

z/OS
3.2

*MVS Programming: Callable Services for
High-Level Languages*



Note

Before using this information and the product it supports, read the information in [“Notices” on page 845](#).

This edition applies to IBM® z/OS® 3.2 (5655-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2025-09-30

© **Copyright International Business Machines Corporation 1994, 2025.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures.....	xi
Tables.....	xiii
About this information.....	xxi
Who should use this information.....	xxi
How to use this information.....	xxi
z/OS information.....	xxi
How to provide feedback to IBM.....	xxiii
Summary of changes.....	xxv
Summary of changes for z/OS 3.2.....	xxv
Summary of changes for z/OS 3.1.....	xxv
Part 1. Window services.....	1
Chapter 1. Introduction to window services.....	3
Permanent data objects.....	3
Temporary data objects.....	3
Structure of a data object.....	3
What does window services provide?.....	4
The ways that window services can map an object.....	4
Access to permanent data objects.....	7
Access to temporary data objects.....	8
Chapter 2. Using window services.....	9
Obtaining access to a data object.....	10
Identifying the object.....	10
Specifying the object's size.....	11
Specifying the type of access.....	11
Obtaining a scroll area.....	11
Defining a view of a data object.....	11
Identifying the data object.....	12
Identifying a window.....	12
Defining the disposition of a window's contents.....	12
Defining the expected reference pattern.....	13
Identifying the blocks you want to view.....	13
Extending the size of a data object.....	14
Defining multiple views of an object.....	14
Non-overlapping views.....	14
Overlapping views.....	14
Saving interim changes to a permanent data object.....	15
Updating a temporary data object.....	15
Refreshing changed data.....	16
Updating a permanent object on DASD.....	16
When there is a scroll area.....	16
When there is no scroll area.....	17
Changing a view in a window.....	17

Terminating access to a data object.....	18
Handling return codes and abnormal terminations.....	18
Chapter 3. Window services.....	19
CSREVIEW — View an object and sequentially access it.....	19
Abend codes.....	21
Return codes and reason codes.....	21
CSRIDAC — Request or terminate access to a data object.....	22
Abend codes.....	25
Return codes and reason codes.....	25
CSRREFR — Refresh an object.....	26
Abend codes.....	27
Return codes and reason codes.....	27
CSRSAVE — Save changes made to a permanent object.....	28
Abend codes.....	29
Return codes and reason codes.....	29
CSRSCOT — Save object changes in a scroll area.....	30
Abend codes.....	31
Return codes and reason codes.....	31
CSRVIEW — View an object.....	32
Abend codes.....	35
Return codes and reason codes.....	35
Chapter 4. Window services coding examples.....	37
ADA example.....	37
C/370 example.....	41
COBOL example.....	43
FORTRAN example.....	46
Pascal example.....	50
PL/I example.....	53
Part 2. Reference pattern services.....	59
Chapter 5. Introduction to reference pattern services.....	61
How does the system manage data?.....	61
An example of how the system manages data in an array.....	62
What pages does the system bring in when a gap exists?.....	63
Chapter 6. Using reference pattern services.....	65
Defining the reference pattern for a data area.....	65
Defining the range of the area.....	65
Identifying the direction of the reference.....	65
Defining the reference pattern.....	66
Choosing the number of bytes on a page fault.....	67
Examples of using CSRIRP to define a reference pattern.....	69
Removing the definition of the reference pattern.....	70
Handling return codes.....	70
Chapter 7. Reference pattern services.....	71
CSRIRP — Define a reference pattern.....	71
Return codes and reason codes.....	73
CSRRRP — Remove a reference pattern.....	73
Return codes and reason codes.....	74
Chapter 8. Reference pattern services coding examples.....	75
C/370 example.....	75
COBOL example.....	77

FORTTRAN example.....	81
Pascal example.....	83
PL/I example.....	85
Part 3. Global resource serialization latch manager services.....	89
Chapter 9. Using the latch manager services.....	91
Syntax and linkage conventions for latch manager callable services.....	91
ISGLCRT — Create a latch set.....	91
ABEND codes.....	93
Return codes.....	93
Examples of calls to latch manager services.....	94
ISGLOBT — Obtain a latch.....	95
ABEND codes.....	97
Return codes.....	97
Example.....	98
ISGLREL — Release a latch.....	98
ABEND codes.....	100
Return codes.....	100
Example.....	100
ISGLPRG — Purge a requestor from a latch set.....	101
ABEND codes.....	101
Return codes.....	101
Example.....	102
ISGLPBA — Purge a group of requestors from a group of latch sets.....	102
ABEND codes.....	103
Return codes.....	103
Part 4. Resource recovery services (RRS).....	105
Chapter 10. Using protected resources.....	107
Resource recovery programs.....	107
Two-phase commit protocol.....	108
Resource recovery process.....	108
Requesting resource protection and recovery.....	110
Using distributed resource recovery.....	110
Application_Backout_UR (SRRBACK).....	111
Description.....	111
Application_Commit_UR (SRRCMIT).....	114
Description.....	114
Additional callable services.....	118
Part 5. CEA TSO/E address space services.....	119
Chapter 11. Introduction to CEA TSO/E address space services.....	121
CEA TSO/E address space manager components.....	121
System prerequisites for the CEA TSO/E address space services.....	122
Working with TSO/E address spaces started by CEA	123
Communicating with programs running in the TSO/E address spaces.....	124
Reconnecting to CEA TSO/E address spaces.....	127
Chapter 12. Using CEA TSO/E address space services.....	129
Invoking the CEATsoRequest API.....	129
Parameters.....	129
Requirements for callers.....	134
Understanding the request types.....	134
Invoking the CEAMsgsnd API.....	143

Invoking the CEAMsgrcv API.....	145
Invoking the CEAWSNDT API.....	146
Return, reason, and diagnostic codes.....	148
Return codes.....	148
Reason codes.....	149
Diagnostic codes.....	158
CEAYTSOR header file.....	162
CEAXRDEF header file.....	165
Programming example.....	169
Sample compile job.....	183
Part 6. zEnterprise Data Compression (zEDC).....	185
Chapter 13. Overview and planning of zEnterprise Data Compression (zEDC).....	187
Requirements for zEnterprise Data Compression.....	188
Product Enablement for zEnterprise Data Compression.....	188
Planning for zEnterprise Data Compression.....	189
Chapter 14. Application interfaces for zEnterprise Data Compression.....	191
Invoking unauthorized interfaces for zEnterprise Data Compression.....	191
zlib for zEnterprise Data Compression.....	191
Invoking z System authorized interfaces for zEnterprise Data Compression.....	196
z System authorized compression services.....	197
Chapter 15. Troubleshooting for zEDC.....	215
Part 7. Other callable services.....	217
Chapter 16. IEAAFFN — Assign processor affinity for encryption or decryption.....	219
Restrictions and limitations.....	220
Requirements.....	220
Return codes.....	220
Chapter 17. CSRL16J/CSRLJ1 — Transfer control with all registers intact.....	221
Defining the entry characteristics of the target routine.....	221
Freeing dynamic storage associated with the caller.....	222
Programming requirements.....	222
Restrictions.....	225
Performance implications.....	225
Syntax diagram.....	226
C/370 syntax.....	226
PL/I syntax.....	226
Parameters.....	226
Return codes.....	226
Example.....	227
C/370 example program.....	227
Assembler program for use with the C/370 example.....	229
Chapter 18. CSRSI — System information service.....	231
Description.....	231
Environment.....	231
Programming requirements.....	231
Restrictions.....	231
Input register information.....	232
Output register information.....	232
Syntax.....	232
Parameters.....	232

Return codes.....	234
CSRSIC C/370 header file.....	234

Part 8. Base Control Program internal interface (BCPii) services.....243

Chapter 19. Base Control Program internal interface (BCPii).....	245
BCPii setup and installation.....	245
Setting up connectivity to the support element.....	246
Setting up authority to use BCPii.....	251
BCPii configuration considerations.....	256
Setting up event notification for BCPii z/OS UNIX applications.....	258
Setting up access to BCPii REXX execs.....	259
BCPii startup and shutdown.....	260
BCPii communication monitoring.....	261
SMF recording in BCPii.....	262
BCPii callable services.....	262
Syntax, linkage and programming considerations.....	263
Calling formats.....	263
BCPii connection scope.....	263
Linkage considerations.....	264
REXX programming considerations.....	264
Assembler programming considerations.....	273
Programming Examples.....	274
HWICMD / HWICMD2 — Issue a BCPii hardware management command.....	274
Description.....	274
HWICONN — Establish a BCPii connection.....	298
Description.....	298
HWIDISC — Release a BCPii connection.....	311
Description.....	311
HWIEVENT — Register or unregister for BCPii events.....	318
Monitoring events occurring on a particular CPC or image.....	318
Monitoring operating system message events (Hwi_Event_OpSysMsg).....	318
Monitoring communication availability between BCPii and the CPC.....	319
Monitoring the status of the BCPii address space.....	319
Description.....	320
HWILIST — Retrieve HMC and BCPii configuration-related information.....	332
Description.....	332
HWIQUERY — BCPii retrieval of SE/HMC-managed attributes.....	347
Description.....	347
HWIREST — Issue RESTlike requests to the SE.....	378
Description.....	378
HWIREST2 Callable Service.....	394
Description.....	394
HWISET/HWISSET2 — BCPii set single or multiple SE/HMC-managed attributes.....	399
Description.....	399
HWIBeginEventDelivery — Begin delivery of BCPii event notifications.....	446
Description.....	446
HWIEndEventDelivery — End delivery of BCPii event notifications.....	450
Description.....	450
HWIManageEvents — Manage the list of BCPii events.....	453
Description.....	453
HWIGetEvent — Retrieve outstanding BCPii event notifications.....	458
Description.....	458

Part 9. z/OS client web enablement toolkit.....465

Chapter 20. The z/OS JSON parser.....	467
---------------------------------------	-----

Elements of the z/OS JSON parser.....	468
Availability of the z/OS JSON parser.....	469
Syntax, linkage, and programming considerations.....	470
z/OS JSON parser callable services.....	475
HWTCONST — Initialize predefined variables (REXX).....	477
HWTJCREN — Create JSON entry.....	478
HWTJDEL — Delete a JSON entry.....	490
HWTJESCT — Encode or decode escape sequences (REXX).....	497
HWTJGAEN — Get array entry.....	498
HWTJGBOV — Get boolean value.....	502
HWTJGENC — Get JSON encoding.....	506
HWTJGJST — Get JSON type.....	510
HWTCONST — Initialize predefined variables (REXX).....	514
HWTJGNUE — Get number of entries.....	515
HWTJGNUV — Get number value (non-REXX).....	519
HWTJGOEN — Get object entry.....	525
HWTJGVAL — Get value.....	531
HWTJINIT — Initialize a parser instance.....	535
HWTJOPTS — Set parser options.....	539
HWTJPARS — Parse a JSON string.....	544
HWTJSENC — Set JSON encoding.....	551
HWTJSERI — Serialize (build) JSON text.....	555
HWTJSRCH — Search.....	561
HWTJTERM — Terminate a parser instance.....	569
 Chapter 21. The z/OS HTTP/HTTPS protocol enabler.....	575
Elements of the z/OS HTTP/HTTPS enabler.....	576
Availability of the z/OS HTTP/HTTPS enabler.....	577
Syntax, linkage, and programming considerations.....	577
AT-TLS usage overview.....	590
Server identity.....	593
z/OS HTTP/HTTPS callable services.....	594
HWTCONST — Initialize predefined variables (REXX).....	595
HWTCONN — Connect to an HTTP server.....	596
HWTDISC — Disconnect from an HTTP server.....	602
HWTINIT — Initialize an HTTP connection or request.....	609
HWTREQST — Send a request to an HTTP server.....	613
HWTRESET — Reset an HTTP connection or request.....	619
HWTSET — Set HTTP connection or request options.....	624
HWTSLST — Linked list append service.....	631
HWTTERM — Terminate an HTTP connection or request.....	638
HTTP/HTTPS enabler options and values.....	643
Options for connections.....	643
Options for requests.....	653
Capturing trace data through environment variables.....	656
Sending data to a server (non-REXX).....	659
Buffer with the HWT_OPT_REQUESTBODY option.....	660
Streaming send exit.....	660
Receiving data from a server (non-REXX).....	661
Processing response headers with the response header callback routine.....	662
Response body processing options.....	662
Usage considerations for the toolkit callback routines.....	664
 Part 10. SMF Services.....	665
 Chapter 22. SMF real-time interface.....	667
IFAMCON — Connect to an SMF in-memory resource.....	667

IFAMDSC — Disconnect from an SMF in-memory resource.....	670
IFAMGET — Obtain data from an SMF in-memory resource.....	673
IFAMQRY — Query SMF in-memory resources.....	677
Part 11. Cloud Data Access (CDA) Services.....	683
Chapter 23. Introduction to DFSMSdfp Cloud Data Access (CDA).....	685
Chapter 24. Cloud Data Access configuration.....	687
System administrator configuration quick-start.....	687
User configuration quick-start.....	689
Chapter 25. Cloud Data Access files.....	691
Key file.....	691
Config file.....	691
Provider file.....	692
Chapter 26. Cloud Data Access (CDA) API basics.....	699
Elements of Cloud Data Access.....	699
Chapter 27. Cloud Data Access cloud credential storage.....	701
Error conditions.....	703
Credential panels for different authentication types.....	704
Chapter 28. Availability of the Cloud Data Access callable services.....	707
Chapter 29. Syntax, linkage, and programming considerations.....	709
Chapter 30. Cloud Data Access callable services.....	715
GDKGET — Retrieve a cloud object.....	716
GDKWRITE — Write an object to cloud storage.....	728
GDKDEL — Delete an object from cloud storage.....	742
GDKLIST — List cloud objects.....	748
GDKLISTL — List cloud objects.....	755
GDKMSGTR — Translate a CDA return code into text.....	762
GDKGETP — List cloud providers.....	765
GDKKEYSR — Retrieve cloud credentials.....	769
GDKKEYAD — Store cloud credentials.....	773
GDKKEYDL — Delete cloud credentials.....	778
GDKKEYGR — List resources for provider.....	781
GDKINIT — Initialize a Session.....	785
GDKTERM — Terminate a Session.....	789
GDKGEN — Perform a Configurable request.....	792
GDKVALD — Validate Provider File.....	802
GDKQUERY — Query available CDA functions.....	807
gdkkeysJ/gdkkeysr64J — Retrieve cloud credentials JSON API.....	811
gdkkeyadJ/gdkkeyad64J — Store cloud credentials JSON API.....	813
gdkkeydlJ/gdkkeydl64J — Delete cloud credentials JSON API.....	815
gdkkeygrJ/gdkkeygr64J — List cloud credentials resources JSON API.....	817
Appendix A. BCPii communication error reason codes.....	819
Appendix B. BCPii summary tables.....	821
BCPii configuration considerations.....	821
HWICMD / HWICMD2.....	821
HWIEVENT.....	823
HWIQUERY and HWISET / HWISET2 attributes.....	825

HWIREST attributes.....	840
Appendix C. General use C/C++ header files.....	841
Appendix D. Accessibility.....	843
Notices.....	845
Terms and conditions for product documentation.....	846
IBM Online Privacy Statement.....	847
Policy for unsupported hardware.....	847
Minimum supported hardware.....	847
Additional notices.....	848
Programming interface information.....	848
Trademarks.....	848
Glossary.....	849
Index.....	851

Figures

1. Structure of a Data Object.....	4
2. Mapping a Permanent Object That Has No Scroll Area.....	5
3. Mapping a Permanent Object That Has a Scroll Area.....	5
4. Mapping a Temporary Object.....	6
5. Mapping an Object to Multiple Windows.....	6
6. Mapping Multiple Objects.....	7
7. Illustration of a Reference Pattern with a Gap.....	64
8. Two Typical Reference Patterns.....	66
9. Illustration of Forward Direction of Reference.....	67
10. Illustration of Backward Direction of Reference.....	67
11. ATM Transaction.....	108
12. Two-Phase Commit Actions.....	109
13. Backout — Application Request.....	109
14. Backout — Resource Manager Votes NO.....	110
15. Transaction — Distributed Resource Recovery.....	110
16. Sample REXX EXEC.....	124
17. Example illustrating that the REXX SYSTERMID is the same as the z/OSMF ISPF application identifier	124
18. Sample TSO/E messages written to the queue.....	126
19. Contents included in the ceasapit.x file.....	129
20. Compression and Decompression Workflow.....	187
21. CSRLJPLI declarations for return codes for PL/I.....	225
22. BCPii setup and installation steps.....	246

23. Tasks index.....	250
24. Customize API settings.....	251
25. Retrieve LPAR GPP weight for the LOCAL LPAR.....	379
26. Retrieve LPAR GPP weight when the CPC and LPAR name are known.....	380
27. Polling result of an asynchronous operation.....	380
28. Example of JSON text.....	467
29. Example of JSON single line comment.....	468
30. Example of JSON multi-line comment on one line.....	468
31. Example of JSON multi-line comment.....	468
32. Example of comments on JSON data.....	468
33. Example of JSON single line comment.....	540
34. Example of JSON multi-line comment on one line.....	541
35. Example of JSON multi-line comment.....	541
36. Example of JSON multi-line comment.....	541
37. Example of comments on JSON data.....	541
38. Commented JSON Example 1.....	541
39. Commented JSON Example 2.....	541
40. Commented JSON Example 3.....	541
41. Invoking the Cloud Data Access authorization utility from the ISPF command shell.....	702
42. Cloud Data Access authorization utility Options Menu.....	703
43. Cloud Data Access authorization utility entering alternate credentials.....	703

Tables

1. CSREVV Return and Reason Codes.....	21
2. CSRIDAC Return and Reason Codes.....	25
3. CSRREFR Return and Reason Codes.....	28
4. CSRSAVE Return and Reason Codes.....	30
5. CSRSCOT Return and Reason Codes.....	32
6. CSRVIEW Return and Reason Codes.....	35
7. ISGLCRT Return Codes.....	93
8. ISGLOBT Return Codes.....	97
9. ISGLREL Return Codes.....	100
10. ISGLPRG Return Codes.....	102
11. ISGLPBA Return Codes.....	104
12. CEA TSO/E address space manager components.....	121
13. System prerequisites.....	122
14. Message type identifiers.....	125
15. Message types.....	125
16. Data types.....	126
17. Input and output for each structure used for the CeaTsoStart request type.....	135
18. Input and output for each structure used for the CeaTsoAttn request type.....	136
19. Input and output for each structure used for the CeaTsoEnd request type.....	137
20. Input and output for each structure used for the CeaTsoPing request type.....	138
21. Input and output for each structure that is used for Version 1 of the CeaTsoQuery request type.....	138
22. Input and output for each structure that is used for Version 2 of the CeaTsoQuery request type.....	140
23. Input and output for each structure used for Version 1 of the CeaTsoQueryApp request type.....	141

24. Input and output for each structure used for Version 2 of the CeaTsoQueryApp request type.....	142
25. Return codes.....	148
26. Reason codes.....	149
27. Diagnostic code.....	159
28. Comparison table between unauthorized and z System authorized interfaces for zEDC.....	189
29. Standard zlib functions and whether they are supported using zEDC.....	192
30. Compression and decompression with zlib.....	196
31. Compression and decompression with z System authorized interfaces for zEDC.....	197
32. Environment for the FPZ4RZV service.....	198
33. Parameters for the FPZ4RZV service.....	198
34. Return and reason codes for the FPZ4RZV service.....	199
35. Environment for the FPZ4PRB service.....	200
36. Parameters for the FPZ4PRB service.....	201
37. Return and Reason Codes for the FPZ4PRB service.....	201
38. Environment for the FPZ4RMR service.....	202
39. Parameters for the FPZ4RMR service.....	202
40. Return and Reason Codes for the FPZ4RMR service.....	203
41. Environment for the FPZ4DMR service.....	204
42. Parameters for the FPZ4DMR service.....	205
43. Return and Reason Codes for the FPZ4DMR service.....	205
44. Environment for the FPZ4ABC service.....	206
45. Parameters for the FPZ4ABC service.....	206
46. Header elements in the FPZ4ABC-generated list.....	208
47. Entries elements in the FPZ4ABC-generated list.....	208
48. Return and Reason Codes for the FPZ4ABC service.....	208

49. Environment for the FPZ4URZ service.....	211
50. Parameters for the FPZ4URZ service.....	211
51. Return and Reason Codes for the FPZ4URZ service.....	212
52. IEAAFFN Return Codes.....	220
53. CSRL16J/CSRLJ1 Return Codes.....	226
54. Minimum BCPii microcode levels by SE hardware level.....	247
55. Minimum BCPii microcode levels by HMC level.....	247
56. Minimum BCPii microcode levels by LPAR level.....	248
57. BCPii APIs supported in the REXX environment.....	264
58. HWIREXX keywords.....	265
59. Return codes from the HWIREXX service.....	266
60. Return codes from a REXX BCPii host command.....	270
61. REXX return codes from the BCPii hwihost function.....	273
62. HWICMD syntax.....	276
63. HWICMD2 syntax.....	276
64. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX).....	280
65. Reasons for abend X'042', RC X'0001yyyy' for HWICMD or X'0008yyyy' for HWICMD2 for.....	290
66. Reasons for abend X'042', RC X'0002yyyy'.....	303
67. Reasons for abend X'042', RC X'0003yyyy'.....	314
68. Reasons for abend X'042', RC X'0004yyyy'.....	325
69. Reasons for abend X'042', RC X'0005yyyy'.....	338
70. Valid query attribute identifiers.....	350
71. Reasons for abend X'042', RC X'0006yyyy'.....	369
72. Reasons for abend X'042', RC X'0009yyyy' for HWIREST.....	382
73. Non-REXX parameters.....	383

74. RequestParmPtr parameter.....	383
75. ResponseParmPtr parameter.....	385
76. REXX parameters.....	389
77. RequestParm stem tail variables.....	390
78. ResponseParm stem variables.....	392
79. Reasons for abend X'042', RC X'0009yyyy' for HWIREST2.....	396
80. Request2ParmPtr parameter.....	396
81. Non-REXX parameters.....	399
82. HWISET syntax.....	401
83. HWISET2 syntax.....	401
84. Parameters of the (SetParm) structure pointed by the SetParm_Ptr	436
85. Reasons for abend X'042', RC X'0007yyyy' for HWISET or X'0009yyyy' for HWISET2.....	437
86. Reasons for abend X'042', RC X'0004yyyy'	455
87. JSON parser programming interface.....	470
88. Calling formats for the z/OS JSON parser callable services.....	470
89. Host return codes for REXX.....	473
90. JSON parser programming sample files.....	475
91. Return codes for the HWTCONST service.....	478
92. Return codes for the HWTJCREN service.....	483
93. Return codes for the HWTJDEL service.....	492
94. Return codes for the HWTJESCT service.....	498
95. Return codes for the HWTJGAEN service.....	500
96. Return codes for the HWTJGBOV service.....	504
97. Return codes for the HWTJGENC service.....	508
98. Return codes for the HWTJGJST service.....	512

99. Return codes for the HWTCONST service.....	515
100. Return codes for the HWTJGNUE service.....	517
101. Return codes for the HWTJGNUV service.....	522
102. Return codes for the HWTJGOEN service.....	528
103. Return codes for the HWTJGVAL service.....	533
104. Return codes for the HWTJINIT service.....	537
105. Return codes for the HWTJOPTS service.....	542
106. Return codes for the HWTJPARS service.....	547
107. Return codes for the HWTJSENC service.....	553
108. Return codes for the HWTJSERI service.....	558
109. Return codes for the HWTJSRCH service.....	565
110. Return codes for the HWTJTERM service.....	571
111. HTTP enabler.....	577
112. Calling formats for the z/OS HTTP enabler callable services.....	577
113. Toolkit handling of HTTP redirection status response codes.....	586
114. Host return codes for REXX.....	587
115. z/OS HTTP enabler programming sample files.....	589
116. AT-TLS policy types.....	592
117. Verification rules.....	594
118. Return codes for the HWTCONST service.....	596
119. Return codes for the HWTHCONN service.....	598
120. Return codes for the HWTHDISC service.....	604
121. Return codes for the HWTHINIT service.....	611
122. Return codes for the HWTHRQST service.....	615
123. Return codes for the HWTHRSET service.....	621

124. Return codes for the HWTHTSET service.....	627
125. Return codes for the HWTHTSLST service.....	634
126. Return codes for the HWTHTTERM service.....	640
127. Return and reason codes for the IFAMCON service.....	669
128. Return and reason codes for the IFAMDSC service.....	672
129. Return and reason codes for the IFAMGET service.....	676
130. Return and reason codes for the IFAMQRY service.....	680
131. DFSMSdftp CDA variables.....	697
132. ICSF reason codes.....	704
133. CDA parser programming interface.....	709
134. Calling formats for the z/OS Cloud Data Access parser callable services.....	709
135. GDK_OPTIONAL_PARMS_TYPE.....	715
136. GDK_OPTION_TYPE.....	716
137. Return and reason codes for the GDKGET service.....	726
138. IGGRPFx mapping.....	737
139. List of CDA symbols and their meaning.....	737
140. Return and reason codes for the GDKWRITE service.....	741
141. Return and reason codes for the GDKDEL service.....	747
142. GDK_LISTOBJECTS_TYPE.....	749
143. GDK_OBJECTDESCRIPTION_TYPE.....	749
144. Optional parameters.....	750
145. Return and reason codes for the GDKLIST service.....	754
146. GDK_LISTOBJECTS_TYPE.....	756
147. GDK_LARGEOBJECTDESCRIPTION_TYPE.....	757
148. Optional parameters.....	758

149. Return and reason codes for the GDKLIST service.....	761
150. Return and reason codes for the GDKMSGTR service.....	765
151. GDK_PROVIDER_NAME_TYPE.....	766
152. GDK_PROVIDER_NAME_LIST_TYPE.....	766
153. Return and reason codes for the GDKGETP service.....	768
154. Return and reason codes for the GDKKEYSR service.....	772
155. Return and reason codes for the GDKKEYAD service.....	777
156. Return and reason codes for the GDKKEYDL service.....	781
157. SEARCH_RESULT_ELEMENT_TYPE.....	782
158. SEARCH_RESULT_STRUCT_TYPE.....	782
159. Return and reason codes for the GDKKEYGR service.....	784
160. Return and reason codes for the GDKKEYGR service.....	789
161. Return and reason codes for the GDKKEYGR service.....	792
162.	794
163.	795
164.	796
165. Return and reason codes for the GDKKEYGR service.....	801
166. Return and reason codes for the GDKKEYGR service.....	807
167. Return and reason codes for the GDKKEYGR service.....	811
168. HWICMD types.....	821
169. HWIEVENT types.....	823
170. HWIQUERY and HWISET / HWISET2 attributes.....	825

About this information

Callable services are for use by any program coded in C, COBOL, FORTRAN, Pascal, or PL/I — this information 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 information

This information 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 information

This information 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*](#).

z/OS information

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross-document links that go directly to the topic in reference 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*](#).

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. For more information, see [How to send feedback to IBM](#).

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under [IBM z/OS Product Documentation Update Policy](http://www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy) (www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy).

Summary of changes for z/OS 3.2

The following content is new, changed, or no longer included in z/OS 3.2.

New

The following content is new.

September 2025 release

- JWT authorization to use BCPii is added. See [“Setting up JSON Web Tokens \(JWTs\) for authorization”](#) on page 254 for more information.
- HWIREST2 service is added. See [“HWIREST2 Callable Service”](#) on page 394 for more information.

Changed

The following content is changed.

September 2025 release

- The Get-Body-Buffer-Size is updated in [#unique_13/unique_13_Connect_42_atable_lxd_4gv_vwb](#) on page 730, [#unique_14/unique_14_Connect_42_ctable_lxd_4gv_vwb](#) on page 743, and [#unique_15/unique_15_Connect_42_utable_lxd_4gv_vwb](#) on page 797.
- The Get-Header-Buffer-Size is updated in [#unique_16/unique_16_Connect_42_table_orj_2jr_vwb](#) on page 718, [#unique_13/unique_13_Connect_42_atable_lxd_4gv_vwb](#) on page 730, [Table 144](#) on page 750, [Table 148](#) on page 758, and [#unique_15/unique_15_Connect_42_utable_lxd_4gv_vwb](#) on page 797.
- [“GDKGET — Retrieve a cloud object”](#) on page 716 is updated with return and reason code 158
- [“Authority to the particular resource”](#) on page 252 is updated with information on checking authority to a particular resource in BCPii.
- HWIREST service is updated to include BCPii enhancements. See sections within [“HWIREST — Issue RESTlike requests to the SE”](#) on page 378 for more information.

Deleted

The following content is deleted.

September 2025 release

- None.

Summary of changes for z/OS 3.1

The following content is new, changed, or no longer included in z/OS 3.1.

New

The following content is new.

May 2025 refresh

- Added optional parameters to support GDKUTIL program in “GDKGET — Retrieve a cloud object” on page 716 as per APAR OA66579 (www.ibm.com/support/pages/apar/OA66579)(APAR OA66579 applies to z/OS 3.1 and above).
- Added Metadata tags table that support GDKUTIL program in “GDKGET — Retrieve a cloud object” on page 716 as per APAR OA66579 (www.ibm.com/support/pages/apar/OA66579)(APAR OA66579 applies to z/OS 3.1 and above).
- Added Return code 159, 160 and 161 in Table 137 on page 726 as per APAR OA66579 (www.ibm.com/support/pages/apar/OA66579)(APAR OA66579 applies to z/OS 3.1 and above).
- Added cloudformat under Optional parameters in “GDKWRITE — Write an object to cloud storage” on page 728 as per APAR OA66579 (www.ibm.com/support/pages/apar/OA66579)(APAR OA66579 applies to z/OS 3.1 and above).
- Added IGGRPFX mapping table as per APAR OA66579 (www.ibm.com/support/pages/apar/OA66579)(APAR OA66579 applies to z/OS 3.1 and above).
- Added CDA symbols to support GDKUTIL program in “GDKWRITE — Write an object to cloud storage” on page 728 as per APAR OA66579 (www.ibm.com/support/pages/apar/OA66579)(APAR OA66579 applies to z/OS 3.1 and above).
- Added applications supported by CDA for GDKUTIL program in “Elements of Cloud Data Access” on page 699 as per APAR OA66579 (www.ibm.com/support/pages/apar/OA66579)(APAR OA66579 applies to z/OS 3.1 and above).
- Added new API Entry Offset from DFVCDVAVT values in Chapter 29, “Syntax, linkage, and programming considerations,” on page 709 as per APAR OA65068 (www.ibm.com/support/pages/apar/OA65068)). (APAR OA65068 applies to z/OS 3.1 and above).
- Added new Optional Parameter value in “GDKKEYGR — List resources for provider” on page 781 as per APAR OA65068 (www.ibm.com/support/pages/apar/OA65068)). (APAR OA65068 applies to z/OS 3.1 and above).
- Added new Optional Parameter value in “GDKKEYSR — Retrieve cloud credentials” on page 769 as per APAR OA65068 (www.ibm.com/support/pages/apar/OA65068)). (APAR OA65068 applies to z/OS 3.1 and above).

December 2024 refresh

- A new function is added. For more information, see the following files. (APAR OA65990, which applies to both z/OS 2.5 and 3.1)
 - [“gdkkeyadJ/gdkkeyad64J — Store cloud credentials JSON API” on page 813](#)
 - [“gdkkeydlJ/gdkkeydl64J — Delete cloud credentials JSON API” on page 815](#)
 - [“gdkkeygrJ/gdkkeygr64J — List cloud credentials resources JSON API” on page 817](#)
 - [“gdkkeysrJ/gdkkeysr64J — Retrieve cloud credentials JSON API” on page 811](#)
 - [“GDKKEYAD — Store cloud credentials” on page 773](#)
 - [“GDKKEYDL — Delete cloud credentials” on page 778](#)
 - [“GDKKEYGR — List resources for provider” on page 781](#)
 - [“GDKKEYSR — Retrieve cloud credentials” on page 769](#)
 - [“GDKGEN — Perform a Configurable request” on page 792](#)
 - [“GDKINIT — Initialize a Session” on page 785](#)
 - [“GDKTERM — Terminate a Session” on page 789](#)
 - [“GDKDEL — Delete an object from cloud storage” on page 742](#)
 - [“GDKGET — Retrieve a cloud object” on page 716](#)

- [“GDKGETP — List cloud providers” on page 765](#)
- [“GDKLIST — List cloud objects” on page 748](#)
- [“GDKLISTL — List cloud objects” on page 755](#)
- [“GDKMSGTR — Translate a CDA return code into text” on page 762](#)
- [“GDKWRITE — Write an object to cloud storage” on page 728](#)
- [“GDKQUERY — Query available CDA functions” on page 807](#)
- [“GDKVALD — Validate Provider File” on page 802](#)
- [Chapter 29, “Syntax, linkage, and programming considerations,” on page 709](#)
- DFSMSdftp supports the TLS1.3 security level for communication. See [“Provider file” on page 692](#).

August 2024 refresh

- A new API called GDKLISTL allows callers to receive information about objects that are larger than 4GB in size. (APAR OA65451)

June 2024 refresh

- New key value pairs are added. For more information, see the following files
 - [“Provider file” on page 692](#) (APAR OA65717, which applies to both z/OS 2.4 and 2.5)
- New section on [“BCPii communication monitoring” on page 261](#).

May 2024 refresh

- A new function is added. For more information, see the following files
 - [“Provider file” on page 692](#)
 - [“GDKGEN — Perform a Configurable request” on page 792](#)
 - [“GDKINIT — Initialize a Session” on page 785](#)
 - [“GDKTERM — Terminate a Session” on page 789](#)
 - [“Elements of Cloud Data Access” on page 699](#)
 - [Chapter 29, “Syntax, linkage, and programming considerations,” on page 709](#)
 - [“GDKDEL — Delete an object from cloud storage” on page 742](#)
 - [“GDKGET — Retrieve a cloud object” on page 716](#)
 - [“GDKGETP — List cloud providers” on page 765](#)
 - [“GDKLIST — List cloud objects” on page 748](#)
 - [“GDKMSGTR — Translate a CDA return code into text” on page 762](#)
 - [“GDKWRITE — Write an object to cloud storage” on page 728](#)
 - [“GDKQUERY — Query available CDA functions” on page 807](#)
 - [“GDKVALD — Validate Provider File” on page 802](#)
 - [Chapter 30, “Cloud Data Access callable services,” on page 715](#)

April 2024 refresh

- [“Product Enablement for zEnterprise Data Compression” on page 188](#) is a new section for [Chapter 13, “Overview and planning of zEnterprise Data Compression \(zEDC\),” on page 187](#).
- Information about a thread-safe data key is added to [“Running zlib” on page 194](#). (APAR OA75661, which applies to both z/OS 2.5 and 3.1)

February 2024 release

- Two new return codes are added to the [“HWTJGOEN — Get object entry” on page 525](#) description.

December 2023 release

- In “Options for connections” on page 643, there is a new socket option, “[HWTH_OPT_CONNECT_TIMEOUT_MS](#)” on page 647, that sets a timeout value, in milliseconds, for a connection to wait while attempting to connect a socket.

November 2023 release

- In Chapter 24, “Cloud Data Access configuration,” on page 687, the “Provider file” on page 692 is updated to support a new request parameter, **METAHEADER**, that describes how to construct a metadata header.
- Part 11, “Cloud Data Access (CDA) Services,” on page 683 is updated to support a new **metadata** optional parameter in “[GDKWRITE — Write an object to cloud storage](#)” on page 728. There is also a new table of [Table 131](#) on page 697. “[GDKLIST — List cloud objects](#)” on page 748 is updated to add two new optional parameters, **Get-Header-Buffer** and **Get-Header-Buffer-Size**, to the [Table 144](#) on page 750 table.

September 2023 release

- Part 11, “Cloud Data Access (CDA) Services,” on page 683 is added in support of z/OS DFSMSdfp Cloud Data Access.

Changed

The following content is changed.

December 2024 refresh

- The provider file is updated. For more information, see the following file “[Provider file](#)” on page 692.

September 2024 refresh

- The provider file is updated. For more information, see the following files “[GDKWRITE — Write an object to cloud storage](#)” on page 728 and “[Provider file](#)” on page 692.

May 2024 refresh

- A usage note is added to “[Running zlib](#)” on page 194. (APAR OA65661, which applies to z/OS 2.5 and later)

Part 1. Window services

Chapter 1. Introduction to window services

Window services allow HLL programs to:

- Read or update an existing permanent data object
- Create and save a new permanent data object
- Create and use a temporary data object

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

Permanent data objects

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

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

Temporary data objects

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

Structure of a data object

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

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

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

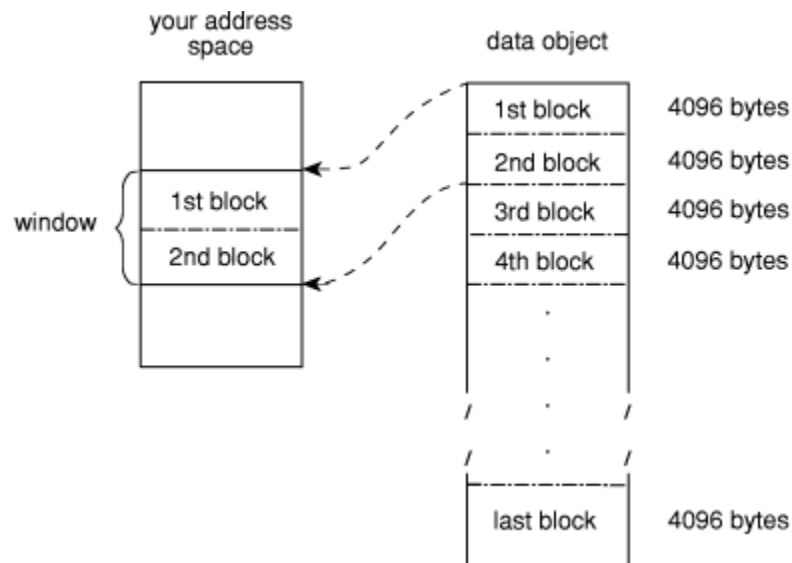


Figure 1. Structure of a Data Object

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.

Note:

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

The ways that window services can map an object

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

- Map a permanent object that has no scroll area
- Map a permanent object that has a scroll area

- Map a temporary object
- Map an object to multiple windows
- Map multiple objects

Example 1 — Mapping a permanent object that has no scroll area

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

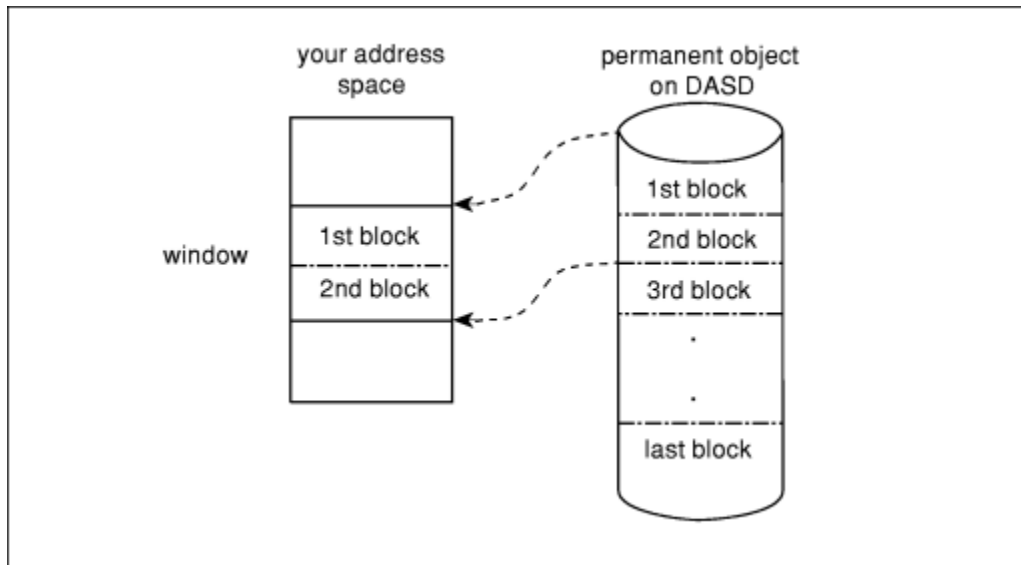


Figure 2. Mapping a Permanent Object That Has No Scroll Area

Example 2 — Mapping a permanent object that has a scroll area

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

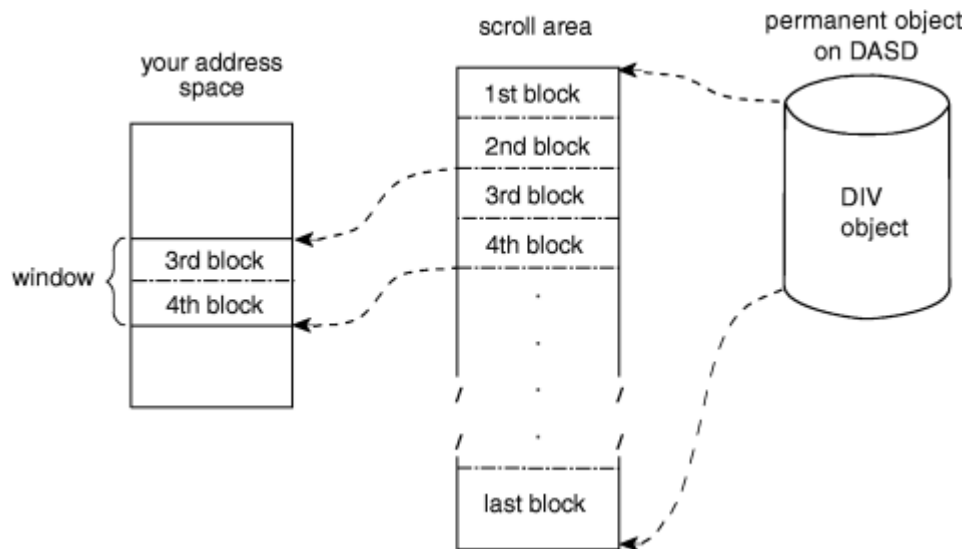


Figure 3. Mapping a Permanent Object That Has a Scroll Area

Example 3 — Mapping a temporary object

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

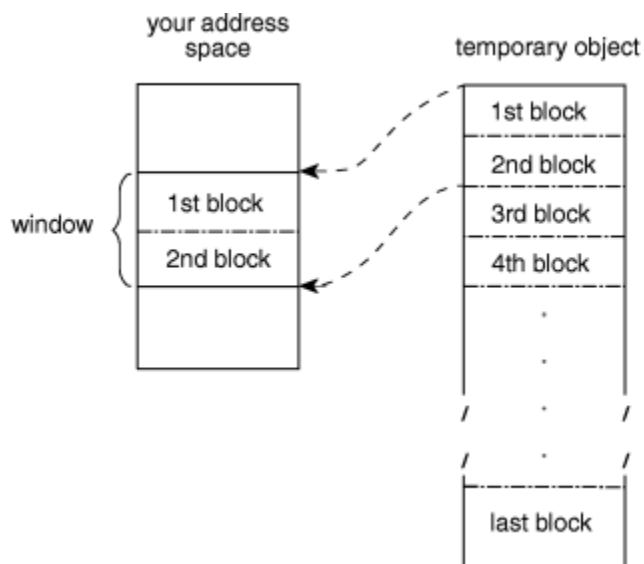


Figure 4. Mