrrp(A, &arraysize, &rc, &rsn);
chk_code(rc,rsn,__LINE__);
arraysize = n*p*element_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 */
/*--------------------------------------------------------------------*/
/* JCL USED TO EXECUTE THE C/370 PROGRAM */
/*--------------------------------------------------------------------*/
IDENTIFICATION DIVISION.
*****************************************************************************
* MULITPLY 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,
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,
  CSРFORWARD,
  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,
  CSРFORWARD,
  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,
   RSNODE

DISPLAY "EIGHTH RETURN CODE IS "
DISPLAY RETCODE

DISPLAY "BPAGE PROGRAM END"
GOBACK.

/**************************************************************/
/* JCL USED TO COMPIL, LINK, THE COBOL PROGRAM */
/**************************************************************/

//FCHANGC JOB 'D3113P,D31,?','FCHANG6-6756',CLASS=T,
// MSGCLASS=H,NOTIFY=FCHANG,REGION=OK
//CSTEP EXEC EDCCO,
// CPARM='LIST,XREF,OPTIMIZE,RENT,SOURCE',
// INFILE='FCHANG.PUBL.TEST(C)' 
//COMPILE.SYSLIN DD DSN='FCHANG.MPS.OBJ(C),DISP=SHR'
//COMPILE.USERLIB DD DSN='FCHANGDECLARE.SET,DISP=SHR'
//LKSTEPS EXEC EDCPLO,
// LPARM='AMOD=31,List,REFR,RENT,RMOD=ANY,XREF'
//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.SYSLIN DD *
// LIBRARY IN(CSRIRP,CSRRRP)
// NAME BPGC(R)
//LKED.IN DD DSN=FCHANG.MPS.OBJ,DISP=SHR
/**************************************************************/
/* LINK PROGRAM */
/**************************************************************/

//COBOLLK JOB
//LINKEDIT EXEC PGM=IEWL,
// PARM='MAP,XREF,LIST,LET,AC=1,SIZE=100K',
// SYSLIN DD DDNAME=SYSLIB
// SYSLMOD DD DSN='FCHANG.USER.OBJ',DISP=OLD
// SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR
// MYLIB DD DSN='FCHANG.USER.OBJ',DISP=SHR
// CSRLIB DD DSN='SYS1.CSSLIB,DISP=SHR'
// SYSPRINT DD SYSOUT=* 
/**************************************************************/
/* JCL USED TO EXECUTE THE COBOL PROGRAM */
/**************************************************************/

//COB2 JOB MSGLEVEL=(1,1),TIME=1440
//GO EXEC PGM=COBLOAD
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
// DD DSN='FCHANG.USER.OBJ',DISP=SHR, VOL=SER=RSMPAK,
// UNIT=3380
//SYSABOUT DD SYSOUT=*
FORTRAN Example

********************************************************************
* This is FORTRAN. Followed by an assembler routine called ADDR that has to be linked with the object code from this testcase, and the CSR stubs.
********************************************************************

@PROCESS DC(BPAGEFOR)
PROCEDURE BPAGEFOR

C
INCLUDE 'SYS1.SAMPLIB(CSRBPFOR)'

C
Multiply two arrays together - testing CSIRP, CSRRP services

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
Initialize the arrays

C
CALL CSIRP(A(1,1),
*   M*N*NKELEMENT_SIZE,
*   CSR_FORWARD,
*   M*NKELEMENT_SIZE,
*   0,
*   20,
*   RC,
*   RSN)
CALL CSIRP(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 CSRRP(A(1,1),
*   M*N*NKELEMENT_SIZE,
*   RC,
*   RSN)
CALL CSRRP(B(1,1),
*   N*P*NKELEMENT_SIZE,
*   RC,
FORTRAN Example

* RSN)

C Multiply the two arrays together

CALL CSRIRP (A(1,1),
* M*N*KELEMENT_SIZE,
* CSR_FORWARD,
* N*KELEMENT_SIZE,
* (N-1)*KELEMENT_SIZE,
* 50,
* RC,
* RSN)

CALL CSRIRP (B(1,1),
* N*P*KELEMENT_SIZE,
* CSR_FORWARD,
* KELEMENT_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)
      CONTINUE
   CONTINUE
112 CONTINUE

CALL CSRIRRP (A(1,1),
* M*N*KELEMENT_SIZE,
* RC,
* RSN)

CALL CSRIRRP (B(1,1),
* N*P*KELEMENT_SIZE,
* RC,
* RSN)

STOP

END

********************************************************************
00010000
* THIS IS THE JCL THAT Compiles THE PROGRAM.
* *
********************************************************************
00030000
//FORTJOB JOB
00090007
// MSGCLASS=H,RDR=R,
00110007
// MSGLEVEL=(1,1),CLASS=T
00120000
//*
00130000
//*
00140000
//*
00150000
//*
00160000
//*
00170000
//*
00180000
//*
00190000
//*
00200000
//*
00210000
//*
00220000
//*
00230000
//*
00240000
//*
00250000
//*
00260000
//*
00270000
//*
00280000
//*
00290000
//*
00300000
//*
00310000
//*
00320000
//*
00330000
//*
FORTRAN Example

Compilation Options:
- FVPDECK
- FVPOLST
- FVPOPT
- FVTTERM
- FVLNSPC
- PGMLIB
- PGNAME
- PGTERM

JCL Options:
- STEPLIB
- SYSPRINT
- SYSTERM
- SYSLIN
- SYSLIB
- SYSUT1
- SYSLMOD
- SYSLIN

Environment:
- PGMLIB
- LKED.SYSLMOD
- LKED.SYSIN
- LKED.IN

Execution:
- EXEC VSF2CL, GOPGM='MAIN', GOREGN=100K,
- EXEC VSF2G, GOPGM='MAIN', GOREGION=100K,
- EXEC VSF2M, FVTTERM='SYSOUT=H',
- EXEC VSF2CL, FVTERM='SYSOUT=A',
- EXEC VSF2G, GOPGM='MAIN', GOREGION=100K,
- EXEC VSF2M, FVTTERM='SYSOUT=A',

Important Notes:
- COPYRIGHT: 5668-806
- LICENSED MATERIALS - PROPERTY OF IBM
- REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
- STATUS: 02.03.00 (VV.RR.MM)
- PEND

- EXECUTE A FORTRAN TESTCASE - CHANGE ALL CRTFONXX TO CRTFONZZ

COPYRIGHT: IBM 1985, 1988
LICENSED MATERIALS - PROPERTY OF IBM
REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
STATUS: 02.03.00 (VV.RR.MM)

PARAMETER DEFAULT-VALUE

COPYRIGHT: 5668-806
(C) COPYRIGHT IBM CORP 1985, 1988
LICENSED MATERIALS - PROPERTY OF IBM
REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
STATUS: 02.03.00 (VV.RR.MM)

EXECUTE A FORTRAN TESTCASE - CHANGE ALL CRTFONXX TO CRTFONZZ

COPYRIGHT: IBM 1985, 1988
LICENSED MATERIALS - PROPERTY OF IBM
REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
STATUS: 02.03.00 (VV.RR.MM)

PARAMETER DEFAULT-VALUE

COPYRIGHT: 5668-806
(C) COPYRIGHT IBM CORP 1985, 1988
LICENSED MATERIALS - PROPERTY OF IBM
REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
STATUS: 02.03.00 (VV.RR.MM)

PARAMETER DEFAULT-VALUE

COPYRIGHT: 5668-806
(C) COPYRIGHT IBM CORP 1985, 1988
LICENSED MATERIALS - PROPERTY OF IBM
REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
STATUS: 02.03.00 (VV.RR.MM)

PARAMETER DEFAULT-VALUE

COPYRIGHT: 5668-806
(C) COPYRIGHT IBM CORP 1985, 1988
LICENSED MATERIALS - PROPERTY OF IBM
REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
STATUS: 02.03.00 (VV.RR.MM)

PARAMETER DEFAULT-VALUE

COPYRIGHT: 5668-806
(C) COPYRIGHT IBM CORP 1985, 1988
LICENSED MATERIALS - PROPERTY OF IBM
REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
STATUS: 02.03.00 (VV.RR.MM)

PARAMETER DEFAULT-VALUE

COPYRIGHT: 5668-806
(C) COPYRIGHT IBM CORP 1985, 1988
LICENSED MATERIALS - PROPERTY OF IBM
REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
STATUS: 02.03.00 (VV.RR.MM)

PARAMETER DEFAULT-VALUE

COPYRIGHT: 5668-806
(C) COPYRIGHT IBM CORP 1985, 1988
LICENSED MATERIALS - PROPERTY OF IBM
REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
STATUS: 02.03.00 (VV.RR.MM)

PARAMETER DEFAULT-VALUE

COPYRIGHT: 5668-806
(C) COPYRIGHT IBM CORP 1985, 1988
LICENSED MATERIALS - PROPERTY OF IBM
REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
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

\[
b(.,j.) := i + j; \\
\text{csrrrp (a(.,1.), a\_size, rc, rsn);} \\
\text{csrrrp (b(.,1.), b\_size, rc, rsn);} \\
/* Multiply the two arrays together */ \\
\text{csrirp (a(.,1.), m*n*kelement\_size, csr\_forward, kelement\_size*n, 0, 20, rc, rsn);} \\
\text{csrirp (b(.,1.), n*p*kelement\_size, csr\_forward, (p-1)*kelement\_size, 0, 50, rc, rsn);} \\
\text{for i:=1 to m do} \\
\text{for j:=1 to p do} \\
\text{begin; c(.,i,j.) := 0; for k:=1 to n do c(.,i,j.) := c(.,i,j.) + a(.,i,k.) * b(.,k,j.); end; } \\
\text{csrrrp (a(.,1.), m*n*kelement\_size, rc, rsn);} \\
\text{csrrrp (b(.,1.), n*p*kelement\_size, rc, rsn);} \\
\text{END.} \\
\]

**********************************************************************
* JCL TO COMPILEx AND LINKEDIT                        00010000
* *                                                            00020000
* * JCL TO COMPILEx AND LINKEDIT                        00030000
* *                                                            00040000
**********************************************************************
//PASCJOB JOB                                          00050000
//GOGO EXEC PAS22CL                                     00060008
//*                                                   00100000
//* COMPILEx AND LINKEDIT FOR PASCAL                      00110000
//*                                                   00120000
//* CHANGE THE MEMBER NAME ON THE NEXT LINE AND THE       00130000
//* NAME CRTPANXX(R) SIX LINES DOWN                      00140000
//*                                                   00150000
//*                                                   00160000
//PASC.SYSLIB DD                                          00161006
// DD                                                   00162006
// DD DSN=REFPAT.DECLARE.SET(CSRBPPAS),DISP=SHR           00163008
//PASC.SYIN DD DSN=REFPAT.SAMPLE.PROG(PASCAL),DISP=SHR    00170008
//LKED.SYSLMOD DD DSN=REFPAT.USER.LOAD,DISP=SHR,UNIT=3380,
// VOL=SER=VM2TSO                                        00180008
//LKED.SYSLIN DD *                                       00190009
//LIBRARY IN(CSRIRP,CSRRRP)                              00210005
// NAME BPGPASC(R)                                       00220003
//*                                                   00230000
// SYSI.CSSLIB IS THE SOURCE OF THE CSR STUBS             00240008
//*                                                   00250000
//LKED.IN DD DSN=SYSI.CSSLIB,DISP=SHR                   00260008
**********************************************************************
* JCL TO EXECUTE PASCAL                               00270000
* *                                                            00280000
**********************************************************************
Pascal Example

```pascal
//PASCIJOB JOB
//GO EXEC PAS22CL
// Compile and linkedit for PASCAL
//PASC.SYSIN DD=WINDOW.XAMPLE.LIB(CRTPAN06),DISP=SHR
//LKED.SYSMOD DD=WINDOW.USER.LOAD,DISP=SHR,UNIT=3380,
// VOL=SER=VM2TSO
//LKED.SYSIN DD *
// LIBRARY IN(CRSSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC)
// NAME CRTPAN06(R)
// SYS1.CSSLIB is the source of the CSR stubs
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR

******************************************************************************
* JCL TO COMPILE AND LINKEDIT.                                          *
******************************************************************************

//PASCGO JOB MSGLEVEL=(1,1),TIME=1440
// STEPLIB DD DSN=REFPAT.USER.LOAD,
// DISP=SHR,UNIT=3380,
// VOL=SER=VM2TSO
//CSRDD1 DD DSN=DIV.TESTDS,DISP=SHR
//OUTPUT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=133)
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=133)

PL/I Example

******************************************************************************
* PLI example                                                          *
******************************************************************************

BPGPLI: PROCEDURE OPTIONS(MAIN);
%INCLUDE SYSLIB(CSRBPPLI);
/* INITs */
DCL M INIT(512) FIXED BIN(31);
DCL N INIT(512) FIXED BIN(31);
DCL P INIT(512) FIXED BIN(31);
/* Arrays */
DCL A(M,N) BIN FIXED(31); /* First array */
DCL B(N,P) BIN FIXED(31); /* Second array */
DCL C(M,P) BIN FIXED(31); /* Product of first and second */
DCL KELEMENT_SIZE INIT(4) FIXED BIN(31); /* Size of an element of an array. This value is tied directly to the data type of */
```

Chapter 8. Reference Pattern Services Coding Examples 8-13
the three arrays (i.e. FIXED(31) is 4 bytes)

/* Indices */
DCL I FIXED BIN(31),
J FIXED BIN(31),
K FIXED BIN(31);

/* Others */
DCL RC FIXED BIN(31);
DCL RSN FIXED BIN(31);

/* Initialize the first two arrays such that each element equals the sum of the indices for that element (eg. A(4,10) = 14 */

CALL CSIRP (A(1,1), M*N*KELEMENT_SIZE, CSR_FORWARD,
KELEMENT_SIZE=N,
0,
20,
RC,
RSN);

CALL CSIRP (B(1,1), N*P*KELEMENT_SIZE, CSR_FORWARD,
KELEMENT_SIZE=P,
0,
20,
RC,
RSN);

DO I = 1 TO M;
DO J = 1 TO N;
A(I,J) = I + J;
END;
END;

DO I = 1 TO N;
DO J = 1 TO P;
B(I,J) = I + J;
END;
END;

CALL CSRRRP (A(1,1), M*N*KELEMENT_SIZE,
RC,
RSN);

CALL CSRRRP (B(1,1), N*P*KELEMENT_SIZE,
RC,
RSN);

/* Multiply the two arrays together */

CALL CSIRP (A(1,1), M*N*KELEMENT_SIZE, CSR_FORWARD,
KELEMENT_SIZE=N,
0,
20,
RC,
RSN);

CALL CSIRP (B(1,1), N*P*KELEMENT_SIZE, CSR_FORWARD,
KELEMENT_SIZE,
(P-1)*KELEMENT_SIZE,
50,
RC,
RSN);

DO I = 1 TO M;
DO J = 1 TO P;
C(I,J) = 0;
DO K = 1 TO N;

C(I,J) = C(I,J) + A(I,K) * B(K,J);
END;
CALL CSRRRP (A(1,1), M*N*KELEMENT_SIZE, RC, RSN);
CALL CSRRRP (B(1,1), N*P*KELEMENT_SIZE, RC, RSN);
END BPGPLI;

***********************************************************************
* JCL TO COMPIL AND LINKEDIT. *
***********************************************************************

//PLIJOB JOB
// PL/I Compile and Linkedit
// Change all CRTPLNx to CRTPLNy
//GO EXEC PLIXCL, PARM.PLI='MACRO'
//PLI.SYSLIB DD DSN=REFPAT.DECLARE.SET, DISP=SHR
//PLI.SYSIN DD DSN=REFPAT.SAMPLE.PROG(PLI), DISP=SHR
//LKED.SYSLMOD DD DSN=REFPAT.USER.LOAD, UNIT=3380, VOL=SER=RSMPAK,
// DISP=SHR
//LKED.SYSIN DD *
// IN(CSRIRP,CSRRRP)
// NAME BPGPLI(R)
// SYSLIB is source of CSR stubs
//LKED.IN DD SYS1.CSSLIB, DISP=SHR
//NAME=SYS1.CSSLIB
// PLIJOB JOB

***********************************************************************
* JCL TO EXECUTE. *
***********************************************************************

//PLIRUN JOB MSGLEVEL=(1,1), TIME=1440
// EXECUTE A PL/I TESTCASE - CHANGE NAME ON NEXT LINE
//GO EXEC PGM=CRTPLN3
// STEPLIB DD DSN=REFPAT.USER.LOAD, DISP=SHR,
// UNIT=3380, VOL=SER=VM2TSO
// DD DSN=CEE.SCEERUN, DISP=SHR
// SYSABEND DD SYSPRINT DD SYSOUT**
// SYSPRINT DD SYSOUT**
PL/I Example
Part 3. Global Resource Serialization Latch Manager Services

Chapter 9. Using the Latch Manager Services ........................................ 9-1
Syntax and Linkage Conventions for Latch Manager Callable Services .......... 9-1
ISGLCRT — Create a Latch Set ......................................................... 9-2
  ABEND Codes ............................................................................ 9-4
  Return Codes ............................................................................ 9-4
  Examples of Calls to Latch Manager Services .................................. 9-4
ISGLOBT — Obtain a Latch ............................................................... 9-6
  ABEND Codes ............................................................................ 9-8
  Return Codes ............................................................................ 9-8
  Example ...................................................................................... 9-9
ISGLREL — Release a Latch ............................................................... 9-9
  ABEND Codes ............................................................................ 9-11
  Return Codes ............................................................................ 9-11
  Example ...................................................................................... 9-12
ISGLPRG — Purge a Requestor from a Latch Set ................................. 9-12
  ABEND Codes ............................................................................ 9-13
  Return Codes ............................................................................ 9-13
  Example ...................................................................................... 9-13
ISGLPBA — Purge a Group of Requestors from a Group of Latch Sets ....... 9-13
  ABEND Codes ............................................................................ 9-15
  Return Codes ............................................................................ 9-15
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:

- “ISGLCRT — Create a Latch Set” on page 9-2
- “ISGLOBT — Obtain a Latch” on page 9-6
- “ISGLREL — Release a Latch” on page 9-9
- “ISGLPRG — Purge a Requestor from a Latch Set” on page 9-12
- “ISGLPBA — Purge a Group of Requestors from a Group of Latch Sets” on page 9-13

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 Callable Service

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 IDF s 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 one of the following values:

- ISGLCRT_PRIVATE (or a value of 0)
- ISGLCRT_PRIVATE + ISGLCRT_LOWSTGUSAGE (or a value of 2)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 (or a value of 64)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 (or a value of 128)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 + ISGLCRT_LOWSTGUSAGE (or a value of 66)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 + ISGLCRT_LOWSTGUSAGE (or a value of 130)

If the creating address space is constrained by private storage, use the ISGLCRT_LOWSTGUSAGE option. ISGLCRT_LOWSTGUSAGE reduces storage usage at the cost of performance. IBM suggests that this option is only used if there is a known or possible storage constraint issue. See “Specifying the Number of Latches in a Latch Set” section in z/OS MVS Programming [Authorized Assembler Services Guide] for a description of the amount of storage that can be consumed by a latch set.

If you want to have the latch obtain services detect some simple latch deadlock situations, consider using the ISGLCRT_DEADLOCKDET1 and ISGLCRT_DEADLOCKDET2 options. For performance reasons, latch deadlock detection is not exhaustive. It can detect some simple deadlock situations.

When ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 is specified, it can detect the following deadlock situations:

- The work unit requests exclusive ownership of a latch that the work unit already owns exclusively.
- The work unit requests shared ownership of a latch that the work unit already owns exclusively.

When ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 is specified, it can detect all the deadlock situations listed under ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1, and it can also detect if the work unit holding a SHARED latch requests exclusive use of the same latch.

Because ISGLCRT_DEADLOCKDET2 provides the best deadlock detection, IBM suggests that you use ISGLCRT_DEADLOCKDET1 in cases where it can be used and use ISGLCRT_DEADLOCKDET2 in all cases where there are not many SHARED latch holders.

Note:

1. The unit of work context of the requester is captured at latch obtain time. The system does not know if the application passes responsibility for releasing the latch to another unit of work. To prevent false detection, deadlock detection can not be used if latches are used in such a way that responsibility for releasing the latch is passed between the obtainer and the releaser.
2. Deadlock detection can be safely used by SRBs, if all the obtained latches are released by the SRB work unit before the unit of work completes. There is a possibility of false deadlock hits otherwise.
3. Deadlock detection is not performed if the latches are obtained conditionally using the ISGLOBT_ASYNC_ECB option in ISGLOBT.

.latch_set_token

Specifies an 8-byte area to contain the latch set token returned by the
ISGLCRT Callable Service

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>
<thead>
<tr>
<th>Return code and Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (0) ISGLCRT_SUCCESS</td>
<td>Meaning: The Latch_Create service completed successfully.</td>
</tr>
<tr>
<td></td>
<td>Action: None required.</td>
</tr>
<tr>
<td>04 (4) ISGLCRT_DUPLICATE_NAME</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>10 (16) ISGLCRT_NO_STORAGE</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Action: Specify a smaller value on the number_of_latches parameter.</td>
</tr>
</tbody>
</table>
```

Examples of Calls to Latch Manager Services

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

```
#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 */
} /* C Example */
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 */

}
ISGLCRT Callable Service

SAVE (14,12)    save regs
SAC 0          ensure primary mode
LR 12,15       establish addressability
USING SETSUP,12 set supervisor state
MODESET MODE=SUP ensure primary mode
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 set problem state
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:
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 9-12.

**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>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (0) ISGLOBT_SUCCESS</td>
<td><strong>Meaning</strong>: The Latch_Obtain service completed successfully. <strong>Action</strong>: None.</td>
</tr>
</tbody>
</table>
Table 9-2. ISGLOBT Return Codes (continued)

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>04 (4) ISGLOBT_CONTENTION</td>
<td>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.</td>
</tr>
</tbody>
</table>

Example

See [“Examples of Calls to Latch Manager Services” on page 9-4](#) 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 IDF 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](https://www.ibm.com) 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*

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 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. |
ISGLREL Callable Service

Table 9-3. ISGLREL Return Codes (continued)

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C (12) ISGLREL_INCORRECT_LATCH_TOKEN</td>
<td>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.</td>
</tr>
</tbody>
</table>

Example

See [Examples of Calls to Latch Manager Services](#) on page 9-4 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
- requester_ID

Latch_Purge returns a value in the return_code parameter.

```call
CALL ISGLPRG
  (latch_set_token
   ,requester_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


### 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 parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (0) ISGLPRG_SUCCESS</td>
<td>Meaning: The Latch_Purge service completed successfully. Action: None.</td>
</tr>
<tr>
<td>04 (4) ISGLPRG_DAMAGE_DETECTED</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Example

See ["Examples of Calls to Latch Manager Services" on page 9-4](#) 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 space 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 fullword 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

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (0) ISGLPRG_SUCCESS</td>
<td>Meaning: The Latch_Purge_by_Address_Space service completed successfully.</td>
</tr>
<tr>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>04 (4) ISGLPRG_DAMAGE_DETECTED</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
ISGLPBA Callable Service
<table>
<thead>
<tr>
<th>Chapter 10. Using Protected Resources</th>
<th>10-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Recovery Programs</td>
<td>10-1</td>
</tr>
<tr>
<td>Two-Phase Commit Protocol</td>
<td>10-2</td>
</tr>
<tr>
<td>Resource Recovery Process</td>
<td>10-2</td>
</tr>
<tr>
<td>Requesting Resource Protection and Recovery</td>
<td>10-4</td>
</tr>
<tr>
<td>Using Distributed Resource Recovery</td>
<td>10-5</td>
</tr>
<tr>
<td>Application Backout UR (SRRBACK)</td>
<td>10-5</td>
</tr>
<tr>
<td>Description</td>
<td>10-5</td>
</tr>
<tr>
<td>Environment</td>
<td>10-5</td>
</tr>
<tr>
<td>Programming Requirements</td>
<td>10-6</td>
</tr>
<tr>
<td>Application Commit UR (SRRCMIT)</td>
<td>10-9</td>
</tr>
<tr>
<td>Description</td>
<td>10-9</td>
</tr>
<tr>
<td>Environment</td>
<td>10-9</td>
</tr>
<tr>
<td>Programming Requirements</td>
<td>10-9</td>
</tr>
<tr>
<td>Restrictions</td>
<td>10-10</td>
</tr>
<tr>
<td>Input Register Information</td>
<td>10-10</td>
</tr>
<tr>
<td>Output Register Information</td>
<td>10-10</td>
</tr>
<tr>
<td>Performance Implications</td>
<td>10-11</td>
</tr>
<tr>
<td>Syntax</td>
<td>10-11</td>
</tr>
<tr>
<td>Parameters</td>
<td>10-11</td>
</tr>
<tr>
<td>ABEND Codes</td>
<td>10-11</td>
</tr>
<tr>
<td>Return Codes</td>
<td>10-11</td>
</tr>
<tr>
<td>Example</td>
<td>10-13</td>
</tr>
</tbody>
</table>
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.

The actions required to process the ATM transaction are:

![ATM Transaction Diagram](image-url)
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.

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.
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 10-3. Backout — Application Request**

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:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATRSASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>ATRSC</td>
<td>C/390 declarations</td>
</tr>
<tr>
<td>ATRSCOB</td>
<td>COBOL 390 declarations</td>
</tr>
<tr>
<td>ATRSPAS</td>
<td>Pascal 390 declarations</td>
</tr>
<tr>
<td>ATRSPLI</td>
<td>PL/I 390 declarations</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 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 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:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATRSASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>ATRSC</td>
<td>C/390 declarations</td>
</tr>
<tr>
<td>ATRSCOB</td>
<td>COBOL 390 declarations</td>
</tr>
<tr>
<td>ATRSPAS</td>
<td>Pascal 390 declarations</td>
</tr>
<tr>
<td>ATRSPLI</td>
<td>PL/I 390 declarations</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>
When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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](https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.doc/zos_r0000_systemcodes.html) 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](https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.doc/zos_programming.html) 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](https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.doc/zos_programming.html) contains additional details about these return codes.
<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Code: RR_OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: Successful completion. The changes to all protected resources have been made permanent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>65</td>
<td>101</td>
<td>Code: RR_COMMITTED_OUTCOME_PENDING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: Environmental error. The commit was not completed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: The action by an application depends on the system environment. Some possible actions are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Display a warning message to the end user.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Write an exception entry into an output log.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Abnormally end the application because the resource manager will not allow any further changes to the resource until the situation is resolved.</td>
</tr>
<tr>
<td>66</td>
<td>102</td>
<td>Code: RR_COMMITTED_OUTCOME_MIXED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Same as the action for return code 65 (101).</td>
</tr>
<tr>
<td>C8</td>
<td>200</td>
<td>Code: RR_PROGRAM_STATE_CHECK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: Environmental error. The commit failed. The resource managers did not make the changes to the resources because one of the following occurred:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A resource on the same system as the application is not in the proper state for a commit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A protected conversation is not in the required state: send, send pending, defer receive, defer allocate, sync point, sync point send, sync point deallocate.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A protected conversation is in send state. The communications manager started sending the basic conversation logical record, but did not finish sending it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>12C</td>
<td>300</td>
<td>Code: RR_BACKED_OUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Same as the action for return code 65 (101).</td>
</tr>
<tr>
<td>Hexadecimal Return Code</td>
<td>Decimal Return Code</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>12D</td>
<td>301</td>
<td>Code: RR_BACKED_OUT_OUTCOME_PENDING</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> Environmental error. The commit failed for one of the following reasons:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· 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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· RRS is not active.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Same as the action for return code 65 (101).</td>
</tr>
<tr>
<td>12E</td>
<td>302</td>
<td>Code: RR_BACKED_OUT_OUTCOME_MIXED</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> 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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Same as the action for return code 65 (101).</td>
</tr>
</tbody>
</table>

**Example**

In the pseudocode example, the application issues a call to request that RRS commit a UR.

```plaintext
::
CALL SRRCMIT(RETCODE)
::
```
Application_Commit_URI
Part 5. Other Callable Services

Chapter 11. IEAAFFN — Assign Processor Affinity for Encryption or Decryption

Restrictions and Limitations

Requirements

Return Codes

Chapter 12. CSRL16J — Transfer Control to Another Routine

Defining the Entry Characteristics of the Target Routine

Freeing Dynamic Storage Associated with the Caller

Programming Requirements

Restrictions

Performance Implications

Syntax Diagram

C/370 Syntax

PL/I Syntax

Parameters

Return Codes

Example

C/370 Example Program

Assembler program for use with the C/370 example

Chapter 13. CSRSI — System Information Service

Description

Environment

Programming Requirements

Restrictions

Input Register Information

Output Register Information

Syntax

Parameters

Return Codes

CSRSIC C/370 Header File
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.
- 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>
<thead>
<tr>
<th>Return code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 (0)</td>
<td><strong>Meaning:</strong> The operation was successful. <strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>00000004 (4)</td>
<td><strong>Meaning:</strong> 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. <strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>0000000C (12)</td>
<td><strong>Meaning:</strong> Your program was not running in task mode. <strong>Action:</strong> 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.</td>
</tr>
<tr>
<td>00000010 (16)</td>
<td><strong>Meaning:</strong> The feature you specified was not a valid feature. <strong>Action:</strong> Specify a valid feature name.</td>
</tr>
<tr>
<td>00000014 (20)</td>
<td><strong>Meaning:</strong> The operation type you specified was not valid. <strong>Action:</strong> Specify a valid operation type.</td>
</tr>
<tr>
<td>00000018 (24)</td>
<td><strong>Meaning:</strong> The feature you specified is not installed on any of the processors in the system. <strong>Action:</strong> To the system programmer: See that the program runs on a system with the feature installed.</td>
</tr>
<tr>
<td>0000001C (28)</td>
<td><strong>Meaning:</strong> A system error has occurred. <strong>Action:</strong> To the system programmer: The error is recorded in LOGREC. Look for a record with a subcomponent of &quot;IEAAFFN CSS&quot;; then call your IBM Support Center.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Do not contain any information for use by the routine</td>
</tr>
<tr>
<td>2-13</td>
<td>The contents are the same as they were when the caller issued the CSRL16J service.</td>
</tr>
<tr>
<td>14-15</td>
<td>Do not contain any information for use by the routine</td>
</tr>
</tbody>
</table>

**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](https://www.ibm.com) 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 */
            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;
    }
} CSRL16J;

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 Rsvid??(8??); /* Reserved */

/* 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 */

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:
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

```
csr16j (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

<table>
<thead>
<tr>
<th>Return Code (hexadecimal)</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: Successful completion. The calling program will never see this return code because it indicates that the target routine received control. Action: None.</td>
</tr>
<tr>
<td>04</td>
<td>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.</td>
</tr>
<tr>
<td>08</td>
<td>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.</td>
</tr>
<tr>
<td>0C</td>
<td>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.</td>
</tr>
<tr>
<td>10</td>
<td>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.</td>
</tr>
<tr>
<td>18</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

```c
#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.PSAddr = (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))
{
    csr16j (&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 AMODE ANY
*  
* Assembler assist routine to save contents of registers 0 through 13

12-8  z/OS V1R11.0 MVS Callable Services for HLL
* 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.
System Information Service (CSRSI)

Output Register Information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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)]
System Information Service (CSRSI)

- 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.

<table>
<thead>
<tr>
<th>Return Code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 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;

#endif

#define csrsi_byaddr(Request, Flen, Fptr, Rcptr) ???

struct CSRSI_CSRT ??
    unsigned char CSRSI_csrt_filler1 ??(48??);
    CSRSI_calltype* CSRSI_addr;

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 1 of 14)
System Information Service (CSRSI)

```c
struct CSRSI_CVT {
    unsigned char CSRSI_cvt_filler1; /* {116} */

    struct {
        int CSRSI_cvtcb_rsvd1 : 4;  /* Not needed */
        int CSRSI_cvtoext : 1;  /* If on, indicates that the */
        int CSRSI_cvtcb_rsvd2 : 3;  /* Not needed */
    } CSRSI_cvtcb;

    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 */
        int CSRSI_cvtoislvl1_rsvd1 : 7;  /* Not needed */
    } CSRSI_cvtoslv4;

    unsigned char CSRSI_cvt_filler4; /* {11} */
};
```

```c
struct CSRSI_PSA {
    char CSRSI_psa_filler; /* {16} */

    struct CSRSI_CVT *CSRSI_cvt;
};
```

/* End of CSRSI Header */

```
define /* End of CSRSI Header */
```

```c
typedef struct {
    unsigned char _filler1; /* {32} */  /* Reserved */
    unsigned char s11v1cpcmanufacturer; /* {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 s11v1cpctype; /* {4} */  /* The 4-character (0-9) EBCDIC
        type identifier of the V1 CPC. */
    unsigned char _filler2; /* {12} */  /* Reserved */
} s11v1;
```

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 2 of 14)
unsigned char s11v1cpmodel??(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 s11v1cpcsequencecode??(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 s11v1cpcplantofmanufacture??(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 */

typedef struct ??<
unsigned char _filler1??(32??); /* Reserved */
unsigned char s12v1cpucapability??(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 s12v1totalcpucount : 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 _si22v1mpcpucapafs??(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)
typedef struct {
    unsigned char _filler1; /* 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 _si22v2lcpudirected : 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 dedicated 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. */
    } _si22v2lcpudemanded
} si22v2;

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 si22v2total1cpucount : 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 si22v2configured1cpucount : 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 si22v2standby1cpucount : 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 si22v2reserved1cpucount : 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 si22v2dedicated1cpucount : 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._si22v2lcpuddenicated
#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 si22v3dbpcpname[(8)]; /* The 8-character EBCDIC name of this V3 CPC. The name is left-justified with trailing blank characters if necessary. */

unsigned char si22v3dbpccaff[(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 si22v3dbhidentifier??(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 */
??> 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 sio0cpcvariety; /* SIO0CPCVariety_V1CPC_MACHINE, SIO0CPCVariety_V2CPC_LPAR, or SIO0CPCVariety_V3CPC_VM */
} sio0;

struct {
    int _sio0validsillv1 : 1; /* sillv1 was requested and the information returned is valid */
    int _sio0validsillv2 : 1; /* sillv2 was requested and the information returned is valid */
    int _sio0validsillv3 : 1; /* sillv3 was requested and the information returned is valid */
    int _filler1 : 4; /* Reserved */
} sio0validityflags;

unsigned char _filler2??(2??); /* Reserved */
unsigned char sio0pccacpua??(2??); /* PCCACPUA value for this CPU */
unsigned char sio0pccacafm??(2??); /* PCCACAFM value for this CPU */
unsigned char _filler3??(4??); /* Reserved */
unsigned char sio0lastupdatetimestamp??(8??); /* Time of last STSI update, via STCK */
unsigned char _filler4??(32??); /* Reserved */
} sio0;

#define sio0validsillv1 sio0validityflags._sio0validsillv1
#define sio0validsillv2 sio0validityflags._sio0validsillv2
#define sio0validsillv3 sio0validityflags._sio0validsillv3

typedef struct {
    char sio1siv1si00; /* Area mapped by struct sio0 */
    struct sio0 sio1siv1si00; /* Area */
    char sio1siv1sillv1; /* Area mapped by struct sio1siv1si00 */
    struct sio1siv1sillv1 sio1siv1sillv1; /* Area */
    char sio1siv1sillv2; /* Area mapped by struct sio1siv1sillv1 */
    struct sio1siv1sillv2 sio1siv1sillv2; /* Area */
} sio1;

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 11 of 14)
System Information Service (CSRSI)

typedef struct {?<
    si00 siv1v2si00;       /* Area mapped by */
    siv1v2si11v1;          /* Area mapped by */
    siv1v2si22v1;          /* Area mapped by */
    siv1v2si22v2;          /* Area mapped by */
?> siv1v2;

typedef struct {?<
    si00 siv1v2v3si00;     /* Area mapped by */
    siv1v2v3si11v1;        /* Area mapped by */
    siv1v2v3si22v1;        /* Area mapped by */
    siv1v2v3si22v2;        /* Area mapped by */
    siv1v2v3si22v3;        /* Area mapped by */
?> siv1v2v3;

typedef struct {?<
    si00 siv1v3si00;       /* Area mapped by */
    siv1v3si11v1;          /* Area mapped by */
    siv1v3si22v1;          /* Area mapped by */
    siv1v3si22v2;          /* Area mapped by */
    siv1v3si22v3;          /* Area mapped by */
?> 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 */
*******************************************************************************/

#ifdef SI00CPCVARIETY_V1CPC_MACHINE
#define SI00CPCVARIETY_V1CPC_MACHINE 1
#endif

#ifdef SI00CPCVARIETY_V2CPC_LPAR
#define SI00CPCVARIETY_V2CPC_LPAR 2
#endif

#ifdef SI00CPCVARIETY_V3CPC_VM
#define SI00CPCVARIETY_V3CPC_VM 3
#endif

/*******************************************************************************/
/* CSRSI Constants */
*******************************************************************************/

#ifdef CSRSI_REQUEST_V1CPC_MACHINE
#define CSRSI_REQUEST_V1CPC_MACHINE 1
#endif

#ifdef CSRSI_REQUEST_V2CPC_LPAR
#define CSRSI_REQUEST_V2CPC_LPAR 2
#endif

#ifdef CSRSI_REQUEST_V3CPC_VM
#define CSRSI_REQUEST_V3CPC_VM 4
#endif

/*******************************************************************************/
/* CSRSI Return codes */
*******************************************************************************/

#ifdef CSRSI_SUCCESS
#define CSRSI_SUCCESS 0
#endif

#ifdef CSRSI_STSINOTAVAILABLE
#define CSRSI_STSINOTAVAILABLE 4
#endif

#ifdef CSRSI_SERVICENOTAVAILABLE
#define CSRSI_SERVICENOTAVAILABLE 8
#endif

#ifdef CSRSI_BADREQUEST
#define CSRSI_BADREQUEST 12
#endif

#ifdef CSRSI_BADINFOAREALEN
#define CSRSI_BADINFOAREALEN 16
#endif

#ifdef CSRSI_BADLOCK
#define CSRSI_BADLOCK 20
#endif

Figure 13-1. CSRSIC from SYS1.SAMPLIB (Part 14 of 14)
Chapter 14. Base Control Program Internal Interface (BCPii) .......................... 14-1
  BCPii Setup and Installation ......................................................... 14-1
    Setting up Connectivity to the Support Element ................................. 14-2
      Levels of Hardware BCPii Supports ........................................... 14-2
      Enable BCPii Communications on the Support Element ......................... 14-2
      Define the BCPii Community Name on the Support Element ................... 14-3
    Setting up Authority to Use BCPii ............................................... 14-4
      Program Authority ................................................................... 14-4
      General Security Product Authority ............................................ 14-4
      Authority to the Particular Resource ......................................... 14-5
      Community name defined in the security product for each CPC ............... 14-5
  BCPii Configuration ....................................................................... 14-6
    Setting up Event Notification for BCPii USS Applications ..................... 14-6
      CEA Address Space Setup .......................................................... 14-7
      CEA ENF Security Configuration .................................................. 14-7
  BCPii Startup and Shutdown ............................................................ 14-7
    BCPii Address Space Does not Start up at IPL .................................. 14-8
    Ending the HWIBCPII Address Space .............................................. 14-8
    Restarting the HWIBCPII Address Space ......................................... 14-8
  BCPii Callable Services ................................................................. 14-9
    Syntax, Linkage and Programming Considerations ................................ 14-9
      Calling formats ........................................................................ 14-9
      Assembler considerations .......................................................... 14-9
      Linkage considerations ............................................................. 14-10
    HWICMD — Issue a BCPii Hardware Management Command ...................... 14-10
      Description .............................................................................. 14-10
      Environment. ............................................................................ 14-10
      Programming Requirements ......................................................... 14-10
      Restrictions ............................................................................ 14-10
      Authorization ........................................................................... 14-11
      Syntax ..................................................................................... 14-11
      Parameters .............................................................................. 14-11
      ABEND Codes .......................................................................... 14-16
      Return Codes ........................................................................... 14-16
      Example ................................................................................... 14-20
    HWICONN — Establish a BCPii Connection ........................................... 14-21
      Description .............................................................................. 14-21
      Environment. ............................................................................ 14-21
      Programming Requirements ......................................................... 14-21
      Restrictions ............................................................................ 14-21
      Authorization ........................................................................... 14-21
      Syntax ..................................................................................... 14-22
      Parameters .............................................................................. 14-22
      ABEND Codes .......................................................................... 14-25
      Return Codes ........................................................................... 14-25
      Example ................................................................................... 14-28
    HWIDISC — Release a BCPii Connection ............................................. 14-28
      Description .............................................................................. 14-28
      Environment. ............................................................................ 14-28
      Programming Requirements ......................................................... 14-29
      Restrictions ............................................................................ 14-29
      Authorization ........................................................................... 14-29
HWIEndEventDelivery — End Delivery of BCPii Event Notifications

Description

Environment

Programming Requirements

Restrictions

Authorization

Syntax

Parameters

ABEND Codes

Return Codes

Example

HWIManageEvents — Manage the List of BCPii Events

Description

Environment

Programming Requirements

Restrictions

Authorization

Syntax

Parameters

ABEND Codes

Return Codes

Example

HWIGetEvent — Retrieve Outstanding BCPii Event Notifications

Description

Environment

Programming Requirements

Restrictions

Authorization

Syntax

Parameters

ABEND Codes

Return Codes

Example
Chapter 14. Base Control Program Internal Interface (BCPii)

IBM provides support within z/OS that allows authorized applications to query, change, and perform operational procedures against the installed System z hardware base through a set of application program interfaces. These applications can access the System z hardware that the application is running on and extend their reach to other System z processors within the attached process control (Hardware Management Console) network.

Using the Base Control Program internal interface (BCPii), an authorized z/OS application can perform the following actions:

- Obtain the System z topology of the current interconnected Central Processor Complexes (CPCs) as well as the images, capacity records and activation profiles on a particular CPC.
- Query various CPC, image (LPAR), capacity record, and activation profile information.
- Set various configuration values related to CPC, image and activation profiles.
- Issue commands against both the CPC and image (LPAR) to perform minor or even significant hardware- and software-related functions.
- Listen for various hardware and software events that might take place on various CPCs and images throughout the HMC-connected network.

Communication to the Support Element (SE) / Hardware Management Console (HMC) using BCPii is done completely within the base operating system and therefore does not require communication on an IP network (intranet) for connectivity, providing complete isolation of your System z hardware communication from any other network traffic within the intranet/internet.

Calls using the BCPii Application Programming Interfaces (APIs) can be made from either C or assembler programming languages. See “Syntax, Linkage and Programming Considerations” on page 14-9 for an explanation of how the APIs are called, and see the explanation of each service for the syntax for each of the BCPii APIs.

BCPii Setup and Installation

Before you issue BCPii APIs, perform the following installation steps:

1. Configure the local Support Element (SE) to support BCPii. See “Setting up Connectivity to the Support Element” on page 14-2 for details.
2. Authorize an application to use BCPii, including authority to specific resources (such as CPCs, images and capacity records). See “Setting up Authority to Use BCPii” on page 14-4 for details.
3. Configure the BCPii address space. See “BCPii Configuration” on page 14-6 for details.
4. If the caller is running in a Unix System Services (USS) environment, set up the notification mechanism to allow hardware and software events to be propagated to the USS application. See “Setting up Event Notification for BCPii USS Applications” on page 14-6 for details.

After you have activated the BCPii address space, you need to know how to control the address space. See “BCPii Startup and Shutdown” on page 14-7 for details.
Setting up Connectivity to the Support Element

BCPii uses a low-level operating system connection to establish communication between an authorized application running on a z/OS image (LPAR) and the Support Element (SE) associated with the Central Processor Complex (CPC) that contains this z/OS image. You must configure the support element to permit these BCPii communications if BCPii services are required to be available by your installation.

Levels of Hardware BCPii Supports

The HWIBCPII address space, which supports the issuing of BCPii APIs from a z/OS image, runs on any level of hardware that supports the level of the z/OS operating system in which BCPii is included.

BCPii applications might need to perform hardware or software functions on CPCs other than the CPC on which the application is running. Such requests can be targeted to other System z hardware at a lower or higher hardware level than the local CPC, provided that these hardware levels are supported to coexist with the local CPC level.

The HWICMD service is only allowed to be targeted to at least a System z9® hardware level running on a particular microcode level. BCPii rejects the targeting of this service to any System z hardware level lower than System z9. See "HWICMD — Issue a BCPii Hardware Management Command" on page 14-10 for further information.

Consult this chart as a guide to what minimum levels of microcode are required on various hardware levels to run BCPii.

<table>
<thead>
<tr>
<th>Hardware Level</th>
<th>Minimum Microcode Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>z9®</td>
<td>G40965.133</td>
</tr>
<tr>
<td>z10</td>
<td>F85906.116</td>
</tr>
</tbody>
</table>

Each version of hardware has subtle or sometimes significant changes in the way information is displayed and saved in the support element. The examples serve as a guide only to where the actual definitions that need to be modified are located within the support element configuration windows.

Enable BCPii Communications on the Support Element

You need to enable cross-partition authority on the support element to allow the support element to accept the BCPii APIs flowing from the user application through the HWIBCPII address space. This setting controls whether a logical partition can issue a subset of control program instructions to other logical partitions activated on the same CPC.

Note: This setting must be selected on the local SE associated with the CPC of the image that the z/OS BCPii application is running on. It must also be selected for any other system for which BCPii communication is required.

To change this setting, perform the following steps on the HMC:
1. Select the CPC that is required.
3. Open the CPC Operational Customization task list.
4. Highlight the CPC icon.
5. Open the Change LPAR Security task, and the Change Logical Partition Security window displays.

6. Check the cross-partition authority checkbox for each image (LPAR) that you want to grant BCPii access. At a minimum, the image (LPAR) the BCPii address space is running needs to have this authority activated.

7. Select Save and Change.

See the HMC book and *System z9 Support Element Operations Guide* and *System z10™ Support Element Operations Guide* for more information regarding changing the support element settings.

Failure to set this properly on the local SE associated with the image of z/OS that is running BCPii results in a severe BCPii address space initialization failure. You cannot start the address space, and will receive communications error X'101' with a reason code of X'D4'. Failure to set this up properly on remote SEs to which you want to connect results in the same return code and reason code on the HWICONN service call.

**Note:** Make the same updates to all CPCs that you want BCPii to communicate with and not just the CPC from which the BCPii application is going to run on.

**Define the BCPii Community Name on the Support Element**

BCPii uses an SNMP community name to provide a level of security between the z/OS image that is executing the BCPii service and the support element itself.

An SNMP community is a logical relationship between an SNMP agent and an SNMP manager. The community has a name, and all members of a community have the same access privileges: they are either read-only (members can view configuration and performance information) or read-write (members can view configuration and performance information, and also change the configuration).

To add the BCPii community name definition to the SE configuration, perform the following steps on the HMC:

1. Select the CPC that is required.
3. Select the Console Actions view.
4. Select Support Element Settings.
5. Open the Customize API Settings.
6. Check the Enable SNMP APIs checkbox.
7. Consider checking the "Allow capacity change API requests" checkbox on a z10 or higher operation system if the installation is to allow a BCPii application to perform temporary capacity upgrades.
8. Make sure that the SNMP agent parameters are blank.
9. Add a BCPii community name. Click on Add. When a window is prompted, fill in the following fields:

   **Name** The actual SNMP community name. This value is a 1- to 16-character alphanumeric field. Because of restrictions with the security products on z/OS, the BCPii SNMP community name must not contain any lowercase characters. See "Community name defined in the security product for each CPC" on page 14-5 for more information about the SNMP community name.
Address
For BCPii, this address (sometimes referred to as a loop-back address) must be 127.0.0.1.

Network mask/Prefix
255.255.255.255.

Access Type
Read/write

10. Save the changes.


Failure to set this properly on the local SE associated with the image of z/OS that is running BCPii results in a severe BCPii failure and you cannot start the address space. Message HWI014I might be issued if the community name defined on the support element for the local CPC does not match the definition in the security product for the local CPC. See "Community name defined in the security product for each CPC" on page 14-5 for more information.

Note: Make the same updates to all CPCs that you want BCPii to communicate with.

Setting up Authority to Use BCPii

Given the nature of the BCPii APIs and the capabilities of a BCPii application to potentially modify vital hardware resources, a number of authority validations are performed for each BCPii requestor. A BCPii application needs to have program authority, general security product authority to be able to issue BCPii commands, authority to the particular resource that the application is trying to access, and a community name defined in the security product for each CPC to which communication is required.

Program Authority
BCPii applications must be program-authorized, meaning that one of the following must be true of the application:
- Running in supervisor state.
- Running in an authorized key with PSW key mask (PKM) between 0 and 7.
- Residing in an APF-authorized library.

General Security Product Authority
A BCPii application needs to have general authority to use BCPii. The profile HWI.APPLNAME.HWISERV in the FACILITY resource class controls which applications can use BCPii services. The security administrator must give at least read authority to this resource, in addition to granting authority to any specific resource that the application is attempting to access. In addition, BCPii requires that the FACILITY class to be RACLIST-specified.

This is an RACF® example to allow user JOE to use BCPii services in general:
RDEFINE FACILITY HWI.APPLNAME.HWISERV UACC(NONE)
PERMIT HWI.APPLNAME.HWISERV CLASS(FACILITY) ID(JOE) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH

Generic definitions may be created instead of specific users if the installation does not have specific definitions for every user.
Authority to the Particular Resource

A BCPii application needs to have authority to the particular resource that it is trying to access. That particular resource can be the CPC itself, an image (LPAR) on a particular CPC, or a particular capacity record on a particular CPC. BCPii needs a profile defined in the FACILITY resource class that represents the target of the particular BCPii request. The profile name required to be defined depends on the type of the particular resource required.

<table>
<thead>
<tr>
<th>Request Type</th>
<th>FACILITY Class Profile Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPC</td>
<td>HWI.TARGET.netid.nau where netid.nau represents the 3– to 17–character SNA name of the particular CPC.</td>
</tr>
<tr>
<td>Image</td>
<td>HWI.TARGET.netid.nau.imagename where netid.nau represents the 3– to 17–character SNA name of the particular CPC and imagename represents the 1– to 8–character LPAR name.</td>
</tr>
<tr>
<td>Capacity record</td>
<td>HWI.CAPREC.netid.nau.caprec where netid.nau represents the 3– to 17–character SNA name of the particular CPC and caprec represents an 8–character capacity record name.</td>
</tr>
<tr>
<td>Activation profiles</td>
<td>HWI.TARGET.netid.nau where netid.nau represents the 3– to 17–character SNA name of the particular CPC the activation profile is defined.</td>
</tr>
</tbody>
</table>

The access level required for the particular profile depends on the service that the BCPii application attempts to issue. See the BCPii API documentation in this chapter for specifics regarding the minimum access level required for each BCPii API service.

This is an RACF example to allow user JOE to have Connect, Event, List, and Query access to CPC NET1.CPC001, using community name XYZ123. See “Community name defined in the security product for each CPC” for more details.

```
RDEFINE FACILITY HWI.TARGET.NET1.CPC001 UACC(NONE) APPLDATA('XYZ123')
PERMIT HWI.TARGET.NET1.CPC001 CLASS(FACILITY) ID(JOE) ACCESS(READ)
SETROPTS RACLST(FACILITY) REFRESH
```

This is an RACF example to grant user with JOE Command, Connect, Event, List, and Query access to any image (LPAR) on NET1.CPC001:

```
RDEFINE FACILITY HWI.TARGET.NET1.CPC001.* UACC(NONE)
PERMIT HWI.TARGET.NET1.CPC001.* CLASS(FACILITY) ID(JOE) ACCESS(ALTER)
SETROPTS RACLST(FACILITY) REFRESH
```

Community name defined in the security product for each CPC

BCPii uses an SNMP community name to provide a minimal level of security between the z/OS image executing the BCPii service and the support element itself.

An SNMP community name is associated with a particular CPC. The same SNMP community name that was defined in the support element configuration for a particular CPC also must be defined in the security product for each CPC to which communication is required. This community name definition is extracted from the security product by BCPii and propagated to the support element. The support element validates that the community name passed by BCPii is correct before proceeding with the request. See Define the BCPii community name on the Support Element for information about how to define the community name on the SE or how to obtain the already-defined name.
To define the BCPIi community name in the security product, use the APPLDATA field with the CPC profile definition to associate a community name with a particular CPC.

The APPLDATA field for the BCPIi community name contains a 1– to 16–character alphanumeric field. Because of restrictions with the security products on z/OS, the BCPIi SNMP community name must not contain any lowercase characters.

This is an RACF example to assign a BCPIi community name of “XYZ123” to an existing CPC definition for CPC name NET1.CPC001:

```
RALTER FACILITY HWI.TARGET.NET1.CPC001 APPLDATA('XYZ123')
SETROPTS RACLIST(FACILITY) REFRESH
```

**Note:** A community name definition must be defined for at least the local CPC; otherwise, BCPIi cannot continue with initialization of its address space and BCPIi services are not available. This is accompanied by message HWI014I.

### BCPIi Configuration

The BCPIi address space is the bridge between a z/OS application and the support element. The address space can perform the following steps:

- Manage all application connections.
- Builds and receive all internal communication requests to the SE.
- Provide an infrastructure for storage required by callers and by the transport communicating with the SE.
- Provide diagnostic capabilities to help with BCPIi problem determination.
- Provide security authentication of requests.

The BCPIi address space is mandatory for any BCPIi API request. The system attempts to start the HWIBCPIi address space during IPL.

BCPIi requires the high-level qualifier SCEE RUN2 and high-level qualifier SCEE RUN data sets to be in the link list concatenation. IBM specifies these data sets in the default link list members (PROGxx) in z/OS 1.10 and higher. Failure to have these data sets in the link list results in BCPIi not being able to be started, accompanied by error message HWI009I that indicates that BCPIi could not load a required Language Environment® (LE) part.

BCPIi also includes a parmlib member into SYS1.PARMLIB for default CTRACE settings (CTIHWI00) when BCPIi initializes. See [z/OS MVS Diagnosis: Tools and Service Aids](https://www.ibm.com/support/docview.zhtml?docId=com.ibm.zos.doc/r/hwi599p2r0.pdf) for further information regarding CTRACE settings in BCPIi.

### Setting up Event Notification for BCPIi USS Applications

Applications running in a started procedure, batch, TSO or other non-UNIX System Services (USS) environment can use the HWIEVENT service and provide their own ENF exit that receives control when the application-requested events occur on the target CPC or image.

Applications running in a USS environment do not have normal ENF exit processing capabilities available and cannot readily listen for ENF signals. The Common Event Adapter (CEA) address space allows USS applications to be able to receive such event notifications. BCPIi provides several services that use the CEA functionality to deliver these same events to USS callers. See the documentation for the USS-only services of BCPIi ([HWIBeginEventDelivery — Begin Delivery of BCPIi Event Notifications](https://www.ibm.com/support/docview.zhtml?docId=com.ibm.zos.doc/r/hwi599p2r0.pdf) on page 14-90) [HWIEndEventDelivery — End Delivery of BCPIi Event](https://www.ibm.com/support/docview.zhtml?docId=com.ibm.zos.doc/r/hwi599p2r0.pdf).
The use of the CEA address space by BCPii requires some minor CEA setup before USS-only services of BCPii can work properly.

**CEA Address Space Setup**
The Common Event Adapter (CEA) address space must be active to allow the USS-only services of BCPii to operate. CEA has two modes of operation: minimum or full-function mode. If the USS-only services of BCPii are required to be available, CEA must be running in full-function mode. To activate full-function mode, a set of security product definitions are required. See [z/OS Planning for Installation](#) for more information about how to configure Common Event Adapter for full-function mode.

CEA, like BCPii, starts as part of a system IPL. It can be stopped and restarted as well. See [z/OS Planning for Installation](#) for more information.

**CEA ENF Security Configuration**
A USS BCPii application must be granted authority to listen to ENF68 events. With the CEA ENF controls, it is also possible to fine-tune which BCPii events a user is allowed to listen to.

This is an RACF example to give generic authority to the user id associated with a USS application authority to listen to any BCPii event:

```rsh
AU user_id OMVS(Uid(n))
SETROPTS GENERIC(SERVAUTH)
RDEFINE SERVAUTH CEA.CONNECT UACC(NONE)
RDEFINE SERVAUTH CEA.SUBSCRIBE.ENF_0068* UACC(NONE)
PERMIT CEA.CONNECT CLASS(SERVAUTH) ID(user_id) ACCESS(READ)
PERMIT CEA.SUBSCRIBE.ENF_0068* CLASS(SERVAUTH) ID(user_id) ACCESS(READ)
SETROPTS RACLIST(SERVAUTH) REFRESH
```

To give specific authority to only certain BCPii events, use the event qualifier as part of the profile name. The event qualifier maps to the event mask for ENF68 in the ENFREQ documentation in [z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG](#). Hardware events are in the form '03xx00yy' where xx is the event source ('01'x = CPC, and '02'x =image) and yy denotes the particular event.

This is an RACF example to allow user JOE authority to only receive events related to CPC command responses (CmdResp = '01'x):

```rsh
AU JOE OMVS(Uid(5))
RDEFINE SERVAUTH CEA.CONNECT UACC(NONE)
RDEFINE SERVAUTH CEA.SUBSCRIBE.ENF_006803010001 UACC(NONE)
PERMIT CEA.CONNECT CLASS(SERVAUTH) ID(JOE) ACCESS(READ)
PERMIT CEA.SUBSCRIBE.ENF_006803010001 CLASS(SERVAUTH) ID(JOE) ACCESS(READ)
SETROPTS RACLIST(SERVAUTH) REFRESH
```

**BCPii Startup and Shutdown**
The BCPii address space normally does not need to be started or shut down. BCPii initialization occurs during system IPL. If the configuration is correct, no further action is required. The address space remains active and ready to handle BCPii requests.
BCPii Address Space Does not Start up at IPL

If the HWIBCPII address space is not active after an IPL has been done, look for HWI* messages in the system log. Most of the time, these messages pinpoint the reason for the failure of BCPii to become active.

In most cases, the address space did not start for one of two main reasons:

1. The support element that controls the CPC that contains the image of z/OS on which BCPii is being started has the improper configuration. Make sure all the steps have been followed in “Setting up Connectivity to the Support Element” on page 14-2.

2. The community name of this local CPC is either not defined in the security product, or contains an incorrect value. This is accompanied by message HWI014I. See “Community name defined in the security product for each CPC” on page 14-5 for detailed information.

When these problems have been corrected, restart the BCPii address space. See “Restarting the HWIBCPII Address Space” on page 14-8 for more information.

Ending the HWIBCPII Address Space

The application of certain kinds of code maintenance or other unusual circumstances might require that the BCPii address space be stopped. To stop the BCPii address space, issue the STOP command for the BCPii address space: P HWIBCPII. In most cases, the address space ends normally. BCPii services are no longer available until the address space is restarted. See "z/OS MVS Initialization and Tuning Reference" for more information about the STOP HWIBCPII command.

If the STOP command fails to completely bring down the BCPii address space, you can issue the CANCEL command: C HWIBCPII. The address space then ends in a similar way to the STOP command. See "z/OS MVS Initialization and Tuning Reference" for more information about the CANCEL command.

If the CANCEL command still fails to completely bring down the BCPii, you can issue the FORCE command as a last resort: FORCE HWIBCPII. See "z/OS MVS Initialization and Tuning Reference" for more information about the FORCE command.

BCPii issues an ENF 68 broadcast to notify interested ENF listeners that BCPii services are no longer available. See "z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG" for more information regarding this ENF signal.

Restarting the HWIBCPII Address Space

After the BCPii address space has ended, it can be restarted. A procedure supplied by IBM in SYS1.PROCLIB allows the BCPii address space to be restarted. Issue the S HWISTART command to restart the HWIBCPII address space. When message HWI001I appears, confirming that BCPii is active, then BCPii requests might continue. See "z/OS MVS Initialization and Tuning Reference" for more information about the START HWISTART command.

BCPii issues an ENF 68 broadcast when the address space has completely initialized to notify interested ENF listeners that BCPii services are now available. See "z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG" for more information regarding this ENF signal.
BCPi Callable Services

You can use base control program internal interface (BCPi) services to connect an authorized z/OS application to System z configuration resources (such as CPC, image, capacity record, or activation profile data) and to allow that application to potentially modify these resources.

To use base control program internal interface (BCPi) services, 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 BCPii services. Each description includes a syntax diagram, parameter descriptions, return and reason code explanations with recommended actions. Return and reason codes are shown in hexadecimal and decimal with the associated equate symbols.

This chapter contains the following topics:
- “Syntax, Linkage and Programming Considerations” on page 14-9
- “HWICMD — Issue a BCPii Hardware Management Command” on page 14-10
- “HWICONN — Establish a BCPii Connection” on page 14-21
- “HWIDISC — Release a BCPii Connection” on page 14-28
- “HWIEVENT — Register for BCPii Events” on page 14-32
- “HWILIST — Retrieve HMC and BCPii Configuration-Related Information” on page 14-40
- “HWIQUERY — BCPii Retrieval of SE/HMC-Managed Objects Data” on page 14-49
- “HWISET — BCPii Set SE/HMC-Managed Objects Data” on page 14-69
- “HWIBeginEventDelivery — Begin Delivery of BCPii Event Notifications” on page 14-90
- “HWIEndEventDelivery — End Delivery of BCPii Event Notifications” on page 14-92
- “HWIManageEvents — Manage the List of BCPii Events” on page 14-94
- “HWIGetEvent — Retrieve Outstanding BCPii Event Notifications” on page 14-98

Syntax, Linkage and Programming Considerations

High level language (HLL) definitions are provided in the following languages:
- In C (HWICIC) in data set SYS1.SIEAHDRV.H. Miscellaneous C constants are defined in HWIZHAPI in the same data set.
- In Assembler (HWICIASM) in data set SYS1.MACLIB. Miscellaneous C constants are defined in HWIC2ASM in the same data set.

Calling formats

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

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

Assembler Call macro
CALL routine_name,(parm1,parm2,...return_code),VL

Assembler considerations

Callers must also use the following linkage conventions:
Register 1 must contain the address of a parameter list that is a list of consecutive words, each containing the address of a parameter to be passed. The last word in this list must have a 1 in the high-order (sign) bit.

Register 13 must contain the address of an 18-word save area.

Register 14 must contain the return address.

Register 15 must contain the entry point address of the service being called.

If the caller is running in AR ASC mode, access registers 1, 13, 14, and 15 must all be set to zero.

On return from the service, general and access registers 2 through 14 are restored (registers 0, 1 and 15 are not restored).

**Linkage considerations**

Link-edit your object code with the linkable stub routine HWICSS from SYS1.CSSLIB or LOAD and CALL the service.

**HWICMD — Issue a BCPIi Hardware Management Command**

Call the HWICMD service to perform a command against an HMC-managed object that is associated with central processor complexes (CPCs), CPC images (LPARs) and capacity records.

**Description**

**Environment**

The requirements for the callers are:

- **Minimum authorization:** One of the following: PKM allowing key 0-7, supervisor state, or APF-authorized
- **Dispatchable unit mode:** Task
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 31-bit
- **ASC mode:** Primary or access register (AR)
- **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**

See "Syntax, Linkage and Programming Considerations" on page 14-9 for details about how to call BCPIi services in the various programming languages.

The microcode level that supports the command service call (HWICMD) of BCPIi is required to be installed on the target CPC. See the HWI_CMD_NOT_SUPPORT_WARNING return code in "HWICONN — Establish a BCPIi Connection" on page 14-2 for more information.

**Restrictions**

BCPIi does not allow any HWICMD to be targeted to a CPC that is earlier than a z9 platform.
Authorization

The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

The client application must have at least control access to the following SAF-protected FACILITY class resource profiles:
- HWI.TARGET.netid.nau for ConnectTokens that represents a CPC connection
- HWI.TARGET.netid.nau.imagename for ConnectTokens that represents an image connection

Note: BCPII require the FACILITY class to be RACLIST-specified.

Syntax

Write the call as shown on the syntax diagram. Code all parameters on the CALL statement in the order shown.

```
CALL HWICMD
       (ReturnCode,
        ConnectToken,
        CmdType,
        CmdParm_Ptr,
        DiagArea)
```

Parameters

The parameters are explained as follows:

**ReturnValue**

- Returned parameter
  - Type: integer
  - Length: 4 bytes

ReturnCode contains the return code from the service.

**ConnectToken**

- Supplied parameter
  - Type: character string
  - Length: 16 bytes

ConnectToken specifies the connect token that this command is executed against. A ConnectToken represents a logical connection between the application and a CPC or an image, and is returned as an output parameter on the HWICONN service call.

The ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call.

**CmdType**

- Supplied parameter
  - Type: integer
  - Length: 4 bytes

CmdType specifies the type of the requested command.

See the following publications for more information about how the various commands operate, what inputs are required, and what outputs are expected:
### Constant in Hexadecimal (Decimal) Equate Symbol | Description
--- | ---
1 (1) HWI_CMD_ACTIVATE | Activate request to start target systems.  
**Note:** The input connect token represents a CPC connection or an image connection. This command cannot be issued specifying a connect token that represents either the local CPC or the local image.

2 (2) HWI_CMD_DEACTIVATE | Deactivate request to close down target systems.  
**Note:** The input connect token represents a CPC connection or an image connection. This command cannot be issued specifying a connect token that represents either the local CPC or the local image.

3 (3) HWI_CMD_HWMSG | Hardware messages request.  
**Note:** The input connect token must only represent a CPC connection.

4 (4) HWI_CMD_CBU | Capacity backup CPC feature operation.  
**Note:** The input connect token must only represent a CPC connection.

5 (5) HWI_CMD_OOCOD | On/Off capacity on demand request.  
**Note:** The input connect token must only represent a CPC connection.

6 (6) HWI_CMD_PROFILE | Access CPC activation profiles.  
**Note:** The input connect token must only represent a CPC connection.

7 (7) HWI_CMD_RESERVE | Set exclusive CPC control.  
**Note:** The input connect token must only represent a CPC connection.

8 (8) HWI_CMD_SYSRESET | System reset request for target systems. See Cmdtype HWI_CMD_SYSRESET_IPLT for the latest version of the Sysreset command.  
**Note:** The input connect token must only represent an image connection. This command cannot be issued specifying a connect token that represents the local image.

9 (9) HWI_CMD_START | Start request for all CPs on target systems.  
**Note:** The input connect token must only represent an image connection. This command cannot be issued specifying a connect token that represents the local image.

A (10) HWI_CMD_STOP | Stop request for all CPs on target systems.  
**Note:** The input connect token must only represent an image connection. This command cannot be issued specifying a connect token that represents the local image.

B (11) HWI_CMD_PSWRESTART | Restart request for one CPC on target system. The first CP that is found to be in the correct state is reset.  
**Note:** The input connect token must only represent an image connection. This command cannot be issued specifying a connect token that represents the local image.

C (12) HWI_CMD_OSCMD | Send operating system command request.  
**Note:** The input connect token must only represent an image connection.

D (13) HWI_CMD_LOAD | Load request to IPL target operating systems.  
**Note:** The input connect token must only represent an image connection. This command cannot be issued specifying a connect token that represents the local image.
<table>
<thead>
<tr>
<th>Constant in Hexadecimal (Decimal)</th>
<th>Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>(14) HWI_CMD_TEMP_CAP</td>
<td>Addition or removal of temporary capacity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The input connect token must only represent a CPC connection.</td>
</tr>
<tr>
<td>F</td>
<td>(15) HWI_CMD_SYSRESET_IPLT</td>
<td>System reset request for target systems with IPL token correlation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This is an enhanced version of HWI_CMD_SYSRESET.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
</tbody>
</table>

**CmdParm_Ptr**
- Supplied parameter
- Type: pointer
- Length: 4 bytes

CmdParm_Ptr specifies the address of the command parameter that contains a structure of the input parameters for the requested command.

Take the following action according to the different conditions:
- For all optional parameters, callers are required to initialize the parameters to zero for BCPii to interpret them as null parameters unless otherwise specified.
- For commands with one or more required parameters and also with one or more optional parameters, callers are required to initialize each optional parameters to zero if they require BCPii to take the default action for that parameter.
- For commands that have only optional parameters, callers can initialize the CmdParm_Ptr to zero if they require BCPii to take the default action for all parameters.
- For commands that have no parameters, the CmdParm_Ptr is ignored.

All string type parameters are required to be padded with trailing blanks unless otherwise specified.

<table>
<thead>
<tr>
<th>CmdType : HWI_CMD_</th>
<th>CmdParm</th>
<th>Parameters in Structure</th>
<th>Parameter Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVATE</td>
<td>HWI_CMD_ACT_PARM</td>
<td>ActProfName</td>
<td>A 16-character activation profile name padded with trailing blanks (Optional.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ForceType</td>
<td>A 4-byte integer (optional, the default is FORCE):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1 – means Force YES (HWI_CMD_FORCE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 – means Force NO (HWI_CMD_NOFORCE)</td>
</tr>
<tr>
<td>DEACTIVATE</td>
<td>HWI_CMD_DEACT_PARM</td>
<td>ForceType</td>
<td>A 4-byte integer (optional, the default is FORCE):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1 – means Force YES (HWI_CMD_FORCE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 – means Force NO (HWI_CMD_NOFORCE)</td>
</tr>
<tr>
<td>CmdType</td>
<td>CmdParm</td>
<td>Parameters in Structure</td>
<td>Parameter Values</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------</td>
<td>------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HWMSG</td>
<td>HWI_CMD_HWMSG_PARM</td>
<td>HWMSGType</td>
<td>A 4-byte integer (required):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1 – means REFRESH (HWI_CMD_HWMSG_REFRESH)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 – means DELETE (HWI_CMD_HWMSG_DELETE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HWMSGTimestamp</td>
<td>A 32-character string padded with trailing blanks</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Required only for HWMSGType = HWI_CMD_HWMSG_DELETE</td>
</tr>
<tr>
<td>CBU</td>
<td>HWI_CMD_CBU_PARM</td>
<td>CBUType</td>
<td>A 4-byte integer (required):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1 – means ACTIVATE (HWI_CMD_ACT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 – means UNDO (HWI_CMD_UNDO)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ActivateType</td>
<td>A 4-byte integer (required only for CBUType = HWI_CMD_ACT):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1 – means REAL CBU (HWI_CMD_ACTIVATE_REAL)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 – means TEST CBU (HWI_CMD_ACTIVATE_TEST)</td>
</tr>
<tr>
<td>OOCOD</td>
<td>HWI_CMD_OOCOD_PARM</td>
<td>OOCODType</td>
<td>A 4-byte integer (required):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1 – means ACTIVATE (HWI_CMD_ACT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 – means UNDO (HWI_CMD_UNDO)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OOCODActivateOrderNumber</td>
<td>Required for OOCODType = HWI_CMD_OOCOD</td>
</tr>
<tr>
<td>PROFILE</td>
<td>HWI_CMD_PROFILE_PARM</td>
<td>ProfileType</td>
<td>A 4-byte integer (required):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1 – means IMPORT (HWI_CMD_PROFILE_IMPORT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 – means EXPORT (HWI_CMD_PROFILE_EXPORT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AreaNumber</td>
<td>A 2-byte integer area number is required and must be in the range of 1 to 4.</td>
</tr>
<tr>
<td>RESERVE</td>
<td>HWI_CMD_RESERVE_PARM</td>
<td>ReserveType</td>
<td>A 4-byte integer (required):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1 – means ADD (HWI_CMD_RESERVE_ADD)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 – means DELETE (HWI_CMD_RESERVE_DELETE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ApplName</td>
<td>An 8-character application name padded with trailing blanks (required).</td>
</tr>
<tr>
<td>SYSRESET</td>
<td>HWI_CMD_SYSRESET_PARM</td>
<td>ResetType</td>
<td>A 4-byte integer (required):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1 – means NORMAL (HWI_CMD_RESET_NORMAL)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 – means CLEAR (HWI_CMD_RESET_CLEAR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ForceType</td>
<td>A 4-byte integer (optional, the default is FORCE):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1 – means Force YES (HWI_CMD_FORCE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 – means Force NO (HWI_CMD_NOFORCE)</td>
</tr>
<tr>
<td>START</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>STOP</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>PSWRESTART</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>CmdType : HWI_CMD_</td>
<td>CmdParm</td>
<td>Parameters in Structure</td>
<td>Parameter Values</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>----------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OSCMD</td>
<td>HWI_CMD_OSCMD_PARM</td>
<td>PriorityType</td>
<td>A 4-byte integer (required):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 – means Priority (HWI_CMD_PRIORITY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 – means Non Priority (HWI_CMD_NONPRIORITY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OSCMDString</td>
<td>An 126-null-terminated character operating system command string (required).</td>
</tr>
<tr>
<td>LOAD</td>
<td>HWI_CMD_LOAD_PARM</td>
<td>LoadAddr</td>
<td>A 4-character string consisting only of hexadecimal characters identifying the device address to be used when performing the load (optional).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LoadParm</td>
<td>An 8-character string as determined by the operating system being loaded (optional).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ForceType</td>
<td>A 4-byte integer (optional, the default is FORCE):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 – means Force YES (HWI_CMD_FORCE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 – means Force NO (HWI_CMD_NOFORCE)</td>
</tr>
<tr>
<td>TEMPCAP</td>
<td>HWI_CMD_TEMPCAP_Parm</td>
<td>TEMPCAPType</td>
<td>A 4-byte integer (required):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 – means Add (HWI_CMD_TEMPCAP_ADD)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 – means Remove (HWI_CMD_TEMPCAP_REMOVE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XML_Ptr</td>
<td>A character string pointer points to the address of the XML information that illustrates the markup used to perform activation of the temporary capacity (required).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XML_Size</td>
<td>A 4-byte integer (required). Length in bytes of the XML that the XML_Ptr points to.</td>
</tr>
<tr>
<td>SYSRESET</td>
<td>HWI_CMD_SYSRESET__IPLT_PARM</td>
<td>ResetType</td>
<td>A 4-byte integer (required):</td>
</tr>
<tr>
<td>_IPLT</td>
<td></td>
<td></td>
<td>1 – means NORMAL (HWI_CMD_RESET_NORMAL)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 – means CLEAR (HWI_CMD_RESET_CLEAR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ForceType</td>
<td>A 4-byte integer (optional, the default is FORCE):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 – means Force YES (HWI_CMD_FORCE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 – means Force NO (HWI_CMD_NOFORCE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPL-Token_Ptr</td>
<td>A character string pointer that specifies the address of the IPL token used to correlate a SYSRESET with other outstanding HMC-related activities. This ensures that this SYSRESET is operating with the same IPL instance as when the IPL-Token was retrieved (required).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPL-Token_Len</td>
<td>A 4-byte integer. Length in bytes of the IPL token to which the IPL-Token_Ptr points (required).</td>
</tr>
</tbody>
</table>
DiagArea contains diagnostic data to help determine the cause of a failure from
the service. For many return codes, the DiagArea can contain further
information to help determine the cause of the failure. See the descriptions of
different return codes for a partial list of data returned in this area.

**Note:** For all environmental errors (with return code X’F00’ and higher), the
DiagArea might not be filled in, and the data returned in the area should
be ignored.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag_Index</td>
<td>32-bit integer</td>
<td>The array index to the parameter field that causes the error.</td>
</tr>
<tr>
<td>Diag_Key</td>
<td>32-bit integer</td>
<td>The constant value represents the field that causes the error.</td>
</tr>
<tr>
<td>Diag_Actual</td>
<td>32-bit integer</td>
<td>The incorrect actual value that is specified.</td>
</tr>
<tr>
<td>Diag_Expected</td>
<td>32-bit integer</td>
<td>The expected value to be used.</td>
</tr>
<tr>
<td>Diag_CommErr</td>
<td>32-bit integer</td>
<td>The returned code that is returned from the console application API or the BCPii transport layer.</td>
</tr>
<tr>
<td>Diag_text</td>
<td>Character (12)</td>
<td>Additional diagnostic information in text format.</td>
</tr>
</tbody>
</table>

**ABEND Codes**
The call might result in an abend X’042’ with a reason code of X'0001yyyy' because of one of the following reasons:

<table>
<thead>
<tr>
<th>yyyy</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The parameters passed by the caller are not in the primary address space.</td>
</tr>
<tr>
<td>0001</td>
<td>The parameters passed by the caller are not accessible.</td>
</tr>
<tr>
<td>0002</td>
<td>The number of parameters passed by the caller is not correct.</td>
</tr>
</tbody>
</table>

**Return Codes**
When the service returns control to the caller, GPR 15 and the ReturnCode contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return Code in Hexadecimal</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 HWI_OK</td>
<td></td>
<td><strong>Meaning:</strong> Successful completion.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None.</td>
</tr>
<tr>
<td>Return Code in Hexadecimal</td>
<td>Equate Symbol</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| 100 HWI_CONNECT_TOKEN_INV  |               | **Meaning:** Program error. The specified connect token is not valid. This return code indicates that one of the following conditions has occurred:  
  - The connect token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken.  
  - The connect token does not represent an active connection. The connection specified might have already been disconnected using the HWIDISC service call.  
  - The connect token is not associated with the caller’s address space. The ConnectToken specified is associated with a different address space than the caller of this service call.  
  **Action:** Check for probable coding error. |
| 101 HWI_COMMUNICATION_ERROR|               | **Meaning:** A communication error is detected. The hardware management console application API (HWMCA) or the BCPIi transport layer has returned with a failing return code.  
  **Action:** See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPIi transport layer.  
  HWMCA API and BCPIi transport return codes are provided in Appendix A, "BCPIi Communication Error Reason Codes," on page A-1. |
| 102 HWI_DIAGAREA_INV       |               | **Meaning:** Program error. The DiagArea is not accessible.  
  **Action:** Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field. |
| 103 HWI_CONNECT_TOKEN_INACTIVE|             | **Meaning:** The specified connect token is no longer valid. The connection has been disconnected or it is in the progress of being disconnected.  
  **Action:** Check for probable coding error. Verify that the specified connect token is still active. If connectivity to the targeted CPC connection no longer exists, all connections associated with that CPC will no longer have a connect token that can be used. |
<table>
<thead>
<tr>
<th>Return Code in Hexadecimal</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 602 HWI_CMDTYPE_INV        |               | **Meaning:** Program error. The requested CMDTYPE specified in the call is not valid. The system rejects the service call. This return code indicates that one of the following conditions has occurred:  
  - The CmdType specified is not in the acceptable value range of possible command types. The Diag_Text indicates this error with the text of 'Invalid Cmd'.  
  - The CmdType specified applies only to CPC connections, but the ConnectToken specified represents an image connection. The Diag_Text indicate this error with the text of 'Mismatch'.  
  - The CmdType specified applies only to image connections, but the ConnectToken specified represents a CPC connection. The Diag_Text indicates this error with the text of 'Mismatch'.  
  **Action:** Check for probable coding error. Verify that the specified CmdType is in the acceptable value range. See the CmdType parameter section to verify that the specified connect token is applied for the requested command. See the DiagArea for further diagnostic information. |
| 603 HWI_CMDPARM_INV        |               | **Meaning:** Program error. This return code indicates that one of the following conditions has occurred:  
  - Required parameters are missing.  
  - One or more parameters specified are not valid.  
  **Action:** Check for probable coding error. See the DiagArea for additional diagnostic information. The Diag_Index specifies the value of the CmdType parameter. The Diag_Text specifies the name of the parameter in the CmdParm structure. Note that the name might be abbreviated because of the limited size of the Diag_Text field. |
| 604 HWI_CMD_TARGET_DEST_NOT_ALLOWED | | **Meaning:** Program error. Certain commands are not allowed to be targeted to the same CPC and image on which the BCPii application is currently running. Such commands can cause the local system to be inoperable. Commands that cannot target the local CPC are Hwi_Cmd_Activate and Hwi_Cmd_Deactivate. Commands that cannot target the local Image include the following:  
  - Hwi_Cmd_Activate  
  - Hwi_Cmd_Deactivate  
  - Hwi_Cmd_Load  
  - Hwi_Cmd_PswRestart  
  - Hwi_Cmd_Start  
  - Hwi_Cmd_Stop  
  - Hwi_Cmd_Sysreset  
  - Hwi_Cmd_Sysreset_IPLT  
  **Action:** Validate the name of the target represented by the input connect token. If the target is correct, the command can only be issued from another CPC for a CPC-related command or from another image for an image-related command. BCPii does not allow this command to be executed against the same target where it is currently executing on. |
<table>
<thead>
<tr>
<th>Return Code in Hexadecimal</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| HWI_CMDPARM_INACCESSIBLE   | **Meaning:** Program error. The CmdParm data area cannot be accessed. This return code indicates that one of the following conditions has occurred:  
  - The CmdParm data area is either partially or completely not accessible by the application, or BCPIi, or both.  
  - The CmdParm data area can be too small.  
  **Action:** Check for probable coding error. Validate that the CmdParm_Ptr points to a data area where the CmdParm is and that the data area is accessible. See the DiagArea for further diagnostic information. |
| HWI_CMDTYPE_NOT_SUPPORTED  | **Meaning:** The targeted hardware of the HWICMD request does not recognize the type of command being requested.  
  **Action:** Verify that the targeted hardware is at a level that supports the type of command being issued. |
| HWI_CMD_NOT_SUPPORTED      | **Meaning:** HWICMD is not supported with the current microcode level (MCL) installed on the target CPC, or the target CPC is at a lower hardware level than HWICMD supports (BCPIi requires the target of an HWICMD to be at least at the z9 hardware level). The warning return code, HWI_CMD_NOT_SUPPORTED.WARNING, should have been returned on the previous HWICONN service call when the requested connect token was created to establish a connection to the CPC. See the return code section in "HWICONN — Establish a BCPIi Connection" on page 14-21 for more information.  
  **Action:** Install the MCL that supports HWICMD on the target CPC or refrain from issuing HWICMD with a target older than the z9 hardware level. See the HWI_CMD_NOT_SUPPORTED_WARNING return code in the HWICONN section for the microcode level/engineering change (MCL/EC) that is required for HWICMD service call. |
| HWI_NOT_AVAILABLE          | **Meaning:** BCPIi services are not available, and the system rejects the service request.  
  **Action:** Notify the system programmer to start the BCPIi address space and try the request again. See "Restarting the HWIBCPII Address Space" on page 14-8 about how to start the BCPIi address space.  
  Programs can also listen to ENF68 to determine when BCPIi services are available. See [z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG](https://www.ibm.com/support/docview.zhtml?docname=ibmxr3g) for how to listen for BCPIi activation messages. |
| HWI_AUTH_FAILURE           | **Meaning:** The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.  
  **Action:** Check the calling program for a probable coding error. |
Return Code in Hexadecimal Equate Symbol | Meaning and Action
---|---
F02 HWI_NO_SAF_AUTH | **Meaning:** The user does not have correct SAF authorization for the request.

**Action:** Check for probable error. Consider one or more of the following possible actions:
- Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV.
- Define control access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for a CPC connection.
- Define control access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for an image connection.
- Ensure that the referenced Facility Class Profile is RACLIST-specified.

F03 HWI_INTERRUPT_STATUS_INV | **Meaning:** The calling program is disabled. The system rejects this service request.

**Action:** Check the calling program for a probable coding error.

F04 HWI_MODE_INV | **Meaning:** The calling program is not in task mode. The system rejects this service request.

**Action:** Check the calling program for a probable error.

F05 HWI_LOCKS_HELD | **Meaning:** The calling program is holding one or more locks. The system rejects this service request.

**Action:** Check the calling program for a probable coding error.

F06 HWI_UNSUPPORTED_RELEASE | **Meaning:** The system level does not support this service. The system rejects this service request.

**Action:** Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again.

FFF HWI_UNEXPECTED_ERROR | **Meaning:** System error. The service that was called encountered an unexpected error. The system rejects the service call.

**Action:** In many cases, BCPii has taken an abend to gather further diagnostic information. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.

**Example**
In the pseudocode example, the caller issues a call to activate an activation profile.

```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
| CmdType = HWI_CMD_ACTIVATE;
| HWI_CmdTypeParm.Aprofname = 'My_Aprofilename';
| CmdParm_Ptr = addr(HWI_CmdParm);
| CALL HWICMD (ReturnCode, ConnectToken, CmdType, 
|              CmdParm_Ptr, DiagArea)
|   |
```
HWICONN — Establish a BCPii Connection

Call the HWICONN service to establish a logical connection between the application and a central processor complex (CPC), a CPC image (LPAR), a capacity record, or different types of activation profiles. This facilitates subsequent services to perform operations related to that CPC, image, capacity record, or activation profile.

Note: A connection remains active until a Disconnect service call (HWIDISC) is invoked, or a loss of connectivity to the associated CPC has been detected by BCPii, or the address space of the caller is terminated. When the address space of the caller is terminated, the connection is disconnected implicitly by BCPii.

Description

Environment
The requirements for the callers are:

Minimum authorization: One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
Dispatchable unit mode: Task
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 31-bit
ASC mode: Primary or access register (AR)
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
See “Syntax, Linkage and Programming Considerations” on page 14-9 for details about how to call BCPii services in the various programming languages.

Restrictions
None.

Authorization
The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

The client application must also have at least one of the following access:

- Read access to the SAF-protected FACILITY class resource
  HWI.TARGET.netid.nau for HWI_CPC, HWI_RESET_ACT_PROF,
  HWI_IMAGE_ACT_PROF, or HWI_LOAD_ACT_PROF connections
- Read access to the SAF-protected FACILITY class resource
  HWI.TARGET.netid.nau.imagename for HWI_IMAGE connections
- Read access to the SAF-protected FACILITY class resource
  HWI.CAPREC.netid.nau.caprecid for HWI_CAPREC connections

Note: BCPii require the FACILITY class to be RACLIST-specified.
Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL HWICONN
    (ReturnCode,
    InConnectToken,
    OutConnectToken,
    ConnectType,
    ConnectTypeValue_Ptr,
    DiagArea)
```

Parameters
The parameters are explained as follows:

**ReturnCode**
- Returned parameter
  - Type: integer
  - Length: 4 bytes

  ReturnCode contains the return code from the service.

**InConnectToken**
- Supplied parameter
  - Type: character string
  - Length: 16 bytes

  InConnectToken represents a connect token that was returned by a previous HWICONN HWI_CPC invocation. For image, capacity record, or activation profile connections, the input connect token must represent an active CPC connection.

  The ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call.

  InConnectToken is not relevant to a connect type of HWI_CPC, and it is ignored.

**OutConnectToken**
- Returned parameter
  - Type: character string
  - Length: 16 bytes

  OutConnectToken returns a connect token that uniquely represents a connection to BCPii. This parameter can be used as input on subsequent BCPii invocations to identify which connection the service wants to communicate.

  A connect token returned for an HWI_CPC connection can be specified on subsequent services to perform operations against this particular CPC, or on a subsequent HWICONN as the InConnectToken parameter when attempting a connection to a particular image (LPAR), capacity record (CAPREC), or activation profile.

  Likewise, a connect token returned for an HWI_IMAGE or HWI_CAPREC connection can be specified on subsequent services to perform operations against this particular image (LPAR) or capacity record (CAPREC) respectively.
A connect token returned for an HWI_RESET_ACT_PROF, HWI_IMAGE_ACT_PROF, or HWI_LOAD_ACT_PROF connection can be specified on subsequent HWIQUERY or HWISET service calls to query or set specific values associated with the specified Reset, Image, or Load activation profile respectively.

**ConnectType**

Supplied parameter
- Type: integer
- Length: 4 bytes

ConnectType specifies the type of connection to be established.

<table>
<thead>
<tr>
<th>Constant in Hexadecimal (Decimal) Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1) HWI_CPC</td>
<td>Requests to establish a connection to a target CPC that the application is to communicate with.</td>
</tr>
<tr>
<td>2 (2) HWI_IMAGE</td>
<td>Requests to establish a connection to an image of a CPC that the application is to communicate with. The input connect token must represent an active CPC connection.</td>
</tr>
<tr>
<td>3 (3) HWI_CAPREC</td>
<td>Requests to establish a connection to a capacity record of a CPC that the application is to communicate with. The input connect token must represent an active CPC connection.</td>
</tr>
<tr>
<td>4 (4) HWI_RESET_ACT_PROF</td>
<td>Requests to establish a connection to a reset activation profile associated with a particular CPC. The input connect token must represent an active CPC connection.</td>
</tr>
<tr>
<td>5 (5) HWI_IMAGE_ACT_PROF</td>
<td>Requests to establish a connection to an image activation profile associated with a particular CPC. The input connect token must represent an active CPC connection.</td>
</tr>
<tr>
<td>6 (6) HWI_LOAD_ACT_PROF</td>
<td>Requests to establish a connection to a load activation profile associated with a particular CPC. The input connect token must represent an active CPC connection.</td>
</tr>
</tbody>
</table>

**ConnectTypeValue_Ptr**

Supplied parameter
- Type: pointer
- Length: 4 bytes

ConnectTypeValue_Ptr specifies the address of the name value of the requested target to be connected to.
The type of connection determines what value is required.

<table>
<thead>
<tr>
<th>Connect Types</th>
<th>Values to be specified</th>
</tr>
</thead>
</table>
| HWI_CPC            | • A 17-character network address (sometimes referred to as the SNA address) that uniquely represents a CPC in the attached process control network. The network address should be in the form of a 1- through 8-character network identifier (netid), followed by a period, and then followed by a 1- through 8-character network addressable unit (NAU) name. The network address should be padded with trailing blanks if the total string length of the network address is less than 17 characters. Example: net1.cpc01  
  **Note:** netid.nau is 1- to 17-character symbolic NAU name. The network ID and name of a resource must both begin with a letter (A-Z), @, #, or $. The remaining characters can be letters (A-Z), numbers (0-9), @, #, or $.  
  • An '*' is a special value that can also be specified with this ConnectType. If specified, this allows the application to connect to the local host CPC without having to know the network address of the local host CPC (netid.nau).  
  **Note:** An HWILIST HWI_LIST_CPCS operation returns a list of CPCs available to be connected to in the form of netid.nau. |
| HWI_IMAGE          | An 8-character image name padded with trailing blanks.                                    |
| HWI_CAPREC         | An 8-character capacity record (CAPREC) name padded with trailing blanks.                 |
| HWI_RESET_ACT_PROF | A 16-character alphanumeric (0-9, A-Z) reset activation profile name padded with trailing blanks. |
| HWI_IMAGE_ACT_PROF | A 16-character alphanumeric (0-9, A-Z) image activation profile name padded with trailing blanks. |
| HWI_LOAD_ACT_PROF  | A 16-character alphanumeric (0-9, A-Z) load activation profile name padded with trailing blanks. |

**DiagArea**

- Returned parameter
- **Type:** character
- **Length:** 32 bytes

DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

**Note:** For all environmental errors (with return code X'F00' and higher), the DiagArea might not be filled in, and the data returned in the area should be ignored.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag_Index</td>
<td>32-bit integer</td>
<td>The array index to the parameter field that causes the error.</td>
</tr>
<tr>
<td>Diag_Key</td>
<td>32-bit integer</td>
<td>The constant value represents the field that causes the error.</td>
</tr>
<tr>
<td>Diag_Actual</td>
<td>32-bit integer</td>
<td>The incorrect actual value that is specified.</td>
</tr>
<tr>
<td>Diag_Expected</td>
<td>32-bit integer</td>
<td>The expected value to be used.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Field Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Diag_CommErr</td>
<td>32-bit integer</td>
<td>The returned code that is returned from the console application API or the BCPii transport layer.</td>
</tr>
<tr>
<td>Diag_text</td>
<td>character (12)</td>
<td>Additional diagnostic information in text format.</td>
</tr>
</tbody>
</table>

**ABEND Codes**

The call might result in an abend X'042' with a reason code of X'0002yyyy' due to one of the following reasons:

<table>
<thead>
<tr>
<th>yyyy</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The parameters passed by the caller are not in the primary address space.</td>
</tr>
<tr>
<td>0001</td>
<td>The parameters passed by the caller are not accessible.</td>
</tr>
<tr>
<td>0002</td>
<td>The number of parameters passed by the caller is not correct.</td>
</tr>
</tbody>
</table>

**Return Codes**

<table>
<thead>
<tr>
<th>Return Code in Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 HWI_OK</td>
<td><strong>Meaning:</strong> Successful completion. <strong>Action:</strong> None.</td>
</tr>
<tr>
<td>4 HWI_CMD_NOT_SUPPORTED_WARNING</td>
<td><strong>Meaning:</strong> Successful completion. This warning return code is informational. <strong>Action:</strong> Install the MCL/EC that supports HWICMD for the target CPC. The required MCL/EC are G40965.133 for a z9 CPC, and F85906.116 for a z10 CPC.</td>
</tr>
<tr>
<td>100 HWI_CONNECT_TOKEN_INV</td>
<td><strong>Meaning:</strong> Program error. The specified input connect token is not valid. This return code indicates that one of the following conditions has occurred: <strong>Action:</strong> Check for probable coding error.</td>
</tr>
<tr>
<td></td>
<td>• The input connect token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken.</td>
</tr>
<tr>
<td></td>
<td>• The input connect token does not represent an active connection. The connection specified might have already been disconnected by the HWIDISC service call, or have been implicitly disconnected by BCPii because of loss of connectivity with the target CPC.</td>
</tr>
<tr>
<td></td>
<td>• The input connect token is not associated with the address space of the caller. The InConnectToken specified is associated with a different address space than the caller of this service call.</td>
</tr>
<tr>
<td>Return Code in Hexadecimal Equate Symbol</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| 101 HWI_COMMUNICATION_ERROR            | **Meaning:** A communication error is detected. The hardware management console application API (HWMCA) or the BCPii transport layer has returned with a failing return code.  
**Action:** See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPii transport layer.  
HWMCA API and BCPii transport return codes are provided in Appendix A, “BCPii Communication Error Reason Codes,” on page A-1. |
| 102 HWI_DIAGAREA_INV                   | **Meaning:** Program error. The DiagArea is not accessible.  
**Action:** Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field. |
| 103 HWI_CONNECT_TOKEN_INACTIVE         | **Meaning:** The specified connect token is no longer valid. The connection has been disconnected or it is in the progress of being disconnected.  
**Action:** Check for probable coding error. Verify that the specified connect token is still active. If connectivity to the targeted CPC connection no longer exists, all connections associated with that CPC will no longer have a connect token that can be used. |
| 201 HWI_CONNTYPE_INV                   | **Meaning:** Program error. The connection type specified in the call is not valid. The system rejects the service call.  
**Action:** Check for probable coding error. Validate that the conntype value passed to the service is one of the accepted values. |
| 202 HWI_CONNTYPE_VALUE_INV             | **Meaning:** Program error. The connection name specified in the call is not valid. The specified connection name is not syntactically valid, it does not exist, or it is currently not available. The system rejects the service call.  
**Action:** Check for probable coding error. Verify that the connection name is syntactically correct, valid in the current HMC configuration, and currently available. |
| 203 HWI_CONNTYPE_VALUE_INACCESSIBLE    | **Meaning:** Program error. The connection type value data area is either partially or completely inaccessible by the application, or the Base Control Program internal interface (BCPii) address space, or both.  
**Action:** Check for probable coding error. Verify that the ConnectTypeValue_Ptr points to a data area where the connect type value is, and make sure that the data area is accessible. |
| 204 HWI_MAX_CONNECTIONS_REACHED        | **Meaning:** The number of connections has reached the maximum number of system-wide connections (512) that BCPii permits, or BCPii has run out of system resources to satisfy the HWICONN request, or both.  
**Action:** Disconnect connections that are no longer needed, and try the request again. |
<table>
<thead>
<tr>
<th>Return Code in Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| F00 HWI_NOT_AVAILABLE                   | **Meaning**: BCPIi services are not available, and the system rejects the service request.  
**Action**: Notify the system programmer to start the BCPIi address space and try the request again. See [“Restarting the HWIBCPII Address Space” on page 14-8](#) about how to start the BCPIi address space.  
Programs can also listen to ENF68 to determine when BCPIi services are available. See [z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG](#) for how to listen for BCPIi activation messages. |
| F01 HWI_AUTH_FAILURE                    | **Meaning**: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.  
**Action**: Check the calling program for a probable coding error. |
| F02 HWI_NO_SAF_AUTH                     | **Meaning**: The user does not have correct SAF authorization for the request.  
**Action**: Check for probable error. Consider one or more of the following possible actions:  
- Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV.  
- Define read access authorization to the FACILITY class resource profile HWITARGET.netid.nau for a CPC or activation profile connection.  
- Define read access authorization to the FACILITY class resource profile HWITARGET.netid.nau.imagename for an image connection.  
- Define read access authorization to the FACILITY class resource profile HWICAPREC.netid.nau.caprecid for a capacity record connection.  
- Ensure that the referenced Facility Class Profiles are RACLST-specified.  
- For CPC connections only: The SNMP community name specified in the security product (SAF) for a particular target CPC does not match the SNMP community name defined in the support element of the target CPC. See [“Community name defined in the security product for each CPC” on page 14-5](#) for further information regarding community name setup. |
| F03 HWI_INTERRUPT_STATUS_INV            | **Meaning**: The calling program is disabled. The system rejects this service request.  
**Action**: Check the calling program for a probable coding error. |
| F04 HWI_MODE_INV                        | **Meaning**: The calling program is not in task mode. The system rejects this service request.  
**Action**: Check the calling program for a probable error. |
| F05 HWI_LOCKS_HELD                      | **Meaning**: The calling program is holding one or more locks. The system rejects this service request.  
**Action**: Check the calling program for a probable coding error. |
Return Code in Hexadecimal Equate Symbol | Meaning and Action
--- | ---
F06 HWI_UNSUPPORTED_RELEASE | **Meaning**: The system level does not support this service. The system rejects this service request.  
**Action**: Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again.

FFF HWI_UNEXPECTED_ERROR | **Meaning**: System error. The service that was called encountered an unexpected error. The system rejects the service call.  
**Action**: Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.

**Example**
In the pseudocode example, the application attempts to establish a connection between the application and the target CPC.

```c
InConnectToken = 16blanks;  
ConnectType = HWI_CPC;  
ConnectTypeValue_Ptr = Addr(ConnectTypeValue);  
ConnectTypeValue = 'CPCPLEX1.CPC01';  
CALL HWICONN (ReturnCode, InConnectToken, OutConnectToken,  
              ConnectType, ConnectTypeValue_Ptr, DiagArea)  
| (After the call, OutConnectToken contains a token that can be used on all  
   subsequent calls to perform CPC functions against the 'CPCPLEX1.CPC01' CPC  
   including connecting to images, capacity records, and activation profiles  
   residing on the CPC.)  
```

**HWIDISC — Release a BCPii Connection**
Call the HWIDISC service to release the logical connection between the application and the identified CPC, image, capacity record, or activation profile. If the connect token represents a CPC, any subordinate image, capacity record, or activation profile connection associated with the same CPC connection is also released.

**Description**

**Environment**
The requirements for the callers are:

- **Minimum authorization**: One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
- **Dispatchable unit mode**: Task
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 31-bit
- **ASC mode**: Primary or access register (AR)
- **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
See “Syntax, Linkage and Programming Considerations” on page 14-9 for details about how to call BCPii services in the various programming languages.

Restrictions
None.

Authorization
The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

The client application must also have at least read access to the following class resources:

- The SAF-protected FACILITY class resource HWI.TARGET.netid.nau for HWI_CPC, HWI_RESET_ACT_PROF, HWI_IMAGE_ACT_PROF, or HWI_LOAD_ACT_PROF connections
- The SAF-protected FACILITY class resource HWI.TARGET.netid.nau.imagename for HWI_IMAGE connections
- The SAF-protected FACILITY class resource HWI.CAPREC.netid.nau.caprecid for HWI_CAPREC connections

Note: BCPii require the FACILITY class to be RACLIST-specified.

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

```
CALL HWIDISC (ReturnCode,ConnectToken,DiagArea)
```

Parameters
The parameters are explained as follows:

**ReturnCode**
- Returned parameter
  - Type: integer
  - Length: 4 bytes

  ReturnCode contains the return code from the service.

**ConnectToken**
- Supplied parameter
  - Type: character string
  - Length: 16 bytes

  ConnectToken specifies the logical connection to be released. A ConnectToken represents a logical connection between the application and a CPC, image, capacity record, or activation profile, and is returned as an output parameter on the HWICONN service call.

  The ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call.
**DiagArea**

Returned parameter
- Type: character
- Length: 32 bytes

DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

**Note:** For all environmental errors (with return code X'F00' and higher), the DiagArea might not be filled in, and the data returned in the area should be ignored.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag_Index</td>
<td>32-bit integer</td>
<td>The array index to the parameter field that causes the error.</td>
</tr>
<tr>
<td>Diag_Key</td>
<td>32-bit integer</td>
<td>The constant value represents the field that causes the error.</td>
</tr>
<tr>
<td>Diag_Actual</td>
<td>32-bit integer</td>
<td>The incorrect actual value specified.</td>
</tr>
<tr>
<td>Diag_Expected</td>
<td>32-bit integer</td>
<td>The expected value to be used.</td>
</tr>
<tr>
<td>Diag_CommErr</td>
<td>32-bit integer</td>
<td>The returned code that is returned from the console application API or the BCPii transport layer.</td>
</tr>
<tr>
<td>Diag_text</td>
<td>character (12)</td>
<td>Additional diagnostic information in text format.</td>
</tr>
</tbody>
</table>

**ABEND Codes**

The call might result in an abend X'042' with a reason code of X'0003yyyy' due to one of the following reasons:

<table>
<thead>
<tr>
<th>yyyy</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The parameters passed by the caller are not in the primary address space.</td>
</tr>
<tr>
<td>0001</td>
<td>The parameters passed by the caller are not accessible.</td>
</tr>
<tr>
<td>0002</td>
<td>The number of parameters passed by the caller is not correct.</td>
</tr>
</tbody>
</table>

**Return Codes**

When the service returns control to the caller, GPR 15 and ReturnCode contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return Code in Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 HWI_OK</td>
<td>Meaning: Successful completion. Action: None.</td>
</tr>
<tr>
<td>Return Code in Hexadecimal Equate Symbol</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| 100 HWI_CONNECT_TOKEN_INV | **Meaning:** Program error. The specified connect token is not valid. This return code indicates that one of the following conditions has occurred:  
  • The input connect token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken.  
  • The connect token does not represent an active connection. The connection specified might have already been disconnected using the HWIDISC service call.  
  • The connect token is not associated with the address space of the caller. The ConnectToken specified is associated with a different address space than the caller of this service call.  
  **Action:** Check for probable coding error. |
| 101 HWI_COMMUNICATION_ERROR | **Meaning:** A communication error is detected. The hardware management console application API (HWMCA) or the BCPii transport layer has returned with a failing return code.  
  **Action:** See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPii transport layer.  
  HWMCA API and BCPii transport return codes are provided in Appendix A, "BCPii Communication Error Reason Codes," on page A-1. |
| 102 HWI_DIAGAREA_INV | **Meaning:** Program error. The DiagArea is not accessible.  
  **Action:** Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field. |
| 901 HWI_DISC_INPROGRESS | **Meaning:** Another Disconnect request is already in progress. This request is redundant.  
  **Action:** None. |
| F00 HWI_NOT_AVAILABLE | **Meaning:** BCPii services are not available, and the system rejects the service request.  
  **Action:** Notify the system programmer to start the BCPii address space and try the request again. See "Restarting the HWIBCPII Address Space" on page 14-8 about how to start the BCPii address space.  
  Programs can also listen to ENF68 to determine when BCPii services are available. See z/OS MVS Programming: Authorized Assembler Services Reference EDT-FXS for how to listen for BCPii activation messages. |
| F01 HWI_AUTH_FAILURE | **Meaning:** The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.  
  **Action:** Check the calling program for a probable coding error. |
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>F02 HWI_NO_SAF_AUTH</td>
<td>The user does not have correct SAF authorization for the request.</td>
<td><strong>Meaning:</strong> The user does not have correct SAF authorization for the request. <strong>Action:</strong> Check for probable error. Consider one or more of the following possible actions:</td>
</tr>
<tr>
<td></td>
<td>- Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for a CPC or activation profile connection.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for an image connection.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Define read access authorization to the FACILITY class resource profile HWI.CAPREC.netid.nau.caprecid for a CAPREC connection.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Ensure that the referenced Facility Class Profiles are RACLIST-specified.</td>
<td></td>
</tr>
<tr>
<td>F03 HWI_INTERRUPT_STATUS_INV</td>
<td>The calling program is disabled. The system rejects this service request.</td>
<td><strong>Meaning:</strong> The calling program is disabled. The system rejects this service request. <strong>Action:</strong> Check the calling program for a probable coding error.</td>
</tr>
<tr>
<td>F04 HWI_MODE_INV</td>
<td>The calling program is not in task mode. The system rejects this service request.</td>
<td><strong>Meaning:</strong> The calling program is not in task mode. The system rejects this service request. <strong>Action:</strong> Check the calling program for a probable error.</td>
</tr>
<tr>
<td>F05 HWI_LOCKS_HELD</td>
<td>The calling program is holding one or more locks. The system rejects this service request.</td>
<td><strong>Meaning:</strong> The calling program is holding one or more locks. The system rejects this service request. <strong>Action:</strong> Check the calling program for a probable coding error.</td>
</tr>
<tr>
<td>F06 HWI_UNSUPPORTED_RELEASE</td>
<td>The system level does not support this service. The system rejects this service request.</td>
<td><strong>Meaning:</strong> The system level does not support this service. The system rejects this service request. <strong>Action:</strong> Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again.</td>
</tr>
<tr>
<td>FFF HWI_UNEXPECTED_ERROR</td>
<td>System error. The service that was called encountered an unexpected error. The system rejects the service call.</td>
<td><strong>Meaning:</strong> System error. The service that was called encountered an unexpected error. The system rejects the service call. <strong>Action:</strong> Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</td>
</tr>
</tbody>
</table>

**Example**

In the pseudocode example, the caller issues a call to release a connection between the application and a CPC.

```plaintext
CALL HWIDISC (ReturnCode, ConnectToken, DiagArea)
```

**HWIEVENT — Register for BCPii Events**

Call the HWIEVENT service for the following purposes:
Register an application and its connection to be notified for one or more hardware or software events occurring on the connected CPC or image.

Delete the registration for notification of one or more previously registered events.

**Description**

**Environment**
The requirements for the callers are:

- **Minimum authorization:** One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
- **Dispatchable unit mode:** Task
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 31-bit
- **ASC mode:** Primary or access register (AR)
- **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**
See “Syntax, Linkage and Programming Considerations” on page 14-9 for details about how to call BCPii services in the various programming languages. For programming language C, see restrictions below.

**Restrictions**
This service is not used by C language callers running in a UNIX® System Services (USS) environment. See “HWIManageEvents — Manage the List of BCPii Events” on page 14-94.

**Authorization**
The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

The client application must have at least read access to the SAF-protected FACILITY class resource HWI.TARGET.netid.nau for a ConnectToken representing a CPC connection, or HWI.TARGET.netid.nau.imagename for ConnectToken representing an image connection.

**Note:** BCPii require the FACILITY class to be RACLIST-specified.

**Syntax**
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.
CALL HWIEVENT
(  
  ReturnCode  
  ,ConnectToken  
  ,EventAction  
  ,EventIDs  
  ,EventExitMode  
  ,EventExitAddr  
  ,EventExitParm  
  ,DiagArea)

Parameters
The parameters are explained as follows:

ReturnCode
  Returned parameter
  • Type: integer
  • Length: 4 bytes
  
  ReturnCode contains the return code from the service.

ConnectToken
  Supplied parameter
  • Type: character string
  • Length: 16 bytes
  
  ConnectToken represents a logical connection between the application and a
  CPC or image. The ConnectToken is an output parameter on the HWICONN
  service call.
  
  The ConnectToken specified must have originated from a HWICONN service
  call that was issued from the same address space as this service call.

EventAction
  Supplied parameter
  • Type: integer
  • Length: 4 bytes
  
  EventAction specifies the type of action for the service.

<table>
<thead>
<tr>
<th>Constant in Hexadecimal (Decimal)</th>
<th>Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1)</td>
<td>HWI_EVENT_ADD</td>
<td>Registers to be notified when the requested events occur.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2)</td>
<td>HWI_EVENT_DELETE</td>
<td>Deletes the registration for notification.</td>
</tr>
</tbody>
</table>

EventIDs
  Supplied parameter
- Type: integer
- Length: 128 bits (16 bytes)

EventIDs specifies the events to be added or deleted. Each event is a 1-bit field from bit position of 97 to 128 in this data area. If the bit is on, the service performs the EventAction operation for this event on the requested connection.

The following event IDs can be specified:

<table>
<thead>
<tr>
<th>Bit position in structure specified on EventIDs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-96 Hwi_EventID_EyeCatcher</td>
<td>Control block identifier. Note: HWI_EVENTID_TEXT can be used to initialize this field.</td>
</tr>
<tr>
<td>97 Hwi_Event_CmdResp</td>
<td>Requests to add or delete the registration for notification of the command response events. Note: The input connect token represents a CPC connection or an image connection.</td>
</tr>
<tr>
<td>98 Hwi_Event_StatusChg</td>
<td>Requests to add or delete the registration for notification of the status change events. Note: The input connect token represents a CPC connection or an image connection.</td>
</tr>
<tr>
<td>99 Hwi_Event_NameChg</td>
<td>Requests to add or delete the registration for notification of the object name change events. Note: The input connect token represents a CPC connection or an image connection.</td>
</tr>
<tr>
<td>100 Hwi_Event_ActProfChg</td>
<td>Requests to add or delete the registration for notification of the change events for the activation profile name. Note: The input connect token represents a CPC connection or an image connection.</td>
</tr>
<tr>
<td>101 Hwi_Event_ObjCreate</td>
<td>Requests to add or delete the registration for notification of the object created events. Note: The input connect token represents a CPC connection or an image connection.</td>
</tr>
<tr>
<td>102 Hwi_Event_ObjDestroy</td>
<td>Requests to add or delete the registration for notification of the object destroyed (deleted) events. Note: The input connect token represents a CPC connection or an image connection.</td>
</tr>
<tr>
<td>103 Hwi_Event_ObjException</td>
<td>Requests to add or delete the registration for notification of the exception state events. Note: The input connect token represents a CPC connection or an image connection.</td>
</tr>
<tr>
<td>104 Hwi_Event_ApplStarted</td>
<td>Requests to add or delete the registration for notification of the console application started events. Note: The input connect token must only represent a CPC connection.</td>
</tr>
<tr>
<td>105 Hwi_Event_ApplEnded</td>
<td>Requests to add or delete the registration for notification of the console application ended events. Note: The input connect token must only represent a CPC connection.</td>
</tr>
<tr>
<td>106 Hwi_Event_HwMsg</td>
<td>Requests to add or delete the registration for notification of the hardware message events. Note: The input connect token must only represent a CPC connection.</td>
</tr>
<tr>
<td>107 Hwi_Event_HwMsgDel</td>
<td>Requests to add or delete the registration for notification of the hardware message deletion events. Note: The input connect token must only represent a CPC connection.</td>
</tr>
<tr>
<td>108 Hwi_Event_SecurityEvent</td>
<td>Requests to add or delete the registration for notification of the console security events. Note: The input connect token must only represent a CPC connection.</td>
</tr>
<tr>
<td>Bit position in structure specified on EventIDs</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 109 Hwi_Event_CapacityChg | Requests to add or delete the registration for notification of the capacity change events.  
**Note:** The input connect token must only represent a CPC connection. |
| 110 Hwi_Event_CapacityRecord | Requests to add or delete the registration for notification of the capacity record change events.  
**Note:** The input connect token must only represent a CPC connection. |
| 111 Hwi_Event_OpSysMsg | Requests to add or delete the registration for notification of the operating system message events.  
**Note:** The input connect token must only represent an image connection. |
| 112 Hwi_Event_HwCommError | Requests to add or delete the registration for notification of the hardware communication error events.  
**Note:** The input connect token must only represent a CPC connection. |
| 113 Hwi_Event_BCPiiStatus | Requests to add or delete the registration for notification of BCPii status change events. |
| 114 Hwi_Event_DisabledWait | Requests to add or delete the registration for notification of disabled wait events.  
**Note:** The input connect token must only represent an image connection. |
| 115-128 Hwi_Event_Reserved | Reserved, must be initialized to binary zeros. |

**EventExitMode**

Supplied parameter
- **Type:** integer
- **Length:** 4 bytes

EventExitMode specifies the type of the exit mode for the service.

<table>
<thead>
<tr>
<th>Constant in Hexadecimal (Decimal) Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1) HWI_EVENT_TASK</td>
<td>The base control program internal interface gives control in task mode to an ENF listen-exit routine as specified on the EventExitAddr parameter. Task mode ENF exits must reside in common storage.</td>
</tr>
</tbody>
</table>

At present, only one value is allowed for this parameter. In the future, IBM might choose to allow additional values to be specified.

**EventExitAddr**

Supplied parameter
- **Type:** pointer
- **Length:** 4 bytes

EventExitAddr specifies the address of an ENF listen-exit routine that receives control when the requested event occurs. The application is responsible for writing this ENF exit routine, as described in the ENFREQ documentation for ENF 68 found in [z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG](https://www.ibm.com/support/docview.wss?uid=swg27013704). For further information regarding the coding of ENF exits, see the "Listening for System Events" chapter in the [z/OS MVS Programming: Authorized Assembler Services Guide](https://www.ibm.com/support/docview.wss?uid=swg27013704).

**EventExitParm**

Supplied parameter
- **Type:** pointer
- **Length:** 4 bytes
EventExitParm specifies an optional value to be passed to the ENF listen-exit when invoked, as described in the ENFREQ documentation for ENF 68 found in z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG.

DiagArea

Returned parameter

- Type: character string
- Length: 32 bytes

DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X’F00’ and higher), the DiagArea might not be filled in, and the data returned in the area should be ignored.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag_Index</td>
<td>32-bit integer</td>
<td>The array index to the parameter field that causes the error.</td>
</tr>
<tr>
<td>Diag_Key</td>
<td>32-bit integer</td>
<td>The constant value represents the field that causes the error.</td>
</tr>
<tr>
<td>Diag_Actual</td>
<td>32-bit integer</td>
<td>The incorrect actual value specified.</td>
</tr>
<tr>
<td>Diag_Expected</td>
<td>32-bit integer</td>
<td>The expected value to be used.</td>
</tr>
<tr>
<td>Diag_CommErr</td>
<td>32-bit integer</td>
<td>The return code that is returned from the console application API or the BCPii transport layer.</td>
</tr>
<tr>
<td>Diag_Text</td>
<td>character (12)</td>
<td>Additional diagnostic information in text format.</td>
</tr>
</tbody>
</table>

ABEND Codes

The call might result in an abend X'042' with a reason code of X'0004yyyy' due to one of the following reasons:

<table>
<thead>
<tr>
<th>yyyy</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The parameters passed by the caller are not in the primary address space.</td>
</tr>
<tr>
<td>0001</td>
<td>The parameters passed by the caller are not accessible.</td>
</tr>
<tr>
<td>0002</td>
<td>The number of parameters passed by the caller is not correct.</td>
</tr>
</tbody>
</table>

Return Codes

When the service returns control to the caller, GPR 15 and ReturnCode contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return Code in Hexadecimal</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 HWI_OK</td>
<td></td>
<td>Meaning: Successful completion.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>Return Code in Hexadecimal</td>
<td>Equate Symbol</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| 100 HWI_CONNECT_TOKEN_INV  |               | **Meaning:** Program error. The specified connect token is not valid. This return code indicates that one of the following conditions has occurred:  
  - The connect token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken.  
  - The connect token does not represent an active connection. The connection specified might have already been disconnected by the HWIDISC service call.  
  - The connect token is not associated with the address space of the caller. The ConnectToken specified is associated with a different address space than the caller of this service call.  
  **Action:** Check for probable coding error. |
| 101 HWI_COMMUNICATION_ERROR |               | **Meaning:** A communication error is detected. The hardware management console application API (HWMCA) or the BCPii transport layer has returned with a failing return code.  
  **Action:** See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPii transport layer. BCPii’s CTRACE might provide further diagnostic information if the problem cannot easily be resolved. See [z/OS MVS System Commands](https://publib.boulder.ibm.com/infocenter/zos/v1r11/index.jsp?topic=%252Fcom.ibm.zos.doc%252Finfo%252Ficszsm1%252Ficszsm1.html) for further information about starting and stopping CTRACE.  
| 102 HWI_DIAGAREA_INV       |               | **Meaning:** Program error. The DiagArea is not accessible.  
  **Action:** Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field. |
| 103 HWI_CONNECT_TOKEN_INACTIVE |           | **Meaning:** The specified connect token is no longer valid. The connection has been disconnected, or it is in the progress of being disconnected.  
  **Action:** Check for probable coding error. Verify that the specified connect token is still active. If connectivity to the targeted CPC connection no longer exists, all connections associated with that CPC will no longer have a connect token that can be used. |
| 701 HWI_EVENT_EXITMODE_INV |               | **Meaning:** Program error. The requested EventExitMode on the call is not valid. The system rejects the service call.  
  **Action:** Check for probable coding error. |
| 702 HWI_EVENT_EXITADDR_INV |               | **Meaning:** Program error. The requested EventExitAddr on the call is not valid. The system rejects the service call.  
  **Action:** Check for probable coding error. |
| 703 HWI_EVENT_ACTION_INV   |               | **Meaning:** Program error. The requested EventAction on the call is not valid. The system rejects the service call.  
  **Action:** Check for probable coding error. |
<table>
<thead>
<tr>
<th>Return Code in Hexadecimal</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 704 HWI_EVENT_IDS_INV     | **Meaning:** Program error. The requested EventIDs on the call is not valid. The system rejects the service call. This return code indicates one of the following conditions has occurred:  
  - The first 12 bytes of the EventIDs parameter is not equal to the expected Eyecatcher of HWIEVENTBLCK.  
  - The reserved area of the EventIDs parameter contains a non-zero value.  
  - The EventIDs specified applies only to a CPC connection, but the ConnectToken specified represents an image or capacity record connection.  
  - The EventIDs specified applies only to image connections, but the ConnectToken specified represents a CPC or capacity record connection.  
  - A request which specified an EventAction of HWI_EVENT_DELETE also specified EventIDs of one or more events that were not registered on a previous HWIEVENT EventAction = HWI_EVENT_ADD request for the connection.  
**Action:** Check for probable coding error. |
| F00 HWI_NOT_AVAILABLE     | **Meaning:** BCPii is not available, and the system rejects the service request.  
**Action:** Notify the system programmer to start the BCPii address space and try the request again. See "Restarting the HWIBCPII Address Space" on page 14-8 about how to start the BCPii address space.  
Programs can also listen to ENF68 to determine when BCPii services are available. See z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG for how to listen for BCPii activation messages. |
| F01 HWI_AUTH_FAILURE      | **Meaning:** The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.  
**Action:** Check the calling program for a probable coding error. |
| F02 HWI_NO_SAF_AUTH       | **Meaning:** The user does not have correct SAF authorization for the request.  
**Action:** Check for probable error. Consider one or more of the following possible actions:  
  - Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV.  
  - Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for CPC connection.  
  - Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for an image connection.  
  - Ensure that the referenced FACILITY class profiles are RACLIST-specified. |
| F03 HWI_INTERRUPT_STATUS_INV | **Meaning:** The calling program is disabled. The system rejects this service request.  
**Action:** Check the calling program for a probable coding error. |
## HWIEVENT

<table>
<thead>
<tr>
<th>Return Code in Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| F04 HWI_MODE_INV                       | **Meaning:** The calling program is not in task mode. The system rejects this service request.  
**Action:** Check the calling program for a probable error. |
| F05 HWI_LOCKS_HELD                     | **Meaning:** The calling program is holding one or more locks. The system rejects this service request.  
**Action:** Check the calling program for a probable coding error. |
| F06 HWI_UNSUPPORTED_RELEASE            | **Meaning:** The system level does not support this service. The system rejects this service request.  
**Action:** Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again. |
| FFF HWI_UNEXPECTED_ERROR               | **Meaning:** System error. The service that was called encountered an unexpected error. The system rejects the service call.  
**Action:** Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. |

### Example

In the pseudocode example, the caller issues a call to register to be notified when the command response events and status change events occur.

```
Declare (ReturnCode, EventAction, EventExitMode) Fixed(31);
Declare ConnectToken Isa(HWI_CONNTOKEN_TYPE):
Declare EventIDs Isa(HWI_EVENTIDS_TYPE):
Declare (EventExitAddr, EventExitParm) Ptr(31):
Declare DiaArea Isa(HWI_DIAGAREA_TYPE):
Declare EventExit Entry External;

EventAction = HWI_EVENT_ADD;
Hwi_EventID_EyeCatcher = HWI_EVENTID_TEXT;
Hwi_Event_CmdResp = on;
Hwi_Event_StatusChg = on;
Hwi_Event_Reserv = 0;
EventExitMode = HWI_EVENT_TASK;
EventExitAddr = ADDR(EventExit);
EventExitParm = 0;

CALL HWIEVENT (ReturnCode, ConnectToken, EventAction, EventIDs,  
EventExitMode, EventExitAddr, EventExitParm, DiagArea);
```

### HWILIST — Retrieve HMC and BCPii Configuration-Related Information

Call the HWILIST service to retrieve hardware management console (HMC) and BCPii configuration-related information. Depending on which information is requested, the data returned by this service can be used on subsequent BCPii service calls to take the following actions:

- Connect to a central processor complex (CPC), image (LPAR), capacity record (CAPREC), reset activation profile, image activation profile, or load activation profile using the HWICONN API.
- Register for the proper events (HWIEVENT) using the HWIEVENT API.
- Connect to the local CPC or Image.
**Description**

**Environment**
The requirements for the callers are:

- **Minimum authorization:** One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
- **Dispatchable unit mode:** Task
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 31-bit
- **ASC mode:** Primary or access register (AR)
- **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**
See “Syntax, Linkage and Programming Considerations” on page 14-9 for details about how to call BCPii services in the various programming languages.

**Restrictions**
None.

**Authorization**
The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

For a ListType of HWI_LIST_CPCS, when BCPii is creating the list of CPC network addresses, only those CPC network addresses that the application has at least read access to are listed. The HWI.TARGET.netid.nau FACILITY class resource is consulted to determine this.

For a ListType of HWI_LIST_IMAGES, when BCPii is creating the list of image (LPAR) names, only those image names that the application has at least read access to are listed. The HWI.TARGET.netid.nau.imagename FACILITY class resource is consulted to determine this.

For a ListType of HWI_LIST_CAPRECS, when BCPii is creating the list of capacity records, only those capacity records that the application has at least read access to are listed. The HWI.CAPREC.netid.nau.caprecid FACILITY class resource is consulted to determine this.

For a ListType of HWI_LIST_EVENTS, an application must have at least read access to the SAF-protected FACILITY class resource HWI.TARGET.netid.nau for a CPC connection; or at least read access to the SAF-protected FACILITY class resource HWI.TARGET.netid.nau.imagename for an image connection.

For a ListType of HWI_LIST_LOCALCPC, an application must have at least read access to the HWI.TARGET.netid.nau FACILITY class resource profile where netid.nau represents the local CPC network address.
For a ListType of HWI_LIST_LOCALIMAGE, an application must have at least read access to the HWI.TARGET.netid.nau.imagename FACILITY class resource profile where netid.nau represents the local CPC network address and imagename represents the local image (LPAR) name.

For a ListType of HWI_LIST_RESET_ACT_PROFS, HWI_LIST_IMAGE_ACT_PROFS, or HWI_LIST_LOAD_ACT_PROFS, when BCPii is creating the list of activation profiles names, an application needs to have at least read access to the HWI.TARGET.netid.nau FACILITY class resource for the CPC to which the activation profiles apply.

**Note:** BCPii require the FACILITY class to be RACLIST-specified.

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

```call
CALL HWILIST
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ReturnCode</strong></td>
<td>Returned parameter</td>
</tr>
<tr>
<td></td>
<td>Type: integer</td>
</tr>
<tr>
<td></td>
<td>Length: 4 bytes</td>
</tr>
<tr>
<td></td>
<td>ReturnCode contains the return code from the service.</td>
</tr>
<tr>
<td><strong>ConnectToken</strong></td>
<td>Supplied parameter</td>
</tr>
<tr>
<td></td>
<td>Type: character string</td>
</tr>
<tr>
<td></td>
<td>Length: 16 bytes</td>
</tr>
<tr>
<td></td>
<td>ConnectToken represents a logical connection between the application and a</td>
</tr>
<tr>
<td></td>
<td>CPC or image. The ConnectToken is an output parameter on the HWICONN</td>
</tr>
<tr>
<td></td>
<td>service call.</td>
</tr>
<tr>
<td></td>
<td>The ConnectToken specified must have originated from a HWICONN service</td>
</tr>
<tr>
<td></td>
<td>call that was issued from the same address space as this service call.</td>
</tr>
<tr>
<td></td>
<td>If the ListType is HWI_LIST_CPCS, HWI_LIST_LOCALCPC, or</td>
</tr>
<tr>
<td></td>
<td>HWI_LIST_LOCALIMAGE, this parameter is not relevant and is ignored.</td>
</tr>
<tr>
<td></td>
<td>If the ListType is HWI_LIST_IMAGES, HWI_LIST_CAPRECS,</td>
</tr>
<tr>
<td></td>
<td>HWI_LIST_RESET_ACT_PROFS, HWI_LIST_IMAGE_ACT_PROFS, or</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ListType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWI_LIST_LOCALIMAGE</td>
<td>Set to local image for which activation profiles are to be listed.</td>
</tr>
<tr>
<td>HWI_LIST_CPCS</td>
<td>Set to local CPCs for which activation profiles are to be listed.</td>
</tr>
<tr>
<td>HWI_LIST_LOCALCPC</td>
<td>Set to local CPCs for which activation profiles are to be listed.</td>
</tr>
<tr>
<td>HWI_LIST_IMAGES</td>
<td>Set to local images for which activation profiles are to be listed.</td>
</tr>
<tr>
<td>HWI_LIST_CAPRECS</td>
<td>Set to local CAP records for which activation profiles are to be listed.</td>
</tr>
<tr>
<td>HWI_LIST_RESET_ACT_PROFS</td>
<td>Set to local activation profiles for which reset action is specified.</td>
</tr>
</tbody>
</table>
HWI_LIST_LOAD_ACT_PROFS, this request must be directed to a specific CPC. Therefore, a connect token that represents an already active BCPIi CPC connection must be specified.

For a ListType of HWI_LIST_EVENTS, the connect token must represent an already active BCPIi CPC or image connection, depending on which events are to be listed. If a list of CPC events is required, the connect token must represent an active CPC connection; likewise, if a list of image events is required, the connect token must represent an active image connection.

**ListType**
- **Supplied parameter**
  - **Type:** integer
  - **Length:** 4 bytes

ListType specifies the type of request for the service.

<table>
<thead>
<tr>
<th>Constant in Hexadecimal (Decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1) HWI_LIST_CPCS</td>
<td>Requests for a list of CPCs that can be accessed.</td>
</tr>
<tr>
<td>2 (2) HWI_LIST_IMAGES</td>
<td>Requests for a list of image names that can be accessed.</td>
</tr>
<tr>
<td>3 (3) HWI_LIST_EVENTS</td>
<td>Requests for a list of previously subscribed events.</td>
</tr>
<tr>
<td>4 (4) HWI_LIST_CAPRECS</td>
<td>Requests for a list of capacity record ID names that can be accessed.</td>
</tr>
<tr>
<td>5 (5) HWI_LIST_LOCALCPC</td>
<td>Requests the name of the local CPC on which the caller is currently executing.</td>
</tr>
<tr>
<td>6 (6) HWI_LIST_LOCALIMAGE</td>
<td>Requests the name of the local Image (LPAR) on which the HWILIST caller is currently executing.</td>
</tr>
<tr>
<td>7 (7) HWI_LIST_RESET_ACT_PROFS</td>
<td>Requests for a list of the currently defined reset activation profiles.</td>
</tr>
<tr>
<td>8 (8) HWI_LIST_IMAGE_ACT_PROFS</td>
<td>Requests for a list of the currently defined image activation profiles.</td>
</tr>
<tr>
<td>9 (9) HWI_LIST_LOAD_ACT_PROFS</td>
<td>Requests for a list of the currently defined load activation profiles.</td>
</tr>
</tbody>
</table>
NumofDataItemsReturned
Returned parameter
- Type: integer
- Length: 4 bytes

NumofDataItemsReturned contains the number of data items returned in the answer area.

AnswerArea_Ptr
Supplied parameter
- Type: pointer
- Length: 4 bytes

AnswerArea_Ptr specifies the address of the answer area where the requested data is returned.

The ListType specified determines the format of the returned data.

<table>
<thead>
<tr>
<th>ListType</th>
<th>Data to be returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWI_LIST_CPCS</td>
<td>A string comprised of a list of blank-separated concatenated 17-character CPC network addresses. Each network address is in the form of a 1- through 8-character netid, followed by a period, and followed by a 1- through 8-character network addressable unit (NAU) name. The network address is padded with trailing blanks if the total string length of the network address is less than 17 characters. Example: net1.cpc01.</td>
</tr>
<tr>
<td>HWI_LIST_IMAGES</td>
<td>A string comprised of a list of blank-separated concatenated 8-character image names padded with trailing blanks.</td>
</tr>
<tr>
<td>HWI_LIST_EVENTS</td>
<td>A 128-bit string. The first 96 bits (12 bytes) is an eye-catcher value of HWIEVENTBLCK. The last 32 bits represents events already registered for notification. These events were registered by previous HWIEVENT ADD service calls. The returned event indicators are specific to the ConnectToken specified. These indicators are mapped by the type structure HWI_EVENTIDS_TYPE from the BCPii services interface declaration file. If a particular indicator is on, that event is active for this connection.</td>
</tr>
<tr>
<td>HWI_LIST_CAPRECS</td>
<td>A string comprised of a list of blank-separated concatenated 8-character CAPREC names padded with trailing blanks.</td>
</tr>
<tr>
<td>HWI_LIST_LOCALCPC</td>
<td>A 17-character string representing the CPC network address of the local CPC. The network address is in the form of a 1- to 8-character netid, followed by a period, followed by a 1- to 8-character network addressable unit (NAU) name. The network address is padded with trailing blanks.</td>
</tr>
<tr>
<td>HWI_LIST_LOCALIMAGE</td>
<td>An 8-character string representing the image name of the local image (LPAR) padded with trailing blanks.</td>
</tr>
<tr>
<td>HWI_LIST_RESET_ACT_PROFS</td>
<td>A string comprised of a list of concatenated 16-character reset activation profile names padded with trailing blanks.</td>
</tr>
<tr>
<td>HWI_LIST_IMAGE_ACT_PROFS</td>
<td>A string comprised of a list of concatenated 16-character image activation profile names padded with trailing blanks.</td>
</tr>
<tr>
<td>HWI_LIST_LOAD_ACT_PROFS</td>
<td>A string comprised of a list of concatenated 16-character load activation profile names padded with trailing blanks.</td>
</tr>
</tbody>
</table>

AnswerAreaLen
Supplied parameter
- Type: integer
- Length: 4 bytes
AnswerAreaLen specifies the length in bytes of the AnswerArea pointed to by the AnswerArea_Ptr. The amount of storage required by the application at the AnswerArea_Ptr location depends primarily on two factors:

1. The ListType specified
2. The number of data items expected to be returned

For example, if a ListType of HWI_LIST_CPCS is specified and the current HMC LAN has 7 CPCs connected to it, at least 17 bytes x 7 CPCs + the number of blank spaces among the CPCs = 119 + 6 = 125 bytes of data are required for the AnswerArea.

**DiagArea**

Returned parameter
- Type: character
- Length: 32 bytes

DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

**Note:** For all environmental errors (with return code X'F00' and higher), the DiagArea might not be filled in, and the data returned in the area should be ignored.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag_Index</td>
<td>32-bit integer</td>
<td>The array index to the parameter field that causes the error.</td>
</tr>
<tr>
<td>Diag_Key</td>
<td>32-bit integer</td>
<td>The constant value represents the field that causes the error.</td>
</tr>
<tr>
<td>Diag_Actual</td>
<td>32-bit integer</td>
<td>The incorrect actual value specified.</td>
</tr>
<tr>
<td>Diag_Expected</td>
<td>32-bit integer</td>
<td>The expected value to be used.</td>
</tr>
<tr>
<td>Diag_CommErr</td>
<td>32-bit integer</td>
<td>The returned code that is returned from the console application API or the BCPii transport layer.</td>
</tr>
<tr>
<td>Diag_text</td>
<td>character (12)</td>
<td>Additional diagnostic information in text format.</td>
</tr>
</tbody>
</table>

**ABEND Codes**
The call might result in an abend X'042' with a reason code of X'0005yyyy' due to one of the following reasons:

<table>
<thead>
<tr>
<th>yyyy</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The parameters passed by the caller are not in the primary address space.</td>
</tr>
<tr>
<td>0001</td>
<td>The parameters passed by the caller are not accessible.</td>
</tr>
<tr>
<td>0002</td>
<td>The number of parameters passed by the caller is not correct.</td>
</tr>
</tbody>
</table>

**Return Codes**
When the service returns control to the caller, GPR 15 and ReturnCode contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return Code in Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 HWI_OK</td>
<td>Meaning: Successful completion. Action: None.</td>
</tr>
<tr>
<td>Return Code in Hexadecimal Equate Symbol</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------</td>
</tr>
</tbody>
</table>
| 100 HWI_CONNECT_TOKEN_INV              | **Meaning:** Program error. The specified connect token is not valid. This return code indicates one of the following conditions has occurred:  
- The connect token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken.  
- The connect token does not represent an active connection. The connection specified might have already been disconnected using the HWIDISC service call.  
- The connect token is not associated with the address space of the caller. The ConnectToken specified is associated with a different address space than the caller of this service call.  
**Action:** Check for probable coding error. |
| 101 HWI_COMMUNICATION_ERROR            | **Meaning:** A communication error is detected. The hardware management console application API (HWMCA) or the BCPIi transport layer has returned with a failing return code.  
**Action:** See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPIi transport layer.  
HWMCA API and BCPIi transport return codes are provided in Appendix A, “BCPIi Communication Error Reason Codes,” on page A-1. |
| 102 HWI_DIAGAREA_INV                   | **Meaning:** Program error. The DiagArea is not accessible.  
**Action:** Check for probable coding error. Verify the specified DiagArea is defined as a 32-byte character field. |
| 103 HWI_CONNECT_TOKEN_INACTIVE         | **Meaning:** The specified connect token is no longer valid. The connection has been disconnected, or it is in the progress of being disconnected.  
**Action:** Check for probable coding error. Verify that the specified connect token is still active. If connectivity to the targeted CPC connection no longer exists, all connections associated with that CPC will no longer have a connect token that can be used. |
| 301 HWI_LISTTYPE_INV                   | **Meaning:** Program error. The requested LISTTYPE specified in the call is not valid. The system rejects the service call. This return code indicates one of the following conditions has occurred:  
- The ListType specified is not in the acceptable value range of possible list types.  
- The ListType specified applies only to CPC connections, but the ConnectToken specified represents an image connection or a capacity record connection.  
- For ListType HWI_LIST_EVENTS, the ConnectToken must not represent a capacity record because capacity record events do not have events directly associated with capacity records connections. Capacity-related events are associated with a CPC connection.  
**Action:** Check for probable coding error. Validate that the ListType specified is in the valid range of possible values, and that the ListType specified is permitted for the specified connection type. |
<table>
<thead>
<tr>
<th>Return Code in Hexadecimal Equate</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 302 HWI_DATA_EXCEEDED            | **Meaning:** Program error. The amount of returned data exceeded the size of the answer area. No data or only partial data is returned.  
**Action:** Check for probable coding error. See the DiagArea for further diagnostic information. The Diag_Actual indicates the application-specified length. The Diag_Expected indicates the size required for the AnswerArea. |
| 303 HWI_ANSWERAREA_INACCESSIBLE  | **Meaning:** Program error. The answer area data area is either partially or completely inaccessible by the application and the Base Control Program internal interface (BCPi) address space.  
**Action:** Check for probable coding error. Verify that the AnswerArea_Ptr points to a data area where the answer area is and make sure the data area is accessible. |
| 304 HWI_LIST_NODATA_RETURNED    | **Meaning:** There is no data to be returned or the caller does not have enough access to display the listed values.  
**Action:** Check for probable coding error. Verify that proper access is granted for the request. |
| 305 HWI_LISTTYPE_NOT_SUPPORTED  | **Meaning:** The targeted hardware of the HWILIST request does not support the request attempted by the program.  
**Action:** Verify that the targeted hardware supports the type of request being made. |
| F00 HWI_NOT_AVAILABLE            | **Meaning:** BCPii services are not available, and the system rejects the service request.  
**Action:** Notify the system programmer to start the BCPii address space and try the request again. See “Restarting the HWIBCPii Address Space” on page 14-8 about how to start the BCPii address space.  
Programs can also listen to ENF68 to determine when BCPii services are available. See z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG for how to listen for BCPii activation messages. |
| F01 HWI_AUTH_FAILURE             | **Meaning:** The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.  
**Action:** Check the calling program for a probable coding error. |
<table>
<thead>
<tr>
<th>Return Code in Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| F02 HWI_NO_SAF_AUTH                    | **Meaning:** The user does not have correct SAF authorization for the request.  
**Action:** Check for probable error. Consider one or more of the following possible actions:  
- Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV.  
- Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for HWI_LIST_IMAGES ListType.  
- Define read access authorization to the FACILITY class resource profile HWI.CAPREC.netid.nau.caprec for HWI_LIST_CAPRECS ListType.  
- For a ListType of HWI_LIST_EVENTS, define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for an image connection.  
- For a ListType of HWI_LIST_LOCALCPC, define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau where netid.nau represents the local CPC network address.  
- For a ListType of HWI_LIST_LOCALIMAGE, define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename where netid.nau represents the local CPC network address and imagename represents the local Image (LPAR) name.  
- For the ListType of HWI_LIST_RESET_ACT_PROFS, HWI_LIST_IMAGE_ACT_PROFS, or HWI_LIST_LOAD_ACT_PROFS, define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for the CPC that the application is attempting to retrieve the activation profile names.  
- Ensure that the referenced facility class profiles are RACLIST-specified. |
| F03 HWI_INTERRUPT_STATUS_INV            | **Meaning:** The calling program is disabled. The system rejects this service request.  
**Action:** Check the calling program for a probable coding error. |
| F04 HWI_MODE_INV                        | **Meaning:** The calling program is not in task mode. The system rejects this service request.  
**Action:** Check the calling program for a probable error. |
| F05 HWI_LOCKS_HELD                      | **Meaning:** The calling program is holding one or more locks. The system rejects this service request.  
**Action:** Check the calling program for a probable error. |
| F06 HWI_UNSUPPORTED_RELEASE             | **Meaning:** The system level does not support this service. The system rejects this service request.  
**Action:** Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again. |
Return Code in Hexadecimal Equate Symbol | Meaning and Action
---|---
FFF HWI_UNEXPECTED_ERROR | **Meaning:** System error. The service that was called encountered an unexpected error. The system rejects the service call.  
**Action:** Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.

**Example**
In the pseudocode example, the caller issues a call to retrieve a list CPCs that can be accessed.

```plaintext
ListType = HWI_LIST_CPCS;
AnswerArea_Ptr = addr(AnswerArea);
AnswerAreaLen = 125;
CALL HWILIST (ReturnCode, ConnectToken, ListType, NumofDataItemsReturned, 
AnswerArea_Ptr, AnswerAreaLen, DiagArea)
```

---

**HWIQUERY — BCPii Retrieval of SE/HMC-Managed Objects Data**

Call the HWIQUERY service to retrieve information about objects managed by the support element (SE) or hardware management console (HMC) related with central processor complexes (CPCs), CPC images (LPARs), capacity records, or different types of activation profiles.

**Description**

**Environment**
The requirements for the callers are:

- **Minimum authorization:** One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
- **Dispatchable unit mode:** Task
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 31-bit
- **ASC mode:** Primary or access register (AR)
- **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**
See “Syntax, Linkage and Programming Considerations” on page 14-9 for details about how to call BCPii services in the various programming languages.

**Restrictions**
None.
Authorization
The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

Client application must have at least read access to the SAF-protected FACILITY class HWI.TARGET.netid.nau for any CPC or activation profile queries, or HWI.TARGET.netid.nau.imagename for image queries, or HWI.CAPREC.netid.nau.caprecid for capacity record queries.

Note: BCPII require the FACILITY class to be RACLIST-specified.

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

```
CALL HWIQUERY (ReturnCode,ConnectToken,QueryParm_Ptr,NumOfAttributes,DiagArea)
```

Parameters
The parameters are explained as follows:

**ReturnCode**
- Returned parameter
  - Type: integer
  - Length: 4 bytes
  ReturnCode contains the return code from the service.

**ConnectToken**
- Supplied parameter
  - Type: character string
  - Length: 16 bytes
  ConnectToken represents a logical connection between the application and a CPC, image, capacity record, or activation profile. The ConnectToken is an output parameter on the HWICONN service call.

  The ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call.

**QueryParm_Ptr**
- Supplied parameter
  - Type: pointer
  - Length: 4 bytes
  QueryParm_Ptr specifies the address of a user-defined query structure that contains a list of one or more requested attributes to be queried, in the following form: attribute that is required, address of where returned value is to be stored, the length of the storage available to HWIQUERY to store the returned value, and the actual length of the data that will be returned in the data area.
The size of the data area pointed to by this parameter must be 16 bytes multiplied by the NumOfAttribute parameter. For example, if NumOfAttributes is 4, the data area pointed to by this parameter must be at least 64 bytes long (16 x 4).

The storage area that contains each attribute in the QueryParm is shown below:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeIdentifier</td>
<td>32-bit unsigned integer</td>
</tr>
<tr>
<td>AttributeValue_Ptr</td>
<td>Pointer</td>
</tr>
<tr>
<td>AttributeValueLen</td>
<td>32-bit unsigned integer</td>
</tr>
<tr>
<td>AttributeValueLenReturned</td>
<td>32-bit unsigned integer</td>
</tr>
</tbody>
</table>

This table is mapped by the data structure Hwi_QueryParm_Type in the data mappings provided for the various programming languages supported. See “Syntax, Linkage and Programming Considerations” on page 14-9 for more information.

If all of the data can be written into the data area (the AttributeValueLen is greater than or equal to the actual data returned), the AttributeValueLenReturned field contains the actual length of the data written in the storage specified at address AttributeValue_Ptr.

The AttributeValueLenReturned is only used as an output parameter. Any value contained in the field when HWIQUERY is called is ignored.

The following is the list of valid query attributes identifiers. For more information about these attributes, see the following publications:

- *System z9 and eServer zSeries Application Programming Interfaces* (SB10-7030-08)
- *System z10 and eServer zSeries Application Programming Interfaces* (SB10-7030-09)
- Publication appropriate to the level of hardware that the HWIQUERY is targeted

<table>
<thead>
<tr>
<th>Constant in Hexadecimal (Decimal)</th>
<th>Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1)</td>
<td>HWI_NAME  Requests to retrieve the name that represents the connect token.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: The input connect token must represent a CPC connection, an image connection, a reset activation profile connection, an image activation profile connection, or a load activation profile connection.</td>
</tr>
<tr>
<td>2</td>
<td>(2)</td>
<td>HWI_ERRSTAT Requests to retrieve whether the status is acceptable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: The input connect token must represent a CPC connection or an image connection.</td>
</tr>
<tr>
<td>3</td>
<td>(3)</td>
<td>HWI_BUSYSTAT Requests to retrieve whether the status is busy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: The input connect token must represent a CPC connection or an image connection.</td>
</tr>
<tr>
<td>4</td>
<td>(4)</td>
<td>HWI_MSGSTAT Requests to retrieve whether hardware messages are present.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: The input connect token must represent a CPC connection or an image connection.</td>
</tr>
<tr>
<td>Constant in Hexadecimal (Decimal) Equate Symbol</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>5 (5) HWI_OPERSTAT</td>
<td>Requests to retrieve the current status. <strong>Note:</strong> The input connect token represents a CPC connection or an image connection.</td>
<td></td>
</tr>
<tr>
<td>6 (6) HWI_ACCSTAT</td>
<td>Requests to retrieve the acceptable status values. <strong>Note:</strong> The input connect token represents a CPC connection or an image connection.</td>
<td></td>
</tr>
<tr>
<td>7 (7) HWI_APROF</td>
<td>Requests to retrieve the next activation reset profile name. <strong>Note:</strong> The input connect token must represent a CPC connection or an image connection.</td>
<td></td>
</tr>
<tr>
<td>8 (8) HWI_LUAPROF</td>
<td>Requests to retrieve the last used activation profile. <strong>Note:</strong> The input connect token must represent a CPC connection or an image connection.</td>
<td></td>
</tr>
<tr>
<td>9 (9) HWI_OBJTYPE</td>
<td>Requests to retrieve the object type. <strong>Input connect token represents</strong></td>
<td><strong>Returns</strong></td>
</tr>
<tr>
<td></td>
<td>CPC</td>
<td>HWMCA_CPC_OBJECT</td>
</tr>
<tr>
<td></td>
<td>CPC image</td>
<td>HWMCA_CPC_IMAGE_OBJECT</td>
</tr>
<tr>
<td></td>
<td>Capacity record</td>
<td>HWMCA_CAPACITY_RECORD</td>
</tr>
<tr>
<td></td>
<td>Reset activation profile</td>
<td>HWMCA_ACT_PROFILE_RESET</td>
</tr>
<tr>
<td></td>
<td>Image activation profile</td>
<td>HWMCA_ACT_PROFILE_IMAGE</td>
</tr>
<tr>
<td></td>
<td>Load activation profile</td>
<td>HWMCA_ACT_PROFILE_LOAD</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The input connect token must represent a CPC connection, an image connection, a caprec connection, a reset activation profile connection, an image activation profile connection, or a load activation profile connection.</td>
<td></td>
</tr>
<tr>
<td>A (10) HWI_IMLMODE</td>
<td>Requests to retrieve the initial machine load (IML) mode (LPAR). <strong>Note:</strong> The input connect token must only represent a CPC connection or an image connection.</td>
<td></td>
</tr>
<tr>
<td>B-16 (11–22) RESERVED</td>
<td>Reserved for attributes that are common to CPC and image connections unless otherwise noted.</td>
<td></td>
</tr>
<tr>
<td>17 (23) HWI_IPADDR</td>
<td>Requests to retrieve the internet address. <strong>Note:</strong> The input connect token must only represent a CPC connection.</td>
<td></td>
</tr>
<tr>
<td>18 (24) HWI_SNAADDR</td>
<td>Requests to retrieve the SNA address (Netid.NAU). <strong>Note:</strong> The input connect token must only represent a CPC connection.</td>
<td></td>
</tr>
<tr>
<td>19 (25) HWI_MMODEL</td>
<td>Requests to retrieve the machine model. <strong>Note:</strong> The input connect token must only represent a CPC connection.</td>
<td></td>
</tr>
<tr>
<td>Constant in Hexadecimal (Decimal)</td>
<td>Equate Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1A (26)</td>
<td>HWI_MTYPE</td>
<td>Requests to retrieve the machine type.</td>
</tr>
<tr>
<td>1B (27)</td>
<td>HWI_MSERIAL</td>
<td>Requests to retrieve the machine serial.</td>
</tr>
<tr>
<td>1C (28)</td>
<td>HWI_CPCSERIAL</td>
<td>Requests to retrieve the CPC serial number.</td>
</tr>
<tr>
<td>1D (29)</td>
<td>HWI_CPCID</td>
<td>Requests to retrieve the CPC identifier.</td>
</tr>
<tr>
<td>1E (30)</td>
<td>HWI_RESERVEID</td>
<td>Requests to retrieve the name of the application that is holding the reserve (if any).</td>
</tr>
<tr>
<td>1F (31)</td>
<td>HWI_SVCEREQD</td>
<td>Requests to retrieve the service required.</td>
</tr>
<tr>
<td>20 (32)</td>
<td>HWI_CBUINSTD</td>
<td>Requests to retrieve the CBU installed.</td>
</tr>
<tr>
<td>21 (33)</td>
<td>HWI_CBUENABLD</td>
<td>Requests to retrieve the CBU enabled.</td>
</tr>
<tr>
<td>22 (34)</td>
<td>HWI_CBUACTIVE</td>
<td>Requests to retrieve the CBU activated.</td>
</tr>
<tr>
<td>23 (35)</td>
<td>HWI_CBUACTDT</td>
<td>Requests to retrieve the CBU activation date.</td>
</tr>
<tr>
<td>24 (36)</td>
<td>HWI_CBUEXPDT</td>
<td>Requests to retrieve the CBU expiration date.</td>
</tr>
<tr>
<td>25 (37)</td>
<td>HWI_CBUTESTAR</td>
<td>Requests to retrieve the CBU tests left (test activations remaining).</td>
</tr>
<tr>
<td>Constant in Hexadecimal (Decimal) Equate Symbol</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
</tbody>
</table>
| 26 (38) HWI_CBUREALAV | Requests to retrieve the CBU real activation available.  
**Note:** The input connect token must only represent a CPC connection. |
| 27 (39) HWI_PRUNTYPE | Requests to retrieve the processor running time type.  
**Note:** The input connect token must only represent a CPC connection or a reset activation profile connection. |
| 28 (40) HWI_PRUNTIME | Requests to retrieve the processor running time.  
**Note:** The input connect token must only represent a CPC connection or a reset activation profile connection. |
| 29 (41) HWI_PRUNTSEW | Requests to retrieve the processor running time slice end wait processing.  
**Note:** The input connect token must only represent a CPC connection or a reset activation profile connection. |
| 2A (42) HWI_OOCINST | Requests to retrieve the on and off capacity on demand installed.  
**Note:** The input connect token must only represent a CPC connection. |
| 2B (43) HWI_OOCACT | Requests to retrieve the on and off capacity on demand currently activated.  
**Note:** The input connect token must only represent a CPC connection. |
| 2C (44) HWI_OOCENAB | Requests to retrieve the on and off capacity on demand enabled.  
**Note:** The input connect token must only represent a CPC connection. |
| 2D (45) HWI_OOCADT | Requests to retrieve the on and off capacity on demand activation date.  
**Note:** The input connect token must only represent a CPC connection. |
| 2E (46) HWI_PCPCSWM | Requests to retrieve the permanent CPC software model.  
**Note:** The input connect token must only represent a CPC connection. |
| 2F (47) HWI_PPBPSWM | Requests to retrieve the permanent plus billable processor software model.  
**Note:** The input connect token must only represent a CPC connection. |
| 30 (48) HWI_PPTPSWM | Requests to retrieve the permanent plus (all) temporary processor software model.  
**Note:** The input connect token must only represent a CPC connection. |
| 31 (49) HWI_PCPCMSU | Requests to retrieve the permanent CPC millions of service units (MSU) value.  
**Note:** The input connect token must only represent a CPC connection. |
<table>
<thead>
<tr>
<th>Constant in Hexadecimal (Decimal) Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
</table>
| 32 (50)                                       | Requests to retrieve the permanent plus billable processor MSU value.  
  Note: The input connect token must only represent a CPC connection. |
|     HWI_PPBPMSU                                |             |
| 33 (51)                                       | Requests to retrieve the permanent plus (all) temporary processor MSU value.  
  Note: The input connect token must only represent a CPC connection. |
|     HWI_PPTPMSU                                |             |
| 34 (52)                                       | Requests to retrieve the number of general purpose processors.  
  Note: The input connect token must only represent a CPC connection. |
|     HWI_NUMGPP                                 |             |
| 35 (53)                                       | Requests to retrieve the number of service assist processors.  
  Note: The input connect token must only represent a CPC connection. |
|     HWI_NUMSAP                                 |             |
| 36 (54)                                       | Requests to retrieve the number of the integrated facility for applications (IFA) processors.  
  Note: The input connect token must only represent a CPC connection. |
|     HWI_NUMIFAP                                |             |
| 37 (55)                                       | Requests to retrieve the number of the integrated facility for Linux® (IFL) processors.  
  Note: The input connect token must only represent a CPC connection. |
|     HWI_NUMIFLP                                |             |
| 38 (56)                                       | Requests to retrieve the number of the internal coupling facility (ICF) processors.  
  Note: The input connect token must only represent a CPC connection. |
|     HWI_NUMICFP                                |             |
| 39 (57)                                       | Requests to retrieve the number of integrated information processors (IIP).  
  Note: The input connect token must only represent a CPC connection. |
|     HWI_NUMIIPP                                |             |
| 3A (58)                                       | Requests to retrieve the number of defective (faulty) processors.  
  Note: The input connect token must only represent a CPC connection. |
|     HWI_NUMFLTYP                               |             |
| 3B (59)                                       | Requests to retrieve the number of spare processors.  
  Note: The input connect token must only represent a CPC connection. |
|     HWI_NUMSPARE                               |             |
| 3C (60)                                       | Requests to retrieve the number of pending (activation) processors.  
  Note: The input connect token must only represent a CPC connection. |
|     HWI_NUMPENDP                               |             |
| 3D (61)                                       | Requests to determine if activate/deactivate of capacity are permitted.  
  Note: The input connect token must only represent a CPC connection. |
<p>|     HWI_CAPCHGALLWD                            |             |</p>
<table>
<thead>
<tr>
<th>Constant in Hexadecimal (Decimal) Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3E (62)</td>
<td>Requests to retrieve degraded status. <strong>Note:</strong> The input connect token must only represent a CPC connection.</td>
</tr>
<tr>
<td>3F-68 (63-104)</td>
<td>Reserved for CPC attributes unless otherwise noted.</td>
</tr>
<tr>
<td>69 (105)</td>
<td>Requests to retrieve the parent (CPC) name. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>6A (106)</td>
<td>Requests to retrieve the SW operating system name. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>6B (107)</td>
<td>Requests to retrieve the SW operating System Type. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>6C (108)</td>
<td>Requests to retrieve the SW operating system level. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>6D (109)</td>
<td>Requests to retrieve the SW sysplex name (z/OS only). <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>6E (110)</td>
<td>Requests to retrieve the LPAR cluster name. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>6F (111)</td>
<td>Requests to retrieve the partition ID. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>70 (112)</td>
<td>Requests to retrieve the current defined capacity. <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>71 (113)</td>
<td>Requests to retrieve the shared general processor initial processing weight (SGPIPW). <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>72 (114)</td>
<td>Requests to retrieve the SGPIPW to be capped or not capped. <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>Constant in Hexadecimal (Decimal) Equate Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>73 (115) HWI_SGPPWMIN</td>
<td>Requests to retrieve the minimum SGPPW value. <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>74 (116) HWI_SGPPWMAX</td>
<td>Requests to retrieve the maximum SGPPW value. <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>75 (117) HWI_SGPPW</td>
<td>Requests to retrieve the current SGPPW value. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>76 (118) HWI_SGPPWCAP</td>
<td>Requests to retrieve the SGPPW to be capped or not capped. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>77 (119) HWI_WLM</td>
<td>Requests to retrieve whether WLM is allowed to change processing weight-related attributes. <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>78 (120) HWI_IFAIPW</td>
<td>Requests to retrieve the integrated facility for applications initial processing weight (IFAIPW). <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>79 (121) HWI_IFAIPWCAP</td>
<td>Requests to retrieve the IFAIPW to be capped or not capped. <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>7A (122) HWI_IFAPWMIN</td>
<td>Requests to retrieve the minimum IFAPW value. <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>7B (123) HWI_IFAPWMAX</td>
<td>Requests to retrieve the maximum IFAPW value. <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>7C (124) HWI_IFAPW</td>
<td>Requests to retrieve the current IFAPW value. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>7D (125) HWI_IFAPWCAP</td>
<td>Requests to retrieve the IFAPW to be currently capped or not capped. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>7E (126) HWI_IFLIPW</td>
<td>Requests to retrieve the integrated facility for Linux initial processing weight. <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>Constant in Hexadecimal (Decimal) Equate Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 7F (127) HWI_IFLPWCAP                         | Requests to retrieve the IFLPW to be capped or not capped.  

**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
| 80 (128) HWI_IFLPWMIN                         | Requests to retrieve the minimum IFLPW value.  

**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
| 81 (129) HWI_IFLPWMAX                         | Requests to retrieve the maximum IFLPW value.  

**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
| 82 (130) HWI_IFLPW                            | Requests to retrieve current IFLPW value.  

**Note:** The input connect token must only represent an image connection. |
| 83 (131) HWI_IFLPWCAP                         | Requests to retrieve the IFLPW to be capped or not capped.  

**Note:** The input connect token must only represent an image connection. |
| 84 (132) HWI_ICFIPW                           | Requests to retrieve the internal coupling facility initial processing weight (ICFIPW).  

**Note:** The input connect token must only represent an image connection (Coupling Facility Images only) or an image activation profile connection. |
| 85 (133) HWI_ICFIPWCAP                        | Requests to retrieve the ICFPW be capped or not capped.  

**Note:** The input connect token must only represent an image connection (Coupling Facility Images only) or an image activation profile connection. |
| 86 (134) HWI_ICFPWMIN                        | Requests to retrieve the minimum ICFPW value.  

**Note:** The input connect token must only represent an image connection (Coupling Facility Images only) or an image activation profile connection. |
| 87 (135) HWI_ICFPWMAX                        | Requests to retrieve the maximum ICFPW value.  

**Note:** The input connect token must only represent an image connection (Coupling Facility Images only) or an image activation profile connection. |
| 88 (136) HWI_ICFPW                            | Requests to retrieve the current ICFPW value.  

**Note:** The input connect token must only represent an image connection (Coupling Facility Images only). |
| 89 (137) HWI_ICFPWCAP                        | Requests to retrieve the ICFPW to be capped or not capped.  

**Note:** The input connect token must only represent an image connection (Coupling Facility Images only). |
| 8A (138) HWI_IPIPIPW                         | Requests to retrieve the integrated information processors initial processing weight (IIPIPW).  

**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
<table>
<thead>
<tr>
<th>Constant in Hexadecimal (Decimal) Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8B (139) HWI_IPIPWCAP</td>
<td>Requests to retrieve the IIPIPW be capped or not capped. <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>8C (140) HWI_IIPPWMIN</td>
<td>Requests to retrieve the minimum IIPPW value. <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>8D (141) HWI_IIPPWMAX</td>
<td>Requests to retrieve the maximum IIPPW value. <strong>Note:</strong> The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>8E (142) HWI_IIPPW</td>
<td>Requests to retrieve the current IIPPW value. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>8F (143) HWI_IIPPWCAP</td>
<td>Requests to retrieve the IIPPW to be capped or not capped. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>90 (144) HWI_IPLTOKEN</td>
<td>Requests to retrieve the IPL token associated with the current IPL of the image targeted. <strong>Note:</strong> The input connect token must only represent an image connection.</td>
</tr>
<tr>
<td>91 (145) HWI_PSWS</td>
<td>Requests to retrieve the program status word (PSW) for each of the central processors (CP) associated with this image. The returned data is mapped as follows:</td>
</tr>
<tr>
<td></td>
<td>Field</td>
</tr>
<tr>
<td></td>
<td>Number of CPs</td>
</tr>
<tr>
<td></td>
<td>For each CP, the following is returned:</td>
</tr>
<tr>
<td></td>
<td>CPID</td>
</tr>
<tr>
<td></td>
<td>PSW</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The input connect token must represent an image connection. The query parameter for this attribute must specify a data area large enough to contain all of the above structure (that is 32 bits +96 bits per CP). For example, if there are 4 CPs on the targeted image, the attributevaluelen specified for this attribute must be 32 + (96 x 4) = 416 bits (52 bytes).</td>
</tr>
<tr>
<td>92-B6 (146-182) RESERVED</td>
<td>Reserved for image attributes unless otherwise noted.</td>
</tr>
<tr>
<td>B7 (183) HWI_RECID</td>
<td>Requests to retrieve the record ID. <strong>Note:</strong> The input connect token must only represent a capacity record.</td>
</tr>
<tr>
<td>B8 (184) HWI_RECTYPE</td>
<td>Requests to retrieve the record type. <strong>Note:</strong> The input connect token must only represent a capacity record.</td>
</tr>
<tr>
<td>Constant in Hexadecimal (Decimal)</td>
<td>Equate Symbol</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------</td>
</tr>
</tbody>
</table>
| B9 (185)                        | HWI_ACTSTAT   | Requests to retrieve the record activation status.  
  **Note:** The input connect token must only represent a *capacity record.* |
| BA (186)                        | HWI_ACTDATE   | Requests to retrieve the record activation date.  
  **Note:** The input connect token must only represent a *capacity record.* |
| BB (187)                        | HWI_EXPDATE   | Requests to retrieve the record expiration date.  
  **Note:** The input connect token must only represent a *capacity record.* |
| BC (188)                        | HWI_ACTEXP    | Requests to retrieve the record activation expiration date.  
  **Note:** The input connect token must only represent a *capacity record.* |
| BD (189)                        | HWI_MAXRADS   | Requests to retrieve the maximum real activation days.  
  **Note:** The input connect token must only represent a *capacity record.* |
| BE (190)                        | HWI_MAXTADS   | Requests to retrieve the maximum test activation days.  
  **Note:** The input connect token must only represent a *capacity record.* |
| BF (191)                        | HWI_REMRADS   | Requests to retrieve the remaining real activation days.  
  **Note:** The input connect token must only represent a *capacity record.* |
| C0 (192)                        | HWI_REMTADS   | Requests to retrieve the remaining test activation days.  
  **Note:** The input connect token must only represent a *capacity record.* |
| C1 (193)                        | HWI_OOCODREC  | Request to retrieve all aspects of a capacity record in XML format.  
  **Note:** The input connect token must only represent a *capacity record.* |
| C2-C8 (194-200)                 | RESERVED      | Reserved for capacity record attributes. |
| C9 (201)                        | HWI_IOCDS     | Requests to retrieve the IOCDS.  
  **Note:** The input connect token must represent a *reset activation profile.* |
| CA (202)                        | HWI_IPL_ADDRESS | Requests to retrieve the IPL address.  
  **Note:** The input connect token must represent an *image activation profile* or a *load activation profile.* |
<table>
<thead>
<tr>
<th>Constant in Hexadecimal (Decimal) Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
</table>
| CB 203 HWI_IPL_PARM | Requests to retrieve the IPL parameter.  
*Note:* The input connect token must represent an image activation profile or a load activation profile. |
| CC 204 HWI_IPL_TYPE | Requests to retrieve the IPL type for the activation profile.  
*Note:* The input connect token must represent an image activation profile or a load activation profile. |
| CD 205 HWI_WW_PORTNAME | Requests to retrieve the worldwide port name for the activation profile.  
*Note:* The input connect token must represent an image activation profile or a load activation profile. |
| CE 206 HWI_BOOT_PGM_SELECTOR | Requests to retrieve the boot program selector for the activation profile.  
*Note:* The input connect token must represent an image activation profile or a load activation profile. |
| CF 207 HWI_LU_NUM | Requests to retrieve the logical unit number value for the activation profile.  
*Note:* The input connect token must represent an image activation profile or a load activation profile. |
| D0 208 HWI_BOOTREC_BLK_ADDR | Requests to retrieve the boot record logical block address for the activation profile.  
*Note:* The input connect token must represent an image activation profile or a load activation profile. |
| D1 209 HWI_OPSYS_LOADPARM | Requests to retrieve the operating system specific load parameter.  
*Note:* The input connect token must represent an image activation profile or a load activation profile. |
| D2 210 HWI_GROUP_PROF_NAME | Requests to retrieve the name of the group capacity profile that is to be used for the CPC image object activated with this profile.  
*Note:* The input connect token must represent an image activation profile. |
| D3 211 HWI_LOAD_AT_ACTIVATION | Requests to retrieve the indicator if the CPC image object activated with this profile should be loaded (IPLed) at the end of the activation.  
*Note:* The input connect token must represent an image activation profile. |
| D4 212 HWI_CENTRAL_STOR | Requests to retrieve the initial amount of central storage (in megabytes) to be used for the CPC image object activated with this profile.  
*Note:* The input connect token must represent an image activation profile. |
| D5 213 HWI_RES_CENTRAL_STOR | Requests to retrieve the reserved amount of central storage (in megabytes) to be used for the CPC image object activated with this profile.  
*Note:* The input connect token must represent an image activation profile. |
| D6 214 HWI_EXPANDED_STOR | Requests to retrieve the initial amount of expanded storage (in megabytes) to be used for the CPC image object activated with this profile.  
*Note:* The input connect token must represent an image activation profile. |
<table>
<thead>
<tr>
<th>Constant in Hexadecimal (Decimal) Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
</table>
| D7 215 HWI_RES_EXPANDED_STOR                  | Requests to retrieve the reserved amount of expanded storage (in megabytes) to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an *image activation profile*. |
| D8 216 HWI_NUM_GPP                           | Requests to retrieve the number of dedicated general purpose processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an *image activation profile*. |
| D9 217 HWI_NUM_RESGPP                        | Requests to retrieve the number of reserved dedicated general purpose processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an *image activation profile*. |
| DA 218 HWI_NUM_IFA                           | Requests to retrieve the number of dedicated integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an *image activation profile*. |
| DB 219 HWI_NUM_RESIFA                        | Requests to retrieve the number of reserved dedicated integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an *image activation profile*. |
| DC 220 HWI_NUM_IFL                           | Requests to retrieve the number of dedicated integrated facility for Linux (IFL) general purpose processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an *image activation profile*. |
| DD 221 HWI_NUM_RESIFL                        | Requests to retrieve the number of reserved dedicated integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an *image activation profile*. |
| DE 222 HWI_NUM_ICF                           | Requests to retrieve the number of dedicated internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an *image activation profile*. |
| DF 223 HWI_NUM_RESICF                        | Requests to retrieve the number of reserved dedicated internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an *image activation profile*. |
| E0 224 HWI_NUM_ZIIP                          | Requests to retrieve the number of dedicated integrated information processors (zIIP) to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an *image activation profile*. |
| E1 225 HWI_NUM_RESZIIP                       | Requests to retrieve the number of reserved dedicated integrated information processors (zIIP) to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an *image activation profile*. |
| E2 226 HWI_NUM_SHARED_GPP                    | Requests to retrieve the number of shared general purpose processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an *image activation profile*. |
<table>
<thead>
<tr>
<th>Constant in Hexadecimal (Decimal) Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E3 227 HWI_NUM_RES_SHARED_GPP</td>
<td>Requests to retrieve the number of reserved shared general purpose processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an <em>image activation profile</em>.</td>
</tr>
<tr>
<td>E4 228 HWI_NUM_SHARED_IFA</td>
<td>Requests to retrieve the number of shared integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an <em>image activation profile</em>.</td>
</tr>
<tr>
<td>E5 229 HWI_NUM_RES_SHARED_IFA</td>
<td>Requests to retrieve the number of reserved shared integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an <em>image activation profile</em>.</td>
</tr>
<tr>
<td>E6 230 HWI_NUM_SHARED_IFL</td>
<td>Requests to retrieve the number of shared integrated facility for Linux (IFL) general purpose processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an <em>image activation profile</em>.</td>
</tr>
<tr>
<td>E7 231 HWI_NUM_RES_SHARED_IFL</td>
<td>Requests to retrieve the number of reserved shared integrated facility for Linux (IFL) general purpose processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an <em>image activation profile</em>.</td>
</tr>
<tr>
<td>E8 232 HWI_NUM_SHARED_ICF</td>
<td>Requests to retrieve the number of shared internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an <em>image activation profile</em>.</td>
</tr>
<tr>
<td>E9 233 HWI_NUM_RES_SHARED_ICF</td>
<td>Requests to retrieve the number of reserved shared internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an <em>image activation profile</em>.</td>
</tr>
<tr>
<td>EA 234 HWI_NUM_SHARED_ZIIP</td>
<td>Requests to retrieve the number of shared integrated information processors (zIIP) to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an <em>image activation profile</em>.</td>
</tr>
<tr>
<td>EB 235 HWI_NUM_RES_SHARED_ZIIP</td>
<td>Requests to retrieve the number of reserved shared integrated information processors (zIIP) to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an <em>image activation profile</em>.</td>
</tr>
<tr>
<td>EC-FA 236–250 RESERVED</td>
<td>Reserved for activation profile attributes.</td>
</tr>
</tbody>
</table>

**NumOfAttribute**

Supplied parameter

- **Type:** integer
- **Length:** 4 bytes

**NumOfAttribute** specifies the number of attributes to be queried. The maximum number of attributes allowed is 64.
DiagArea

Returned parameter
- Type: character
- Length: 32 bytes

DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the DiagArea might not be filled in, and the data returned in the area should be ignored.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag_Index</td>
<td>32-bit integer</td>
<td>The array index to the parameter field that causes the error.</td>
</tr>
<tr>
<td>Diag_Key</td>
<td>32-bit integer</td>
<td>The constant value represents the field that causes the error.</td>
</tr>
<tr>
<td>Diag_Actual</td>
<td>32-bit integer</td>
<td>The incorrect actual value specified.</td>
</tr>
<tr>
<td>Diag_Expected</td>
<td>32-bit integer</td>
<td>The expected value to be used.</td>
</tr>
<tr>
<td>Diag_CommErr</td>
<td>32-bit integer</td>
<td>The returned code that is returned from the console application API or the BCPii transport layer.</td>
</tr>
<tr>
<td>Diag_Text</td>
<td>character (12)</td>
<td>Additional diagnostic information in text format.</td>
</tr>
</tbody>
</table>

ABEND Codes
The call might result in an abend X'042' with a reason code of X'0006yyyy' due to one of the following reasons:

<table>
<thead>
<tr>
<th>yyyy</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The parameters passed by the caller are not in the primary address space.</td>
</tr>
<tr>
<td>0001</td>
<td>The parameters passed by the caller are not accessible.</td>
</tr>
<tr>
<td>0002</td>
<td>The number of parameters passed by the caller is not correct.</td>
</tr>
</tbody>
</table>

Return Codes
When the service returns control to the caller, GPR 15 and ReturnCode contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return Code in Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 HWI_OK</td>
<td>Meaning: Successful completion.</td>
</tr>
<tr>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>Return Code in Hexadecimal Equate Symbol</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| 100 HWI_CONNECT_TOKEN_INV | **Meaning:** Program error. The specified connect token is not valid. This return code indicates that one of the following conditions has occurred:  
- The connect token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken.  
- The connect token does not represent an active connection. The connection specified might have already been disconnected using the HWIDISC service call.  
- The connect token is not associated with the address space of the caller. The ConnectToken specified is associated with a different address space than the caller of this service call.  
**Action:** Check for probable coding error. |
| 101 HWI_COMMUNICATION_ERROR | **Meaning:** A communication error is detected. The hardware management console application API (HWMCA) or the BCPii transport layer has returned with a failing return code.  
**Action:** See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPii transport layer.  
HWMCA API and BCPii transport return codes are provided in Appendix A, “BCPii Communication Error Reason Codes,” on page A-1. |
| 102 HWI_DIAGAREA_INV | **Meaning:** Program error. The DiagArea is not accessible.  
**Action:** Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field. |
| 103 HWI_CONNECT_TOKEN_INACTIVE | **Meaning:** The specified connect token is no longer valid. The connection has been disconnected or it is in the progress of being disconnected.  
**Action:** Check for probable coding error. Verify that the specified connect token is still active. If connectivity to the targeted CPC connection no longer exists, all connections associated with that CPC will no longer have a connect token that can be used. |
<table>
<thead>
<tr>
<th>Return Code in Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 401 HWI_QUERYPARM_ATTRIB_INV            | **Meaning:** Program error. One of the requested attribute identifiers in the QueryParm is not valid. The system rejects the service call. This return code indicates that one of the following conditions has occurred:  
  - The Query attribute identifier specified is not in the acceptable value range of possible attributes.  
  - The specified Query attribute identifier has been provided with an incompatible connection type. For example, the attribute identifier applies only to CPC connections, but the ConnectToken specified represents an image connection, a capacity record connection, or any of the activation profile connections.  
  **Action:** Check for probable coding error. Validate that the Query attribute specified is in the valid range of possible values. Validate that the Query attribute specified is permitted for the specified connection type. See the DiagArea for further diagnostic information:  
    - The Diag_Index field specifies the index of the element in the attribute array that is in error.  
    - The Diag_Key contains the attribute identifier specified.  
    - The Diag_Text contains “Invalid Attr” if the attribute is one whose value cannot be queried. If the attribute cannot be queried for the specified connection type, the Diag_Text contains “Mismatch.” |
| 402 HWI_QUERYPARM_INACCESSIBLE         | **Meaning:** Program error. The QueryParm data area is either partially or completely inaccessible by the application, the Base Control Program internal interface (BCPii) address space, or both.  
  **Action:** Check for probable coding error. Consider the following possibilities:  
    - The QueryParm length could be too small. The size of QueryParm must be at least the product of the NumofAttributes parameter and the length of the data area mapping for each attribute (16 bytes).  
    - The NumofAttributes value can be larger than the number of parameters actually passed. |
| 403 HWI_QUERYPARM_ATTRIBRETADDR _INACCESSIBLE | **Meaning:** Program error. Storage that is pointed to by one or more of the attribute value pointers in the QueryParm is not accessible by the application. The system is not able to return data for this attribute identifier. Partial data might have already been returned.  
  **Action:** Check for probable coding error. See the DiagArea for further diagnostic information. The Diag_Index field specifies the array index that contained the inaccessible AttributeValuePtr. The Diag_Key contains the erroneous attribute identifier. |
<table>
<thead>
<tr>
<th>Return Code in Hexadecimal</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 404 HWI_QUERYPARM_ATTRIB_LENGTH_INV | **Meaning:** Program error. One of the attribute lengths specified is too small. There is not enough space to contain all of the returned data for this particular attribute. The system returns partial data, filling in the attribute data area for the length specified.  
**Action:** Check for probable coding error. See the DiagArea for further diagnostic information. The Diag_Index field specifies the array index which contained the partially filled-in value. The Diag_Key is the attribute identifier constant that causes the error. The Diag_Actual indicates the application-specified length. The Diag_Expected indicates the size required for the returned data. |
| 405 HWI_QUERY_NUMOFATTRIB_INV | **Meaning:** Program error. The NumOfAttribute specified in the call is not valid. The NumOfAttribute must be greater than zero and smaller than 64.  
**Action:** Check for probable error. Verify that the NumOfAttribute specified is greater than zero and smaller than 64. |
| 406 HWI_QUERY_ATTRIBUTE_NOT_SUPPORTED | **Meaning:** The targeted hardware of the HWIQUERY request does not recognize the attribute attempted to be retrieved.  
**Action:** Verify that the targeted hardware is at a level that supports the type of attribute being queried. |
| F00 HWI_NOT_AVAILABLE | **Meaning:** BCPIi services are not available, and the system rejects the service request.  
**Action:** Notify the system programmer to start the BCPIi address space and try the request again. See "Restarting the HWIBCPIi Address Space" on page 14-8 about how to start the BCPIi address space.  
Programs can also listen to ENF68 to determine when BCPIi services are available. See **z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG** for how to listen for BCPIi activation messages. |
| F01 HWI_AUTH_FAILURE | **Meaning:** The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.  
**Action:** Check the calling program for a probable coding error. |
Return Code in Hexadecimal Equate Symbol | Meaning and Action
---|---
F02 HWI_NO_SAF_AUTH | **Meaning:** The user does not have correct SAF authorization for the request.
**Action:** Check for probable error. Consider one or more of the following possible actions:
- Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for CPC and all activation profile connections.
- Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for an image connections.
- Define read access authorization to the FACILITY class resource profile HWI.CAPREC.netid.nau.caprecid for a capacity record connection.
- Ensure that the referenced facility class profile is RACLIST-specified.

F03 HWI_INTERRUPT_STATUS_INV | **Meaning:** The calling program is disabled. The system rejects this service request.
**Action:** Check the calling program for a probable coding error.

F04 HWI_MODE_INV | **Meaning:** The calling program is not in task mode. The system rejects this service request.
**Action:** Check the calling program for a probable error.

F05 HWI_LOCKS_HELD | **Meaning:** The calling program is holding one or more locks. The system rejects this service request.
**Action:** Check the calling program for a probable coding error.

F06 HWI_UNSUPPORTED_RELEASE | **Meaning:** The system level does not support this service. The system rejects this service request.
**Action:** Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again.

FFF HWI_UNEXPECTED_ERROR | **Meaning:** System error. The service that was called encountered an unexpected error. The system rejects the service call.
**Action:** Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.

**Example**
In the pseudocode example, the caller issues a call to retrieve the CPC name and the Current CPC status of a CPC:

```
  QueryParm_Ptr = ADDR(QueryParm);
  NumberOfAttributes = 2;
  QueryParm(1).Attribute = HWI_NAME;
  QueryParm(1).AttributeValue_Ptr = Addr(Value1);
  QueryParm(1).AttributeValueLen = length of value1;
  QueryParm(2).Attribute = HWI_OPERSTAT;
  QueryParm(2).AttributeValue_Ptr = Addr(Value2);
  QueryParm(2).AttributeValueLen = 4;
```
CALL HWIQUERY (ReturnCode, ConnectToken, QueryParm_Ptr, NumOfAttributes, DiagArea)

HWISET — BCPii Set SE/HMC-Managed Objects Data

Call the HWISET service to change or set data for Hardware Management Console (HMC)-managed objects associated with Central Processor Complexes (CPCs), CPC images (LPARs), or activation profiles.

Description

Environment

The requirements for the callers are:

- Minimum authorization: PKM allowing key 0-7, or supervisor state
- Dispatchable unit mode: Task
- Cross memory mode: Any PASN, any HASN, any SASN
- AMODE: 31-bit
- ASC mode: Primary or access register (AR)
- 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

See “Syntax, Linkage and Programming Considerations” on page 14-9 for details about how to call BCPii services in the various programming languages.

Restrictions

None.

Authorization

The client application must have at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV. This class resource grants the application access to consult to the local CPC.

In addition, the client application must have at least update access to the SAF-protected FACILITY class resource profile HWI.TARGET.netid.nau for setting CPC-related or activation profile-related values, or HWI.TARGET.netid.nau.imagename for setting image-related values.

Note: BCPii require the FACILITY class to be RACLIST-specified.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.
CALL HWISET
  (ReturnCode
   ,ConnectToken
   ,SetType
   ,SetTypeValue_Ptr
   ,SetTypeValueLen
   ,DiagArea)

Parameters
The parameters are explained as follows:

ReturnCode
  Returned parameter
  • Type: Integer
  • Length: 4 bytes
  
  ReturnCode contains the return code from the service.

ConnectToken
  Supplied parameter
  • Type: Character string
  • Length: 16 bytes
  
  ConnectToken represents a logical connection between the application and a
  CPC, image, or activation profile. The ConnectToken is an output parameter on
  the HWICONN service call.
  
  The ConnectToken specified must have originated from a HWICONN service
  call that was issued from the same address space as this service call.

SetType
  Supplied parameter
  • Type: Integer
  • Length: 4 bytes
  
  SetType specifies the type of set request.
  
  The following table is the list of valid set types. See the System z9 and eServer
  zSeries Application Programming Interfaces (SB10-7030-08) and System z10
  and eServer zSeries Application Programming Interfaces (SB10-7030-09)
  documentation for more information.

<table>
<thead>
<tr>
<th>Constant in:</th>
<th>Hexadecimal (Decimal)</th>
<th>Equate</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>(6)</td>
<td></td>
<td>HWI_ACCSTAT</td>
<td>Requests to change or set the acceptable CPC status values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Note: The input connect token represents a CPC connection or an image connection.</td>
</tr>
<tr>
<td>7</td>
<td>(7)</td>
<td></td>
<td>HWI_APROF</td>
<td>Requests to change or set the next activation reset profile name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Note: The input connect token represents a CPC connection or an image connection.</td>
</tr>
<tr>
<td>Constant in: Hexadecimal (Decimal) Equate Symbol</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 27 (39) HWI_PRUNTYPE                           | Requests to change or set the processor running time type.  
**Note:** The input connect token represents a CPC connection or a reset activation profile connection. |
| 28 (40) HWI_PRUNTIME                           | Requests to change or set the processor running time type.  
**Note:** The input connect token must only represent a CPC connection or a reset activation profile connection. |
| 29 (41) HWI_PRUNTSEW                           | Requests to change or set the processor running time slice end wait processing.  
**Note:** The input connect token must only represent a CPC connection or a reset activation profile connection. |
| 70 (112) HWI_DEFCAP                            | Requests to change or set the current defined capacity.  
**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
| 71 (113) HWI_SGPIPW                             | Requests to change or set the shared general processor initial processing weight (SGPIPW).  
**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
| 72 (114) HWI_SGPIPWCAP                          | Requests to change or set the SGPIPW to be capped or not capped.  
**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
| 73 (115) HWI_SGPPWMIN                           | Requests to change or set the minimum SGPPW value.  
**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
| 74 (116) HWI_SGPPWMAX                           | Requests to change or set the maximum SGPPW value.  
**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
| 77 (119) HWI_WLM                                | Requests to change or set whether WLM is allowed to change SGPPW values.  
**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
| 78 (120) HWI_IFAIPW                             | Requests to change or set the integrated facility for applications initial processing weight (IFAIPW).  
**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
| 79 (121) HWI_IFAIPWCAP                          | Requests to change or set the IFAIPW to be capped or not capped.  
**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
| 7A (122) HWI_IFAPWMIN                           | Requests to change or set the minimum IFAPW value.  
**Note:** The input connect token must only represent an image connection or an image activation profile connection. |
<table>
<thead>
<tr>
<th>Constant in: Hexadecimal (Decimal)</th>
<th>Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
</table>
| 7B (123)                           | HWI_IFAPWMAX  | Requests to change or set the maximum IFAPW value.  
Note: The input connect token must only represent an image connection or an image activation profile connection. |
| 7E (126)                           | HWI_IFLIPW    | Requests to change or set the integrated facility for Linux initial processing weight (IFLIPW).  
Note: The input connect token must only represent an image connection or an image activation profile connection. |
| 7F (127)                           | HWI_IFLIPWCAP | Requests to change or set the IFLIPW to be capped or not capped.  
Note: The input connect token must only represent an image connection or an image activation profile connection. |
| 80 (128)                           | HWI_IFLPWMIN  | Requests to change or set the minimum IFLPW value.  
Note: The input connect token must only represent an image connection or an image activation profile connection. |
| 81 (129)                           | HWI_IFLPWMAX  | Requests to change or set the maximum IFLPW value.  
Note: The input connect token must only represent an image connection or an image activation profile connection. |
| 84 (132)                           | HWI_ICFIPW    | Requests to change or set the internal coupling facility initial processing weight (ICFIPW).  
Note: The input connect token must only represent an image connection or an image activation profile connection. |
| 85 (133)                           | HWI_ICFIPWCAP | Requests to change or set the ICFIPW to be capped or not capped.  
Note: The input connect token must only represent an image connection or an image activation profile connection. |
| 86 (134)                           | HWI_ICFPWMIN  | Requests to change or set the minimum ICFW value.  
Note: The input connect token must only represent an image connection or an image activation profile connection. |
| 87 (135)                           | HWI_ICFPWMAX  | Requests to change or set the maximum ICFW value.  
Note: The input connect token must only represent an image connection or an image activation profile connection. |
| 8A (138)                           | HWI_IIPIPW    | Requests to change or set the integrated information processors initial processing weight (IIPIPW).  
Note: The input connect token must only represent an image connection or an image activation profile connection. |
| 8B (139)                           | HWI_IIPIPWCAP | Requests to change or set the IIPIPW to be capped or not capped.  
Note: The input connect token must only represent an image connection or an image activation profile connection. |
| 8C (140)                           | HWI_IIPPWMIN  | Requests to change or set the minimum IIPPW value.  
Note: The input connect token must only represent an image connection or an image activation profile connection. |
<table>
<thead>
<tr>
<th>Constant in: Hexadecimal (Decimal) Equate Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8D (141)</td>
<td>Requests to change or set the maximum IIPPW value.</td>
</tr>
<tr>
<td>HWI_IIPPWMAX</td>
<td><strong>Note</strong>: The input connect token must only represent an image connection or an image activation profile connection.</td>
</tr>
<tr>
<td>C9 (201)</td>
<td>Requests to change or set the IOCDS.</td>
</tr>
<tr>
<td>HWI_IOCDS</td>
<td><strong>Note</strong>: The input connect token must represent a reset activation profile.</td>
</tr>
<tr>
<td>CA (202)</td>
<td>Requests to change or set the IPL address.</td>
</tr>
<tr>
<td>HWI_IPL_ADDRESS</td>
<td><strong>Note</strong>: The input connect token must represent an image activation profile or a load activation profile.</td>
</tr>
<tr>
<td>CB (203)</td>
<td>Requests to change or set the IPL parameter.</td>
</tr>
<tr>
<td>HWI_IPL_PARM</td>
<td><strong>Note</strong>: The input connect token must represent an image activation profile or a load activation profile.</td>
</tr>
<tr>
<td>CC (204)</td>
<td>Requests to change or set the IPL type for the activation profile.</td>
</tr>
<tr>
<td>HWI_IPL_TYPE</td>
<td><strong>Note</strong>: The input connect token must represent an image activation profile or a load activation profile.</td>
</tr>
<tr>
<td>CD (205)</td>
<td>Requests to change or set the worldwide port name for the activation profile.</td>
</tr>
<tr>
<td>HWI_WW_PORTNAME</td>
<td><strong>Note</strong>: The input connect token must represent an image activation profile or a load activation profile.</td>
</tr>
<tr>
<td>CE (206)</td>
<td>Requests to change or set the boot program selector for the activation profile.</td>
</tr>
<tr>
<td>HWI_BOOT_PGM_SELECTOR</td>
<td><strong>Note</strong>: The input connect token must represent an image activation profile or a load activation profile.</td>
</tr>
<tr>
<td>CF (207)</td>
<td>Requests to change or set the logical unit number value for the activation profile.</td>
</tr>
<tr>
<td>HWI_LU_NUM</td>
<td><strong>Note</strong>: The input connect token must represent an image activation profile or a load activation profile.</td>
</tr>
<tr>
<td>D0 (208)</td>
<td>Requests to change or set the boot record logical block address for the activation profile.</td>
</tr>
<tr>
<td>HWI_BOOTREC_BLK_ADDR</td>
<td><strong>Note</strong>: The input connect token must represent an image activation profile or a load activation profile.</td>
</tr>
<tr>
<td>D1 (209)</td>
<td>Requests to change or set the operating system specific load parameter.</td>
</tr>
<tr>
<td>HWI_OPSYS_LOADPARM</td>
<td><strong>Note</strong>: The input connect token must represent an image activation profile or a load activation profile.</td>
</tr>
<tr>
<td>D2 (210)</td>
<td>Requests to change or set the name of the group capacity profile that is to be used for the CPC image object activated with this profile.</td>
</tr>
<tr>
<td>HWI_GROUP_PROF_NAME</td>
<td><strong>Note</strong>: The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>D3 (211)</td>
<td>Requests to change or set the indicator if the CPC image object activated with this profile should be loaded (IPLed) at the end of the activation.</td>
</tr>
<tr>
<td>HWI_LOAD_AT_ACTIVATION</td>
<td><strong>Note</strong>: The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>Constant in: Hexadecimal (Decimal) Equate Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>D4 212 HWI_CENTRAL_STOR</td>
<td>Requests to change or set the initial amount of central storage (in megabytes) to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>D5 213 HWI_RES_CENTRAL_STOR</td>
<td>Requests to change or set the reserved amount of central storage (in megabytes) to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>D6 214 HWI_EXPANDED_STOR</td>
<td>Requests to change or set the initial amount of expanded storage (in megabytes) to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>D7 215 HWI_RES_EXPANDED_STOR</td>
<td>Requests to change or set the reserved amount of expanded storage (in megabytes) to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>D8 216 HWI_NUM_GPP</td>
<td>Requests to change or set the number of dedicated general purpose processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>D9 217 HWI_NUM_RESGPP</td>
<td>Requests to change or set the number of reserved dedicated general purpose processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>DA 218 HWI_NUM_IFA</td>
<td>Requests to change or set the number of dedicated integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>DB 219 HWI_NUM_RESIFA</td>
<td>Requests to change or set the number of reserved dedicated integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>DC 220 HWI_NUM_IFL</td>
<td>Requests to change or set the number of dedicated integrated facility for Linux (IFL) general purpose processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>DD 221 HWI_NUM_RESIFL</td>
<td>Requests to change or set the number of reserved dedicated integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>DE 222 HWI_NUM_ICF</td>
<td>Requests to change or set the number of dedicated internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>DF 223 HWI_NUM_RESICF</td>
<td>Requests to change or set the number of reserved dedicated internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>Constant in: Hexadecimal (Decimal) Equate Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| E0 (224) HWI_NUM_ZIIP | Requests to change or set the number of dedicated integrated information processors (zIIP) to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an image activation profile. |
| E1 (225) HWI_NUM_RESZIIP | Requests to change or set the number of reserved dedicated integrated information processors (zIIP) to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an image activation profile. |
| E2 (226) HWI_NUM_SHARED_GPP | Requests to change or set the number of shared general purpose processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an image activation profile. |
| E3 (227) HWI_NUM_RES_SHARED_GPP | Requests to change or set the number of reserved shared general purpose processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an image activation profile. |
| E4 (228) HWI_NUM_SHARED_IFA | Requests to change or set the number of shared integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an image activation profile. |
| E5 (229) HWI_NUM_RES_SHARED_IFA | Requests to change or set the number of reserved shared integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an image activation profile. |
| E6 (230) HWI_NUM_SHARED_IFL | Requests to change or set the number of shared integrated facility for Linux (IFL) general purpose processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an image activation profile. |
| E7 (231) HWI_NUM_RES_SHARED_IFL | Requests to change or set the number of reserved shared integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an image activation profile. |
| E8 (232) HWI_NUM_SHARED_ICF | Requests to change or set the number of shared internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an image activation profile. |
| E9 (233) HWI_NUM_RES_SHARED_ICF | Requests to change or set the number of reserved shared internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an image activation profile. |
| EA (234) HWI_NUM_SHARED_ZIIP | Requests to change or set the number of shared integrated information processors (zIIP) to be used for the CPC image object activated with this profile.  
**Note:** The input connect token must represent an image activation profile. |
### Constant in: Hexadecimal (Decimal) Equate Symbol

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EB (235)</td>
<td>Requests to change or set the number of reserved shared integrated information processors (zIIP) to be used for the CPC image object activated with this profile. <strong>Note:</strong> The input connect token must represent an image activation profile.</td>
</tr>
<tr>
<td>EC-FA (236-250)</td>
<td>Reserved for activation profile attributes.</td>
</tr>
</tbody>
</table>

**SetTypeValue_Ptr**
- Supplied parameter
  - Type: Pointer
  - Length: 4 bytes

SetTypeValue_Ptr specifies address of the requested type data value to be set or changed.

The particular SetType determines what data value must be specified. See the chart below as well as the System z9 and eServer zSeries Application Programming Interfaces guide and System z10 and eServer zSeries Application Programming Interfaces guide for more details.

<table>
<thead>
<tr>
<th>SetTypes</th>
<th>Values to be specified</th>
</tr>
</thead>
</table>
| HWI_ACCSTAT | A 4-byte integer type value.  
For CPC connections, bit values can be set to:  
  - HWMCA_STATUS_OPERATING  
  - HWMCA_STATUS_NOT_OPERATING  
  - HWMCA_STATUS_NO_POWER  
  - HWMCA_STATUS_EXCEPTIONS  
  - HWMCA_STATUS_STATUS_CHECK  
  - HWMCA_STATUS_SERVICE  
  - HWMCA_STATUS_LINKNOTACTIVE  
  - HWMCA_STATUS POWERSAVE  
  - HWMCA_STATUS_SERVICE_REQ  
  - HWMCA_STATUS_DEGRADED  
For image connections, bit values can be set to:  
  - HWMCA_STATUS_OPERATING  
  - HWMCA_STATUS_NOT_OPERATING  
  - HWMCA_STATUS_NOT_ACTIVATED  
  - HWMCA_STATUS_EXCEPTIONS  
  - HWMCA_STATUS_STATUS_CHECK  
  - HWMCA_STATUS POWERSAVE  |
| HWI_APROF | A 16-character activation profile name padded with trailing blanks. |
| HWI_PRUNTYPE | A 4-byte integer type value.  
  - HWMCA_DETERMINED_SYSTEM  
  - HWMCA_DETERMINED_USER  |
<table>
<thead>
<tr>
<th>SetTypes</th>
<th>Values to be specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWI_PRUNTIME</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>A value between 1 to 100 for the user defined processor running time.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This value can only be set if the processor running time type (HWI_PRUNTYPE) is set to HWMCA_DETERMINED_USER.</td>
</tr>
<tr>
<td>HWI_PRUNTSEW</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>• HWMCA_TRUE</td>
</tr>
<tr>
<td></td>
<td>• HWMCA_FALSE</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This value can only be set if the processor running time type (HWI_PRUNTYPE) is set to HWMCA_DETERMINED_USER.</td>
</tr>
<tr>
<td>HWI_DEFCAP</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>A value represents the amount of defined capacity specified for the logical partition. A value of 0 indicates that no defined capacity is specified for the logical partition.</td>
</tr>
<tr>
<td>HWI_SGPIPW</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>A value from 1 - 999 defines the relative amount of shared general purpose processor resources allocated to the CPC image object.</td>
</tr>
<tr>
<td></td>
<td>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated general purpose processor.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated general purpose processor.</td>
</tr>
<tr>
<td>HWI_SGPIPWCAP</td>
<td>A 4-byte integer type value. This indicates that the initial general purpose processor processing weight for the CPC Image object is capped or not capped.</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_TRUE</strong></td>
</tr>
<tr>
<td></td>
<td>Capped</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_FALSE</strong></td>
</tr>
<tr>
<td></td>
<td>Not capped</td>
</tr>
<tr>
<td>HWI_SGPPWMIN</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>A value from 1 - 999 and less than or equal to the initial processing weight defines the minimum relative amount of shared general purpose processor resources allocated to the CPC image object.</td>
</tr>
<tr>
<td></td>
<td>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated general purpose processor.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated general purpose processor.</td>
</tr>
<tr>
<td>SetTypes</td>
<td>Values to be specified</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HWI_SGPPWMAX</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>A value from 1 - 999 and greater than or equal to the initial processing weight defines the maximum relative amount of shared general purpose processor resources allocated to the CPC image object.</td>
</tr>
<tr>
<td></td>
<td>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated general purpose processor.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated general purpose processor.</td>
</tr>
<tr>
<td>HWI_WLM</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>This indicates whether the Workload Manager is allowed to change processor weight-related attributes.</td>
</tr>
<tr>
<td></td>
<td>• HWMCA_TRUE</td>
</tr>
<tr>
<td></td>
<td>• HWMCA_FALSE</td>
</tr>
<tr>
<td></td>
<td>HWI_WLM must be set to HWMCA_TRUE before any of the settings for the specialized IFA, IFL, ICF, or IIP engines can be modified.</td>
</tr>
<tr>
<td>HWI_IFAIPW</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>A value from 1 - 999 defines the relative amount of shared integrated facility for applications (IFA) processor resources allocated to the CPC image object.</td>
</tr>
<tr>
<td></td>
<td>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor.</td>
</tr>
<tr>
<td>HWI_IFAIPWCAP</td>
<td>A 4-byte integer type value. This indicates whether the initial processing weight for integrated facility for applications (IFA) processors is a limit or a target.</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_TRUE</strong></td>
</tr>
<tr>
<td></td>
<td>Capped</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_FALSE</strong></td>
</tr>
<tr>
<td></td>
<td>Not capped</td>
</tr>
<tr>
<td>HWI_IFAPWMIN</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>A value from 1 - 999 defines the minimum relative amount of shared integrated facility for applications (IFA) processor resources allocated to the CPC image object.</td>
</tr>
<tr>
<td></td>
<td>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor.</td>
</tr>
<tr>
<td>SetTypes</td>
<td>Values to be specified</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HWI_IFAPWMAX</td>
<td>A 4-byte integer type value. A value from 1 - 999 defines the maximum relative amount of shared integrated facility for applications (IFA) processor resources allocated to the CPC image object. A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor. <strong>Note:</strong> The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor.</td>
</tr>
<tr>
<td>HWI_IFLIPW</td>
<td>A 4-byte integer type value. A value from 1 - 999 defines the relative amount of shared integrated facility for Linux (IFL) processor resources allocated to the CPC image object. A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor. <strong>Note:</strong> The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor.</td>
</tr>
<tr>
<td>HWI_IFLIPWCAP</td>
<td>A 4-byte integer type value. This indicates whether the initial processing weight for integrated facility for Linux (IFL) processors is a limit or a target. <strong>HWMCA_TRUE</strong> Capped <strong>HWMCA_FALSE</strong> Not capped</td>
</tr>
<tr>
<td>HWI_IFLPWMIN</td>
<td>A 4-byte integer type value. A value from 1 - 999 defines the minimum relative amount of shared integrated facility for Linux (IFL) processor resources allocated to the CPC image object. A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor. <strong>Note:</strong> The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor.</td>
</tr>
<tr>
<td>HWI_IFLPWMAX</td>
<td>A 4-byte integer type value. A value from 1 - 999 defines the maximum relative amount of shared integrated facility for Linux (IFL) processor resources allocated to the CPC image object. A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor. <strong>Note:</strong> The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor.</td>
</tr>
<tr>
<td>SetTypes</td>
<td>Values to be specified</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HWI_ICFIPW</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>A value from 1 - 999 defines the relative amount of shared internal coupling facility (ICF) processor resources allocated to the CPC image object.</td>
</tr>
<tr>
<td></td>
<td>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor.</td>
</tr>
<tr>
<td>HWI_ICFIPWCAP</td>
<td>A 4-byte integer type value. This indicates whether the initial processing weight for internal coupling facility (ICF) processors is a limit or a target.</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_TRUE</strong>  Capped</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_FALSE</strong> Not capped</td>
</tr>
<tr>
<td>HWI_ICFPWMIN</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>A value from 1 - 999 defines the minimum relative amount of shared internal coupling facility (ICF) processor resources allocated to the CPC image object.</td>
</tr>
<tr>
<td></td>
<td>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor.</td>
</tr>
<tr>
<td>HWI_ICFPWMAX</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>A value from 1 - 999 defines the maximum relative amount of shared internal coupling facility (ICF) processor resources allocated to the CPC image object.</td>
</tr>
<tr>
<td></td>
<td>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor.</td>
</tr>
<tr>
<td>HWI_IIPPW</td>
<td>A 4-byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>A value from 1 - 999 defines the relative amount of shared integrated information processors (IIP) resources allocated to the CPC image object.</td>
</tr>
<tr>
<td></td>
<td>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated information processor (IIP).</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated information processor (IIP).</td>
</tr>
<tr>
<td>SetTypes</td>
<td>Values to be specified</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>HWI_IIPIPWCAP</td>
<td>A 4-byte integer type value. This indicates whether the initial processing weight for integrated information processors (IIP) is a limit or a target.</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_TRUE</strong></td>
</tr>
<tr>
<td></td>
<td>Capped</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_FALSE</strong></td>
</tr>
<tr>
<td></td>
<td>Not capped</td>
</tr>
<tr>
<td>HWI_IIPPWMIN</td>
<td>A 4-byte integer type value. A value from 1 - 999 defines the minimum relative amount of shared integrated information processors (IIP) resources allocated to the CPC image object. A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated information processor (IIP). <strong>Note:</strong> The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated information processor (IIP).</td>
</tr>
<tr>
<td>HWI_IIPPWMAX</td>
<td>A 4-byte integer type value. A value from 1 - 999 defines the maximum relative amount of shared integrated information processors (IIP) resources allocated to the CPC image object. A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated information processor (IIP). <strong>Note:</strong> The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated information processor (IIP).</td>
</tr>
<tr>
<td>HWI_IOCDS</td>
<td>A character string representing the IOCDS. A value of an empty string indicates that the reset activation profile will use the currently active IOCDS.</td>
</tr>
<tr>
<td>HWI_PROCESS_RUNTIME_TYPE</td>
<td>A 4 byte integer type value. <strong>HWMCA DETERMINED_SYSTEM</strong> The processor running is dynamically determined by the system. <strong>HWMCA DETERMINED_USER</strong> The processor running time is set to a constant value.</td>
</tr>
<tr>
<td>HWI_PROCESS_RUNTIME</td>
<td>A 4 byte integer type value. A value between 1 and 100 for the user defined processor running time. <strong>Note:</strong> This value can only be set if the processor running time type is set to HWMCA_DETERMINED_USER.</td>
</tr>
<tr>
<td>HWI_END_TIME_SLICE_IF_DISABLED_WAIT</td>
<td>A 4 byte integer type value. This indicates whether CPC images lose their share of running time when they enter a wait state. • <strong>HWMCA_TRUE</strong> • <strong>HWMCA_FALSE</strong> <strong>Note:</strong> This value can only be set if the processor running time type is set to HWMCA_DETERMINED_USER.</td>
</tr>
<tr>
<td>SetTypes</td>
<td>Values to be specified</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HWI_IPL_ADDRESS</td>
<td>A character string representing the IPL address.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> A value of an empty string indicates that the image activation profile uses the next IPL address set by HCD.</td>
</tr>
<tr>
<td>HWI_IPL_PARM</td>
<td>A character string representing the IPL parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> A value of an empty string indicates that the image activation profile uses the next IPL parameter set by HCD.</td>
</tr>
<tr>
<td>HWI_INIT_PROCESS_WEIGHT</td>
<td>A 4 byte integer type value from 1 - 999 defines the relative amount of shared processor resources allocated to the CPC image.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated central processor.</td>
</tr>
<tr>
<td>HWI_INIT_PROCESS_WEIGHT_CAPPED _STATUS</td>
<td>A 4 byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>This indicates whether the initial processor weight is a limit or a target.</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_TRUE</strong></td>
</tr>
<tr>
<td></td>
<td>Capped</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_FALSE</strong></td>
</tr>
<tr>
<td></td>
<td>Not capped</td>
</tr>
<tr>
<td>HWI_MAX_PROCESS_WEIGHT</td>
<td>A 4 byte integer type value from 1 - 999 and less than or equal to the initial processing weight defines the maximum relative amount of shared processor resources allocated to the CPC image.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated central processor.</td>
</tr>
<tr>
<td>HWI_DEFINED_CAPACITY</td>
<td>A 4 byte integer type value.</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>No defined capacity is specified for this logical partition.</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><em>nnnn:</em> Represents the amount of defined capacity specified for this logical partition.</td>
</tr>
<tr>
<td>HWI_IPL_TYPE</td>
<td>A 4 byte integer type value.</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_IPLTYPE_STANDARD</strong></td>
</tr>
<tr>
<td></td>
<td>Indicates that the image activation profile is used to perform a standard load.</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_IPLTYPE_SCSI</strong></td>
</tr>
<tr>
<td></td>
<td>Indicates that the image activation profile is used to perform a SCSI load.</td>
</tr>
<tr>
<td></td>
<td><strong>HWMCA_IPLTYPE_SCSIDUMP</strong></td>
</tr>
<tr>
<td></td>
<td>Indicates that the image activation profile is used to perform a SCSI dump.</td>
</tr>
<tr>
<td>HWI_WW_PORTNAME</td>
<td>A character string representing the worldwide port name.</td>
</tr>
<tr>
<td>HWI_BOOT_PGM_SELECTOR</td>
<td>A 4-byte integer type value representing the boot program selector value.</td>
</tr>
<tr>
<td>HWI_LU_NUM</td>
<td>A character string representing the logical unit number.</td>
</tr>
<tr>
<td>HWI_BOOTREC_BLK_ADDR</td>
<td>A character string representing the boot record logical block address.</td>
</tr>
<tr>
<td>HWI_OPSYS_LOADPARM</td>
<td>A character string representing the operating system specific load parameters.</td>
</tr>
<tr>
<td>SetTypes</td>
<td>Values to be specified</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| HWI_IFA_PROCESS_WEIGHT                       | A 4-byte integer type value from 1 - 999 defines the relative amount of shared integrated facility for applications (IFA) processor resources allocated to the CPC image.  
**Note:** The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor. |
| HWI_IFA_PROCESS_WEIGHT_CAPPED_STATUS        | A 4-byte integer type value.  
This indicates whether the initial processing weight for IFA processors (zAAP) is a limit or a target.  
**HWMCA_TRUE**  
Capped  
**HWMCA_FALSE**  
Not capped |
| HWI_IFA_MIN_PROCESS_WEIGHT                   | A 4-byte integer type value from 1 - 999 defines the relative amount of shared integrated facility for applications (IFA) processor resources allocated to the CPC image.  
**Note:** The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor. |
| HWI_IFA_MAX_PROCESS_WEIGHT                   | A 4-byte integer type value from 1 - 999 defines the relative amount of shared integrated facility for applications (IFA) processor resources allocated to the CPC image.  
**Note:** The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor. |
| HWI_IFL_PROCESS_WEIGHT                       | A 4-byte integer value from 1 - 999 defines the relative amount of shared integrated facility for Linux (IFL) processor resources allocated to the CPC image.  
**Note:** The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor. |
| HWI_IFL_PROCESS_WEIGHT_CAPPED_STATUS        | A 4-byte integer type value.  
This indicates whether the initial processing weight for IFL processors (Linux) is a limit or a target.  
**HWMCA_TRUE**  
Capped  
**HWMCA_FALSE**  
Not capped |
| HWI_IFL_MIN_PROCESS_WEIGHT                   | A 4-byte integer type value from 1 - 999 defines the minimum relative amount of shared integrated facility for Linux (IFL) processor resources allocated to the CPC image.  
**Note:** The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor. |
| HWI_IFL_MAX_PROCESS_WEIGHT                   | A 4-byte integer type value from 1 - 999 defines the maximum relative amount of shared integrated facility for Linux (IFL) processor resources allocated to the CPC image.  
**Note:** The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor. |
<table>
<thead>
<tr>
<th>SetTypes</th>
<th>Values to be specified</th>
</tr>
</thead>
</table>
| HWI_ICF_PROCESS_WEIGHT                       | A 4-byte integer type value from 1 - 999 defines the relative amount of shared internal coupling facility (ICF) processor resources allocated to the CPC image.  
**Note:** The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor. |
| HWI_ICF_PROCESS_WEIGHT_CAPPED_STATUS        | A 4 byte integer type value.  
This indicates whether the initial processing weight for ICF processors (coupling facility) is a limit or a target.  
| HWMCA_TRUE                                   | Capped                                                                                                                                                    |
| HWMCA_FALSE                                  | Not capped                                                                                                                                               |
| HWI_ICF_MIN_PROCESS_WEIGHT                   | A 4-byte integer type value from 1 - 999 defines the minimum relative amount of shared internal coupling facility (ICF) processor resources allocated to the CPC image.  
**Note:** The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor. |
| HWI_ICF_MAX_PROCESS_WEIGHT                   | A 4-byte integer type value from 1 - 999 defines the maximum relative amount of shared internal coupling facility (ICF) processor resources allocated to the CPC image.  
**Note:** The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor. |
| HWI_ZIIP_PROCESS_WEIGHT                      | A 4-byte integer type value from 1 - 999 defines the relative amount of shared integrated information processors (zIIP) resources allocated to the CPC image.  
**Note:** The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated integrated information processors (zIIP). |
| HWI_ZIIP_PROCESS_WEIGHT_CAPPED_STATUS       | A 4 byte integer type value.  
This indicates whether the initial processing weight for IIP processors (zIIP) is a limit or a target.  
| HWMCA_TRUE                                   | Capped                                                                                                                                                    |
| HWMCA_FALSE                                  | Not capped                                                                                                                                               |
| HWI_ZIIP_MIN_PROCESS_WEIGHT                  | A 4-byte integer type value from 1 - 999 defines the minimum relative amount of shared integrated information processors (zIIP) resources allocated to the CPC image.  
**Note:** The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated integrated information processors (zIIP). |
| HWI_ZIIP_MAX_PROCESS_WEIGHT                  | A 4-byte integer type value from 1 - 999 defines the maximum relative amount of shared integrated information processors (zIIP) resources allocated to the CPC image.  
**Note:** The setting of this attribute is only valid for CPC images that represent a logical partition with at least one not dedicated integrated information processors (zIIP). |
<p>| HWI_GROUP_PROF_NAME                          | A character string that represents the name of a group capacity profile.   |</p>
<table>
<thead>
<tr>
<th>SetTypes</th>
<th>Values to be specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWI_LOAD_AT_ACTIVATION</td>
<td>A 4-byte integer type value. This indicates whether a load should be done at the end of activation.</td>
</tr>
<tr>
<td>HWI_CENTRAL_STOR</td>
<td>A 4-byte integer type value to represent the initial amount of central storage (in megabytes) to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_RES_CENTRAL_STOR</td>
<td>A 4-byte integer type value to represent the reserved amount of central storage (in megabytes) to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_EXPANDED_STOR</td>
<td>A 4-byte integer type value to represent the initial amount of expanded storage (in megabytes) to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_RES_EXPANDED_STOR</td>
<td>A 4-byte integer type value to represent the reserved amount of expanded storage (in megabytes) to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_GPP</td>
<td>A 4-byte integer type value to represent the number of dedicated general purpose processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_RESGPP</td>
<td>A 4-byte integer type value to represent the number of reserved dedicated general purpose processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_IFA</td>
<td>A 4-byte integer value to represent the number of dedicated integrated facility for applications (IFA) processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_RESIFA</td>
<td>A 4-byte integer value to represent the number of reserved dedicated integrated facility for applications (IFA) processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_IFL</td>
<td>A 4-byte integer value to represent the number of dedicated integrated facility for Linux (IFL) processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_RESIFL</td>
<td>A 4-byte integer value to represent the number of reserved dedicated integrated facility for Linux (IFL) processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_ICF</td>
<td>A 4-byte integer value to represent the number of dedicated internal coupling facility (ICF) processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_RESICF</td>
<td>A 4-byte integer value to represent the number of reserved dedicated internal coupling facility (ICF) processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_ZIIP</td>
<td>A 4-byte integer value to represent the number of dedicated integrated information processors (zIIP) to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_RESZIIP</td>
<td>A 4-byte integer value to represent the number of reserved dedicated integrated information processors (zIIP) to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_SHARED_GPP</td>
<td>A 4-byte integer type value to represent the number of shared general purpose processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_RES_SHARED_GPP</td>
<td>A 4-byte integer type value to represent the number of reserved shared general purpose processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_SHARED_IFA</td>
<td>A 4-byte integer value to represent the number of shared integrated facility for applications (IFA) processors to be used for the CPC image.</td>
</tr>
</tbody>
</table>
SetTypes

<table>
<thead>
<tr>
<th>SetType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWI_NUM_RES_SHARED_IFA</td>
<td>A 4-byte integer value to represent the number of reserved shared integrated facility for applications (IFA) processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_SHARED_IFL</td>
<td>A 4-byte integer value to represent the number of shared integrated facility for Linux (IFL) processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_RES_SHARED_IFL</td>
<td>A 4-byte integer value to represent the number of reserved shared integrated facility for Linux (IFL) processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_SHARED_ICF</td>
<td>A 4-byte integer value to represent the number of shared internal coupling facility (ICF) processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_RES_SHARED_ICF</td>
<td>A 4-byte integer value to represent the number of reserved shared internal coupling facility (ICF) processors to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_SHARED_ZIIP</td>
<td>A 4-byte integer value to represent the number of shared integrated information processors (zIIP) to be used for the CPC image.</td>
</tr>
<tr>
<td>HWI_NUM_RES_SHARED_ZIIP</td>
<td>A 4-byte integer value to represent the number of reserved shared integrated information processors (zIIP) to be used for the CPC image.</td>
</tr>
</tbody>
</table>

SetTypeValueLen
Supplied parameter  
Type: Integer  
Length: 4 bytes  
SetTypeValueLen specifies the length in bytes of the SetTypeValue pointed to by the SetTypeValue_Ptr parameter.

DiagArea
Returned parameter  
Type: Character  
Length: 32 bytes  
DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the DiagArea might not be filled in, and the data returned in the area should be ignored.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag_Index</td>
<td>32-bit integer</td>
<td>The array index to the parameter field that causes the error.</td>
</tr>
<tr>
<td>Diag_Key</td>
<td>32-bit integer</td>
<td>The constant value represents the field that causes the error.</td>
</tr>
<tr>
<td>Diag_Actual</td>
<td>32-bit integer</td>
<td>The incorrect actual value specified.</td>
</tr>
<tr>
<td>Diag_Expected</td>
<td>32-bit integer</td>
<td>The expected value to be used.</td>
</tr>
<tr>
<td>Diag_CommErr</td>
<td>32-bit integer</td>
<td>The returned code that is returned from the Console Application API or the BCPii transport layer.</td>
</tr>
<tr>
<td>Diag_text</td>
<td>Character (12)</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
**ABEND Codes**
The call might result in an abend X'042' with a reason code of X'0008yyyy' due to one of the following reasons:

<table>
<thead>
<tr>
<th>yyyy</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The parameters passed by the caller are not in the primary address space.</td>
</tr>
<tr>
<td>0001</td>
<td>The parameters passed by the caller are not accessible.</td>
</tr>
<tr>
<td>0002</td>
<td>The number of parameters passed by the user is not correct.</td>
</tr>
</tbody>
</table>

**Return Codes**
When the service returns control to the caller, GPR 15 and ReturnCode contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return Code in: Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 HWI_OK</td>
<td>Meaning: Successful completion. Action: None.</td>
</tr>
</tbody>
</table>
| 100 HWI.CONNECT_TOKEN_INV                | Meaning: Program error. The specified connect token is not valid. This return code indicates one of the following conditions has occurred:  
- The connect token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken.  
- The connect token does not represent an active connection. The connection specified might have already been disconnected using the HWIDISC service call.  
- The connect token is not associated with the address space of the caller. The ConnectToken specified is associated with a different address space than the caller of this service call.  
Action: Check for probable coding error. |
| 101 HWI_COMMUNICATION_ERROR              | Meaning: A communication error is detected. The hardware management console application API (HWMCA) or the BCPii Transport layer has returned with a failing return code.  
Action: Check for probable coding error. See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPii Transport layer.  
HWMCA API and BCPii transport return codes are provided in Appendix A, “BCPii Communication Error Reason Codes,” on page A-1. |
<p>| 102 HWI_DIAGAREA_INV                     | Meaning: Program error. The DiagArea is not accessible. Action: Check for probable coding error. Verify the specified DiagArea is defined as a 32-byte character field. |</p>
<table>
<thead>
<tr>
<th>Return Code in: Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 501 HWI_SETTYPE_INV                      | **Meaning:** Program error. The requested SetType specified in the call is not valid for the ConnectToken specified. The system rejects the service call. This return code indicates one of the following conditions has occurred:  
  - The SetType specified is not in the acceptable value range of attributes that can be set.  
  - The specified SetType has been provided with an incompatible connection type. For example, the attribute identifier applies only to CPC connections, but the ConnectToken specified represents an image connection, or any of the activation profile connections.  
  **Action:** Check for probable coding error. Validate that the SetType specified is in the valid range of possible values. Validate that the SetType specified is permitted for the specified connection type.  
  See the DiagArea for further diagnostic information.  
  - The Diag_Key contains the value of the attribute in question.  
  - The Diag_Text contain “Bad Set Attr” if the value of the attribute cannot be set; the Diag_Text contains “Mismatch” if the attribute cannot be set for the specified connection type. |
| 502 HWI_SETTYPE_VALUE_INV                 | **Meaning:** Program error. The requested SetTypeValue to be set or changed is not valid. The system rejects the service call.  
  **Action:** Check for probable coding error. Validate that the value to which an attribute is being set is appropriate for that attribute. |
| 503 HWI_SETTYPE_VALUE_LEN_INV             | **Meaning:** Program error. The SetTypeValueLen specified is not valid. The SetTypeValueLen must be equal to or greater than the minimum required length for the set type value.  
  **Action:** Check for probable coding error. Validate that the SetTypeValueLen specified is equal to or greater than the minimum required length for the set type value. |
| 504 HWI_SETTYPE_VALUE_INV_ACCESSIBLE      | **Meaning:** Program error. The set type value data area is either partially or completely inaccessible by the application, or BCPii, or both.  
  **Action:** Check for probable coding error. Verify the SetTypeValue_Ptr points to a data area where the set type value is, and make sure that the data area is accessible. |
| 506 HWI_SETATTRIBUTE_NOT_SUPPORTED       | **Meaning:** The targeted hardware of the HWISET request does not recognize the attribute that the user is attempting to set.  
  **Action:** Verify that the targeted hardware is at a level that supports the type of attribute being set. |
| F00 HWI_NOTAVAILABLE                     | **Meaning:** HWI is not available, and the system rejects the service request.  
  **Action:** Start HWI and try the request again. |
| F01 HWI_AUTH_FAILURE                     | **Meaning:** The caller is PKM8-15 problem state.  
  **Action:** Check the calling program for a probable coding error. |
<table>
<thead>
<tr>
<th>Return Code in: Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>F02 HWI_NO_SAF_AUTH</td>
<td>Meaning: The user does not have correct SAF authorization for the request.</td>
</tr>
<tr>
<td></td>
<td>Action: Check for probable error. Consider one or more of the following possible actions:</td>
</tr>
<tr>
<td></td>
<td>• Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV.</td>
</tr>
<tr>
<td></td>
<td>• Define update access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for a CPC connection.</td>
</tr>
<tr>
<td></td>
<td>• Define update access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for an image connection.</td>
</tr>
<tr>
<td></td>
<td>• Ensure that the referenced Facility Class Profile is RACLST-specified.</td>
</tr>
<tr>
<td>F03 HWI_INTERRUPT_STATUS_INV</td>
<td>Meaning: The calling program is disabled. The system rejects this service request.</td>
</tr>
<tr>
<td></td>
<td>Action: Check the calling program for a probable coding error.</td>
</tr>
<tr>
<td>F04 HWI_MODE_INV</td>
<td>Meaning: The calling program is not in Task mode, which is the required mode. The system rejects this service request.</td>
</tr>
<tr>
<td></td>
<td>Action: Check the calling program for a probable error.</td>
</tr>
<tr>
<td>F05 HWI_LOCKS_HELD</td>
<td>Meaning: The calling program is holding one or more locks. The system rejects this service request.</td>
</tr>
<tr>
<td></td>
<td>Action: Check the calling program for a probable coding error.</td>
</tr>
<tr>
<td>F06 HWI_UNSUPPORTED_RELEASE</td>
<td>Meaning: The system level does not support this service. The system rejects this service request.</td>
</tr>
<tr>
<td></td>
<td>Action: Remove the calling program from the system, and install it on a system that supports HWI. Then rerun the calling program.</td>
</tr>
<tr>
<td>FFF HWI_UNEXPECTED_ERROR</td>
<td>Meaning: System error. The service that was called encountered an unexpected error. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center. In many cases, a dump has been taken by BCPI to attempt the collection of the necessary information to diagnose the error. If so, provide this dump to the IBM support team.</td>
</tr>
</tbody>
</table>

**Example**

In the pseudocode example, the caller issues a call to change or set the CPC status for a CPC.

```c
SetType = HWI_ACCSTAT;
SetTypeValue = HWMCA_STATUS_OPERATING;
SetTypeValue_Ptr = addr(SetTypeValue);
SetTypeValueLen = Length(SetTypeValue);
```
CALL HWISET (ReturnCode, ConnectToken, SetType, SetTypeValue_Ptr, 
   SetTypeValueLen, DiagArea).

HWIBeginEventDelivery — Begin Delivery of BCPii Event Notifications

Call the HWIBeginEventDelivery service to allow a C application running in the 
UNIX System Services (USS) environment to begin delivery of event notifications. 
This service must be issued before the HWIManageEvents service.

Description

Environment
The requirements for the callers are:

- Minimum authorization: None
- Dispatchable unit mode: Task
- Cross memory mode: PASN=HASN=SASN
- AMODE: 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 C linkage conventions are used

Programming Requirements
The file hwicmuss.x contains the sidedeck needed to link the program to the DLL.

USS C language callers must include the header file HWICIC.

Restrictions
None.

Authorization
Read access to the SAF profile CEA.CONNECT in the SERVAUTH class is 
required.

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

```c
int HWIBeginEventDelivery 
   (  
      *DiagArea 
      ,ConnectToken 
      ,**DeliveryToken 
   )
```

Parameters
The parameters are explained as follows:

- *DiagArea
  Returned parameter
• Type: character string
• Length: 32 bytes

*DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the *DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

**Note:** For all environmental errors (with return code X'F00' and higher), the *DiagArea might not be filled in, and the data returned in the area should be ignored.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag_Index</td>
<td>32-bit integer</td>
<td>The array index to the parameter field that causes the error.</td>
</tr>
<tr>
<td>Diag_Key</td>
<td>32-bit integer</td>
<td>The constant value represents the field that causes the error.</td>
</tr>
<tr>
<td>Diag_Actual</td>
<td>32-bit integer</td>
<td>The incorrect actual value specified.</td>
</tr>
<tr>
<td>Diag_Expected</td>
<td>32-bit integer</td>
<td>The expected value to be used.</td>
</tr>
<tr>
<td>Diag_CommErr</td>
<td>32-bit integer</td>
<td>The returned code from the failing operation.</td>
</tr>
<tr>
<td>Diag_Text</td>
<td>character (12)</td>
<td>Additional diagnostic information in text format.</td>
</tr>
</tbody>
</table>

**ConnectToken**
Supplied parameter
• Type: character string
• Length: 16 bytes

`ConnectToken` specifies the value returned from an HWICONN service call.

****DeliveryToken**
Returned parameter
• Type: character string
• Length: 8 bytes

****DeliveryToken** specifies the variable to contain the address of the token that represents the event notification connection on future service calls.

**ABEND Codes**
None.

**Return Codes**
When the service completes, one of the following values is returned to the caller:

<table>
<thead>
<tr>
<th>Return Code in Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 HWIUSS_RC_OK</td>
<td>Meaning: Successful completion. Action: None.</td>
</tr>
<tr>
<td>00001001 HWIUSS_RC_UNAVAILABLE</td>
<td>Meaning: CEA event notification services are not available. Action: The request is rejected. Start the CEA address space and try the request again.</td>
</tr>
<tr>
<td>00001002 HWIUSS_RC_NO_AUTH</td>
<td>Meaning: The program is not authorized to access CEA services. Action: The request is rejected. Determine if the program needs access to CEA services. If so, grant the required access to the proper resources and try this request again.</td>
</tr>
</tbody>
</table>
## HWIBeginEventDelivery

### Example

In the C code example, the caller issues a call to register for event delivery.

```c
HWI_CONNTOKEN_TYPE hwitoken;
HWI_DIAGAREA_TYPE DiagArea;
HWI_DELIVERYTOKEN_TYPE *DeliveryToken;

int localRC;

localRC = HWIBeginEventDelivery(&DiagArea, hwitoken, DeliveryToken)
```

## HWIEndEventDelivery — End Delivery of BCPii Event Notifications

Call the HWIEndEventDelivery service to allow a C application running in the UNIX System Services (USS) environment to end delivery of event notifications. This service unregisters the registration made by the HWIBeginEventDelivery service.

### Description

#### Environment

The requirements for the callers are:

- **Minimum authorization:** None
- **Dispatchable unit mode:** Task
- **Cross memory mode:** PASN=HASN=SASN
- **AMODE:** 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 C linkage conventions are used

---

### Return Code in Hexadecimal Equate Symbol | Meaning and Action
---

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001003</td>
<td>HWIUSS_RC_MAX_CLIENTS</td>
<td><strong>Meaning:</strong> The maximum number of CEA clients has been reached.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> The request is rejected. Determine if other CEA clients can be stopped. If so, try this request again.</td>
</tr>
<tr>
<td>00001007</td>
<td>HWIUSS_RC_SAF_NOTDEF_CONNECT</td>
<td><strong>Meaning:</strong> The SAF profile CEA.CONNECT is not defined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> The request is rejected. Add the CEA.CONNECT profile to the SERVAUTH class and try this request again.</td>
</tr>
<tr>
<td>00001008</td>
<td>HWIUSS_RC_COMM_FAILURE</td>
<td><strong>Meaning:</strong> An error occurred in UNIX socket processing.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> The request is rejected. Verify that the file system is properly configured for UNIX sockets and try this request again.</td>
</tr>
<tr>
<td>FFFFFFFF</td>
<td>HWIUSS_RC_UNEXPECTED_ERROR</td>
<td><strong>Meaning:</strong> An unexpected error has occurred.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> The request is rejected. Consult the diagarea for more specifics regarding the error. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</td>
</tr>
</tbody>
</table>
Programming Requirements
The file hwicmuss.x contains the sidedeck needed to link the program to the DLL.

USS C language callers must include the header file HWICIC.

Restrictions
None.

Authorization
Read access to the SAF profile CEA.CONNECT in the SERVAUTH class is required.

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

```
int HWIEndEventDelivery
   (*DiagArea
    , *DeliveryToken
   )
```

Parameters
The parameters are explained as follows:

*DiagArea
- Returned parameter
- Type: character string
- Length: 32 bytes

- *DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the *DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the *DiagArea might not be filled in, and the data returned in the area should be ignored.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag_Index</td>
<td>32-bit integer</td>
<td>The array index to the parameter field that causes the error.</td>
</tr>
<tr>
<td>Diag_Key</td>
<td>32-bit integer</td>
<td>The constant value represents the field that causes the error.</td>
</tr>
<tr>
<td>Diag_Actual</td>
<td>32-bit integer</td>
<td>The incorrect actual value specified.</td>
</tr>
<tr>
<td>Diag_Expected</td>
<td>32-bit integer</td>
<td>The expected value to be used.</td>
</tr>
<tr>
<td>Diag_CommErr</td>
<td>32-bit integer</td>
<td>The returned code from the failing operation.</td>
</tr>
<tr>
<td>Diag_Text</td>
<td>character (12)</td>
<td>Additional diagnostic information in text format.</td>
</tr>
</tbody>
</table>

*DeliveryToken
- Supplied parameter
- Type: character string
- Length: 8 bytes

DeliveryToken specifies the event notification connection created by a previous HWIBeginEventDelivery call.
ABEND Codes
None.

Return Codes
When the service completes, one of the following values is returned to the caller:

<table>
<thead>
<tr>
<th>Return Code in Hexadecimal</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>HWIUSS_RC_OK</td>
<td>Meaning: Successful completion. Action: None.</td>
</tr>
<tr>
<td>00001001</td>
<td></td>
<td>Meaning: CEA event notification services are not available. Action: The request is rejected. Start the CEA address space and try the request again.</td>
</tr>
<tr>
<td>00001004</td>
<td></td>
<td>Meaning: The provided delivery token is not valid. Action: The request is rejected. This is a probable coding error.</td>
</tr>
<tr>
<td>00000000</td>
<td></td>
<td>Meaning: An unexpected error has occurred. Action: The request is rejected. Consult the diagarea for more specifics regarding the error. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</td>
</tr>
</tbody>
</table>

Example
In the C code example, the caller issues a call to unregister for event delivery.

```c
HWI_DIAGAREA_TYPE DiagArea;
HWI_DELIVERYTOKEN_TYPE *DeliveryToken;
int localRC;

localRC = HWIEndEventDelivery(&DiagArea, DeliveryToken)
```

HWIManageEvents — Manage the List of BCPii Events

Call the HWIManageEvents service to allow a C application running in the UNIX System Services (USS) environment to manage the list of events for which the application is to be notified. The HWIBeginEventDelivery service must have been called before the HWIManageEvents service being called because the appropriate delivery token returned from the HWIBeginEventDelivery service is required as input.

Description

Environment
The requirements for the callers are:

- **Minimum authorization:** One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
- **Dispatchable unit mode:** Task
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 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 C linkage conventions are used

Programming Requirements
The file hwicmuss.x contains the sidedeck needed to link the program to the DLL.

USS C language callers must include the header file HWICIC.

Restrictions
None.

Authorization
The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

Read access is required to the profile CEA.SUBSCRIBE.ENF_0068qqqqqqqq in the SERVAUTH class, where qqqqqqqqq is the specific hexadecimal event qualifier pattern. See the ENF 68 documentation contained in the ENFREQ chapter of [z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG](#) for further information about how to specify this event qualifier.

The client application must have at least read access to the SAF-protected FACILITY class resource HWI.TARGET.netid.nau for a ConnectToken representing a CPC connection, or HWI.TARGET.netid.nau.imagename for a ConnectToken representing an image connection.

Note: BCPI require the FACILITY class to be RACLIST-specified.

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

```
int HWIManageEventDelivery
   (  
      *DiagArea
      ,*DeliveryToken
      ,ConnectToken
      ,EventAction
      ,EventIDs
   )
```

Parameters
The parameters are explained as follows:

* *DiagArea*
  
  Returned parameter
  
  • Type: character string
  
  • Length: 32 bytes

  * *DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the *DiagArea can contain further
information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

**Note:** For all environmental errors (with return code X'F00' and higher), the *DiagArea might not be filled in, and the data returned in the area should be ignored.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag_Index</td>
<td>32-bit integer</td>
<td>The array index to the parameter field that causes the error.</td>
</tr>
<tr>
<td>Diag_Key</td>
<td>32-bit integer</td>
<td>The constant value represents the field that causes the error.</td>
</tr>
<tr>
<td>Diag_Actual</td>
<td>32-bit integer</td>
<td>The incorrect actual value specified.</td>
</tr>
<tr>
<td>Diag_Expected</td>
<td>32-bit integer</td>
<td>The expected value to be used.</td>
</tr>
<tr>
<td>Diag_CommErr</td>
<td>32-bit integer</td>
<td>The returned code from the failing operation.</td>
</tr>
<tr>
<td>Diag_Text</td>
<td>character (12)</td>
<td>Additional diagnostic information in text format.</td>
</tr>
</tbody>
</table>

**DeliveryToken**
Supplied parameter
- Type: character string
- Length: 8 bytes

*DeliveryToken specifies the event notification connection, as returned by a previous HWIBeginEventDelivery call.

**ConnectToken**
Supplied parameter
- Type: character string
- Length: 16 bytes

ConnectToken specifies a logical connection between the application and a CPC or an image. The ConnectToken is an output parameter on the HWICONN service call.

The ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call.

**EventAction**
Supplied parameter
- Type: integer
- Length: 4 bytes

EventAction specifies the type of action for the service. See the EventAction parameter of "HWIEVENT — Register for BCPII Events" on page 14-32 for the exact syntax.

**EventIDs**
Supplied parameter
- Type: integer
- Length: 128 bit (16 bytes)

EventIDs specifies the events to be added or deleted. See the EventIDs parameter of "HWIEVENT — Register for BCPII Events" on page 14-32 for the exact syntax.

**ABEND Codes**
The call might result in an abend X'042' with a reason code of X'0004yyyy' due to one of the following reasons:
<table>
<thead>
<tr>
<th>YYVV</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>The parameters passed by the caller are not in the primary address space.</td>
</tr>
<tr>
<td>0001</td>
<td>The parameters passed by the caller are not accessible.</td>
</tr>
<tr>
<td>0002</td>
<td>The number of parameters passed by the caller is not correct.</td>
</tr>
</tbody>
</table>

**Return Codes**
When the service completes, one of the following values is returned to the caller:

<table>
<thead>
<tr>
<th>Return Code in Hexadecimal</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00000000                    | HWIUSS_RC_OK  | Meaning: Successful completion.  
Action: None. |
| 00001000                    | HWIUSS_RC_HWIEVENT_FAILURE | Meaning: The resultant HWIEVENT service call failed.  
Action: The request is rejected. The DiagArea contains the failure data. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
| 00001001                    | HWIUSS_RC_UNAVAILABLE | Meaning: CEA event notification services are not available.  
Action: The request is rejected. Start the CEA address space and try the request again. |
| 00001004                    | HWIUSS_RC_BAD_DELIVERYTOKEN | Meaning: The provided delivery token is not valid.  
Action: The request is rejected. This is a probable coding error. |
| 00001006                    | HWIUSS_RC_SAF_NOTDEF_EVENT | Meaning: The SAF profile CEA.SUBSCRIBE.ENF_0068* is not defined.  
Action: The request is rejected. Add the proper CEA.SUBSCRIBE.ENF_0068* profile to the SERVAUTH class and try this request again. |
| FFFFFFFF                    | HWIUSS_RC_UNEXPECTED_ERROR | Meaning: An unexpected error has occurred.  
Action: The request is rejected. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. |

**Example**
In the C code example, the caller issues a call to register to be notified when the command response events and status change events occur.

```c
HWI_DIAGAREA_TYPE  DiagArea;
HWI_DELIVERYTOKEN_TYPE  *DeliveryToken;
HWI_CONNTOKEN_TYPE  ConnectToken;
HWI_EVENTIDS_TYPE  EventIDs;
int localRC;

memset ((void*)&eventIDs, 0x00, sizeof (eventIDs));
memcpy (eventIDs.Hwi_EventID_EyeCatcher
         ,HWI_EVENTID_TEXT
         ,sizeof (eventIDs.Hwi_EventID_EyeCatcher));
EventIDs.Hwi_Event_CmdResp = 1;
EventIDs.Hwi_Event_StatusChg = 1;
localRC = HWIManageEvents(&DiagArea, DeliveryToken, ConnectToken,
                           HWI_EVENT_ADD, EventIDs)
```

Chapter 14. Base Control Program Internal Interface (BCPi)  14-97
HWIGetEvent

HWIGetEvent — Retrieve Outstanding BCPii Event Notifications

Call the HWIGetEvent service to allow a C application running in the UNIX System Services (USS) environment to retrieve outstanding BCPii event notifications.

Description

Environment
The requirements for the callers are:

Minimum authorization: None
Dispatchable unit mode: Task
Cross memory mode: PASN=HASN=SASN
AMODE: 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 C linkage conventions are used

Programming Requirements
The file hwicmuss.x contains the sidedeck needed to link the program to the DLL.

USS C language callers must include the header file HWICIC.

Restrictions
None.

Authorization
None.

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

```
int HWIGetEvent (  
    *DiagArea,  
    *DeliveryToken,  
    *Buffer,  
    BufferSize,  
    Timeout,  
    *BytesNeeded
)
```

Parameters
The parameters are explained as follows:

*DiagArea
  Returned parameter
  - Type: character string
*DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the *DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

**Note:** For all environmental errors (with return code X'F00' and higher), the *DiagArea might not be filled in, and the data returned in the area should be ignored.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diag_Index</td>
<td>32-bit integer</td>
<td>The array index to the parameter field that causes the error.</td>
</tr>
<tr>
<td>Diag_Key</td>
<td>32-bit integer</td>
<td>The constant value represents the field that causes the error.</td>
</tr>
<tr>
<td>Diag_Actual</td>
<td>32-bit integer</td>
<td>The incorrect actual value specified.</td>
</tr>
<tr>
<td>Diag_Expected</td>
<td>32-bit integer</td>
<td>The expected value to be used.</td>
</tr>
<tr>
<td>Diag_CommErr</td>
<td>32-bit integer</td>
<td>The returned code from the failing operation.</td>
</tr>
<tr>
<td>Diag_Text</td>
<td>character (12)</td>
<td>Additional diagnostic information in text format.</td>
</tr>
</tbody>
</table>

**DeliveryToken**

Supplied parameter

- Type: character string
- Length: 8 bytes

*DeliveryToken specifies the event notification connection, as returned by a previous HWIBeginEventDelivery call.

**Buffer**

Supplied parameter

- Type: character string
- Length: up to 4096 bytes

*Buffer specifies the address of the storage where the ENF68 event data is to be returned. This data is mapped by the HWIENF68 structure in the HWICIC header file.

**BufferSize**

Supplied parameter

- Type: integer
- Length: 4 bytes

BufferSize specifies the size of the *Buffer storage area.

Constant HWIUSS_MAX_GETBUFFER_SIZE can be used to allocate a buffer large enough to hold the maximum size of ENF68 data returned.

**Timeout**

Supplied parameter

- Type: integer
- Length: 4 bytes

Timeout specifies the amount of time, in seconds, for which the service should wait for an event to occur.

<table>
<thead>
<tr>
<th>Constant in Hexadecimal</th>
<th>Equate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>HWIUSS_TIMEOUT_NOWAIT</td>
<td>Do not wait for an event to occur if one is not ready for delivery.</td>
</tr>
<tr>
<td>Constant in Hexadecimal Equate Symbol</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF HWIUSS_TIMEOUT_INFINITE</td>
<td>Do not return until an event has occurred.</td>
<td></td>
</tr>
<tr>
<td>Any other non-negative number</td>
<td>Wait for the specified number of seconds.</td>
<td></td>
</tr>
</tbody>
</table>

*BytesNeeded*
- Returned parameter
  - Type: integer
  - Length: 4 bytes

*BytesNeeded* specifies the variable to contain the number of bytes used in the output buffer to contain the returned event data. If the buffer is not large enough to contain all the event data, this variable contains the amount of storage required to receive all the event data.

**ABEND Codes**
None.

**Return Codes**
When the service completes, one of the following values is returned to the caller:

<table>
<thead>
<tr>
<th>Return Code in Hexadecimal Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000 HWIUSS_RC_OK</td>
<td>Meaning: Successful completion.</td>
</tr>
<tr>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>000000001 HWIUSS_RC_PARTIAL_DATA</td>
<td>Meaning: The provided buffer was not large enough to contain all the event data.</td>
</tr>
<tr>
<td></td>
<td>Action: The request is successful. To receive all the event data, buffer the size of which is at least BytesNeeded must be provided.</td>
</tr>
<tr>
<td>000000002 HWIUSS_RC_EVENTS_LOST</td>
<td>Meaning: At least one event was not returned because the program has not been retrieving events timely.</td>
</tr>
<tr>
<td></td>
<td>Action: The request is successful. To receive all events, the program must make this service call more often or reduce the number of events requested.</td>
</tr>
<tr>
<td>000000003 HWIUSS_RC_TIMEOUT</td>
<td>Meaning: No events have occurred in the requested time interval.</td>
</tr>
<tr>
<td></td>
<td>Action: The request is successful.</td>
</tr>
<tr>
<td>000010011 HWIUSS_RC_UNAVAILABLE</td>
<td>Meaning: CEA event notification services are not available. Either CEA notification services are no longer available, or the connection that HwiGetEvent is using has been ended abnormally by an HwiEndEventDelivery service call issued from another thread in the same address space.</td>
</tr>
<tr>
<td></td>
<td>Action: The request is rejected. Start the CEA address space and try the request again.</td>
</tr>
<tr>
<td>00001004 HWIUSS_RC_BAD_DELIVERYTOKEN</td>
<td>Meaning: The provided delivery token is not valid.</td>
</tr>
<tr>
<td></td>
<td>Action: The request is rejected. This is a probable coding error.</td>
</tr>
<tr>
<td>00001005 HWIUSS_RC_SMALL_BUFFER</td>
<td>Meaning: The provided buffer is not large enough to contain the event data.</td>
</tr>
<tr>
<td></td>
<td>Action: The request is rejected. This is a probable coding error. Provide a larger buffer and try the request again.</td>
</tr>
</tbody>
</table>
Return Code in Hexadecimal Equate Symbol | Meaning and Action
---|---
0FFFFFFF | HWIUSS_RC_UNEXPECTED_ERROR

**Meaning:** An unexpected error has occurred.

**Action:** The request is rejected. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.

### Example

In the C code example, the caller issues a call to retrieve any outstanding event data, waiting forever until an event occurs.

```c
HWI_DIAGAREA_TYPE DiagArea;
HWI_DELIVERYTOKEN_TYPE DeliveryToken;
char *Buffer[HWIUSS_MAX_GETBUFFER_SIZE];
int BufSize = HWIUSS_MAX_GETBUFFER_SIZE;
int Timeout = HWIUSS_TIMEOUT_INFINITE;
int BytesReturned;
int localRC;

localRC = HWIGetEvent(&DiagArea, DeliveryToken, &Buffer, BufSize, Timeout, &BytesReturned)
```
HWIGetEvent
Appendix A. BCPii Communication Error Reason Codes

All BCPii API invocations can experience a communication failure when communicating between the BCPii address space and the support element of the targeted Central Processor Complex (CPC). The calling program receives the HWI_COMMUNICATION_ERROR (101 hexadecimal, 257 decimal) return code when this occurs. One of the output parameters from each service is a Diagnostic Area (referred to as the DiagArea). For the HWI_COMMUNICATION_ERROR return code, a field in this DiagArea that is called Diag_Commerr contains a more descriptive return code from the BCPii communications transport to help pinpoint the cause of the failure.

Below is a partial list of the descriptive communication transport error return codes, along with a suggested action to take.

<table>
<thead>
<tr>
<th>Return Code in Hexadecimal (in decimal)</th>
<th>Description / Suggested Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-63 (0-99)</td>
<td>These return codes are documented in Appendix C (API Return Codes) in the System z Application Programming Interfaces publication (SB10-7030).</td>
</tr>
<tr>
<td>64-76 (100-118)</td>
<td>An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>77 (119)</td>
<td>The BCPii transport rejected the particular request. Activate CTRACE with CTRACE option &quot;ALL&quot; and reissue the request. If the request failed again, turn off CTRACE, collect the SVCDUMP, and contact the IBM Support Center.</td>
</tr>
<tr>
<td>78-CF (120-207)</td>
<td>An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>D0 (208)</td>
<td>The support element rejected the particular request. Activate CTRACE with CTRACE option &quot;ALL&quot; and reissue the request. If the request failed again, turn off CTRACE, collect the SVCDUMP, and contact the IBM Support Center.</td>
</tr>
<tr>
<td>D1-D3 (209-211)</td>
<td>An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.</td>
</tr>
<tr>
<td>D4 (212)</td>
<td>The support element rejected communication from BCPii, likely because the Cross partition authority was not granted on this support element.</td>
</tr>
<tr>
<td>E0 (224)</td>
<td>No response was received from the support element, after waiting for a considerable amount of time. BCPii times out the request. Check if connectivity to the support element is still there.</td>
</tr>
<tr>
<td>Greater than E0 (&gt;224)</td>
<td>An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.</td>
</tr>
</tbody>
</table>
BCPii Communication Error Reason Codes
Appendix B. 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).

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLSCADPL</td>
<td>Describes same data areas as assembler macro BLSABDPL. Depends on BLSCDESC.</td>
</tr>
<tr>
<td>BLSCADSY</td>
<td>Describes same data areas as assembler macro BLSADSY.</td>
</tr>
<tr>
<td>BLSCCBSP</td>
<td>Describes same data areas as assembler macro BLSACBSP. Depends on BLSCDESC.</td>
</tr>
<tr>
<td>BLSCDESC</td>
<td>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.</td>
</tr>
<tr>
<td>BLSCDRPX</td>
<td>Describes same data areas as assembler macro BLSRDRPX. Depends on BLSCDESC.</td>
</tr>
<tr>
<td>BLSCNAMP</td>
<td>Describes same data areas as assembler macro BLSRNAMP. Depends on BLSCDESC.</td>
</tr>
<tr>
<td>BLSCPCQE</td>
<td>Describes same data areas as assembler macro BLSRPCQE. Depends on BLSCDESC.</td>
</tr>
<tr>
<td>BLSCPPR2</td>
<td>Describes same data areas as assembler macro BLSUPPR2.</td>
</tr>
<tr>
<td>BLSCPWHS</td>
<td>Describes same data areas as assembler macro BLSRPWHS. Depends on BLSCDESC.</td>
</tr>
<tr>
<td>BLSCXmsp</td>
<td>Describes same data areas as assembler macro BLSRXMSP. Depends on BLSCDESC.</td>
</tr>
<tr>
<td>BLSCXSSP</td>
<td>Describes same data areas as assembler macro BLSRXSSP. Depends on BLSCDESC.</td>
</tr>
</tbody>
</table>

End of Programming Interface information
C/C++ header files
Appendix C. 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 such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to z/OS TSO/E Primer, z/OS TSO/E User's Guide, and z/OS ISPF User's Guide Vol I for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at:

http://www.ibm.com/systems/z/os/zos/bkserv/
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.

**Permission Notice**

This book includes information about certain callable service stub and linkage-assist (stub) routines contained in specific data sets that are intended to be bound or link-edited with code and run on z/OS systems. In connection with your authorized use of z/OS, you may bind or link-edit these stubs into your modules and distribute your modules with the included stubs for the purposes of developing, using, marketing and distributing programs conforming to the documented programming interfaces for z/OS, provided that (i) each stub is included in its entirety, including any IBM copyright statements. These stubs have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply the reliability, serviceability, or function of these stub programs. The stub referred to in this book is contained in the following data set:

- SYS1.CSSLIB
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.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS™, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Windows and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.
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

A
access to a data object
permanent object 1-6
temporary object 1-7
access to an object
terminating 2-12
accessibility C-1
ADA programming language
example using window services 4-1
application
in resource recovery 10-1
application_backout_UR call 10-5
return and reason codes 10-8
syntax 10-7
application_commit_UR call 10-9
return and reason codes 10-11
syntax 10-11

B
back out changes to protected resources 10-5
blocks of an object
definition 1-1
identifying blocks to be viewed 2-6
size 1-1
updating blocks 2-10
updating blocks in a temporary object 2-8

C
C programming language
call syntax for latch manager services 9-1
defined to reference pattern services 8-1
defined to using window services 4-6
call statements for latch manager services 9-1
call statements for reference pattern services 7-1, 8-1
call statements for window services 3-1, 4-1
call syntax
for latch manager service 9-1
changed data in an object
refreshing 2-9
COBOL programming language
call syntax for latch manager services 9-1
defined to reference pattern services 8-4
defined to using window services 4-9
commit changes to protected resources 10-9
commit protocol, two-phase 10-2
CSREVW callable service
defining a view of an object 2-4
parameter descriptions 3-2, 3-17
purpose 3-1, 3-16
return codes and reason codes 3-3, 3-19
syntax 3-2, 3-17
CSRIDAC callable service
obtaining access to a data object 2-2
parameter descriptions 3-6
purpose 3-5
CSRIDAC callable service (continued)
return codes and reason codes 3-8
syntax 3-6
terminating access to an object 2-12
CSRIIP callable service
definition 6-5
purpose 7-1
CSRIPP callable service
purpose 7-3
CSRL16J callable service
entry characteristics for the target routine 12-1
freeing dynamic storage for the target routine 12-2
parameter definition 12-1
programming requirements 12-2
return codes 12-6
syntax 12-1
CSRREFR callable service
parameter descriptions 3-10
purpose 3-9
refreshing changed data 2-9
return codes and reason codes 3-10
syntax 3-10
CSRSAVE callable service
parameter descriptions 3-12
purpose 3-11
return codes and reason codes 3-13
syntax 3-12
updating a permanent object on DASD 2-10
CSRSCH callable service
parameter descriptions 3-15
purpose 3-14
return codes and reason codes 3-15
saving interim changes in a scroll area 2-8
syntax 3-14
updating a temporary object 2-8
CSRSIC include file 13-4
CSRVIEW callable service
defining a view of an object 2-4
terminating a view of an object 2-10

D
DASD (direct access storage device)
data transfer from by window services 1-3
updating a permanent object 2-10
data object
define multiple views 2-7
defining a view 2-4
extending the size 2-7
identifying 2-2
mapping 1-1
DASD to a scroll area 1-3
DASD to a window 1-3
multiple objects 1-5
object to multiple windows 1-4
scroll area to a window 1-3
temporary object 1-4
obtaining 2-1
data object *(continued)*
- access to a data object 2-1
- scroll area 2-3
- obtaining access 2-2
- refreshing changed data 2-9
- saving an interim change 2-8
- specifying type of access 2-3
- structure 1-1
- terminating access to an object 2-12
- updating a temporary object 2-8
- data to be viewed
  - identifying 2-6
- data-in-virtual object 1-1
- DFP requirement for window services 2-3
- direct access storage device 1-3
- disability C-1

E
examples
- data object mapped to a window 1-2
- mapping 1-3
  - multiple objects 1-5
  - object to multiple windows 1-4
  - permanent object that has a scroll area 1-3
  - permanent object that has no scroll area 1-3
  - temporary object 1-4
- structure of a data object 1-2

F
FORTRAN programming language
- call syntax for latch manager services 9-1
- example using reference pattern services 8-8
- example using window services 4-13

G
gap in reference pattern services
- defining 5-4
- definition 5-4
- glossary of terms E-1

H
hiperspace
- window services use 1-3

I
identifying data object 2-2
- IEAAFFN callable service
  - parameter descriptions 11-1
  - purpose 11-1
  - requirements 11-2
  - restrictions and limitations 11-2
  - return codes 11-2
  - syntax 11-1
- interim changes to a permanent object
  - saving 2-8

ISGLCRT callable service
- purpose 9-2
- syntax 9-2

ISGLOBT callable service
- purpose 9-6
- syntax 9-6

ISGLPBA callable service
- purpose 9-13
- syntax 9-13

ISGLPRG callable service
- purpose 9-12
- syntax 9-12

ISGLREL callable service
- purpose 9-9
- syntax 9-9

K
keyboard C-1

L
latch manager services
- ISGLCRT callable service
  - syntax 9-2
- ISGLOBT callable service
  - syntax 9-6
- ISGLPBA callable service
  - syntax 9-13
- ISGLPRG callable service
  - syntax 9-12
- ISGLREL callable service
  - syntax 9-9

M
mainframe
- education xiii
- multiple views of an object
  - defining 2-7

N
Notices D-1

O
object
- data 2-1
  - permanent 2-1
  - temporary 2-1

P
Pascal programming language
- example using window services 4-17, 8-11
- permanent object
  - accessing an existing object 2-3
  - creating a new object 2-3
  - data transfer from 1-3
permanent object (continued)  
define multiple views 2-7  
defining a view 2-4  
definition 1-1  
extending the size 2-7  
functions supported 1-6  
identifying 2-2  
mapping a scroll area to a permanent object 1-3  
mapping with no scroll area 1-3  
maximum size 1-1  
new object, creating 2-3  
obtaining 2-1  
  access to a permanent object 2-1  
  access to a permanent object, procedure 2-2  
  scroll area 2-3  
overview of supported functions 1-6  
refreshing 2-1  
refreshing changed data 2-9  
relationship to a data-in-virtual object 1-1  
requirements for new objects 2-3  
saving an interim change 2-8  
saving changes in 2-1  
specifying new or old status 2-3  
specifying type of access for an existing object 2-3  
structure 1-1  
terminating access to a permanent object 2-12  
updating 2-10  
PL/I programming language  
call syntax for latch manager services 9-1  
example using window services 4-21  
processor affinity 11-1  
protected  
resource 10-1

R

reference pattern services 7-1  
coding examples 8-1  
  C programming language 8-1  
  COBOL programming language 8-4  
  FORTRAN programming language 8-8  
  Pascal programming language 8-11  
CSRIRP callable service 7-1  
syntax 7-1  
CSRRRP callable service 7-3  
syntax 7-3  
overview 5-1  
use with data window services 2-5  
  using 6-1  
reference services  
call statements 7-1, 9-1  
  reference information 7-1, 9-1  
reference unit in reference pattern services  
  choosing 5-3  
  definition 5-3, 5-4  
REPLACE option for a window 2-5  
requirements for window services  
  DFP requirement 2-3  
  SMS requirement 2-3  
resource  
  process for protecting 10-2

S

scroll area  
data transfer from 1-3  
definition 1-2  
mapping a scroll area to DASD 1-3  
mapping a window to a scroll area 1-3  
obtaining a scroll area 2-3  
refreshing a scroll area 2-9  
saving changes in 2-1  
saving interim changes 2-8  
storage used 1-2  
updating a permanent object from a scroll area 2-10  
updating DASD from 2-1  
use 1-2  
shortcut keys C-1

T

temporary object  
accessing a temporary object 2-2  
creating a temporary object 2-2  
data transfer from 1-3  
define multiple views 2-7  
defining a view 2-4  
definition 1-1  
extending the size 2-7  
functions supported 1-7  
initialized value 1-2  
requesting protection 10-4  
resource manager  
in resource recovery 10-1  
resource recovery  
distributed 10-5  
process 10-2  
programs 10-1  
requesting 10-4  
service 10-5, 10-9  
RETAI option for a window 2-5  
REXX programming language  
call syntax for latch manager services 9-1  
RRS  
  application_backout.UR call 10-5  
  application_commit.UR call 10-9  
  as sync-point manager 10-1
temporary object (continued)
  obtaining (continued)
    scroll area  2-4
  overview of supported functions  1-7
  refreshing  2-1
  refreshing changed data  2-9
  saving changes in  2-1
  specifying the object size  2-3
  storage used  1-2
  structure  1-1
  terminating access to a temporary object  2-12
  updating a temporary object  2-8

terminology E-1

transferring control to another routine
  CSRL16J  12-1

two-phase commit protocol  10-2

window services
  call statements  3-1
  COBOL programming language  4-9
  coding examples  4-1, 4-9
    ADA programming language  4-1
    C programming language  4-6
    FORTRAN programming language  4-13
    Pascal programming language  4-17
    PL/I programming language  4-21
  functions provided  1-2
  handling abends  2-12
  handling return codes  2-12
  overview  1-1
  reference information  3-1
  services provided  1-2
  using window services  2-1
  ways to map an object  1-3

U

UR (unit of recovery)
  backing out  10-5
  committing  10-9
  using protected resources  10-1

V

view of an object
  terminating  2-10

W

ways that window services can map an object  1-3
what window services provides  1-2
window
  affect of terminating access to an object  2-12
  blocks to be viewed, identifying  2-6
  changing a view in a window  2-10
  changing the view  2-2
  data to be viewed, identifying  2-6
  defining  2-1
    multiple windows  2-7
    window  2-1
    window disposition  2-5
    window reference pattern  2-5
    windows with overlapping views  2-7
  definition  1-1
  identifying a window  2-4
  identifying blocks to be viewed  2-6
  mapping  1-3
    multiple objects  1-5
    to a window  1-3
    to multiple windows  1-4
  refreshing a window  2-9
  REPLACE option  2-5
  RETAIN option  2-5
  size  2-4
  storage  2-4
  terminating a view in a window  2-10
  updating a permanent object from a window  2-10
  use  1-1
Readers’ Comments — We’d Like to Hear from You

z/OS
MVS Programming: Callable Services for High-Level Languages

Publication No. SA22-7613-05

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.
Submit your comments using one of these channels:
• Send your comments to the address on the reverse side of this form.
• Send your comments via e-mail to: mhvrdfs@us.ibm.com

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address
Readers’ Comments — We’d Like to Hear from You

SA22-7613-05

IBM Corporation
MHVRCS, Mail Station P181
2455 South Road
Poughkeepsie, NY
12601-5400

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES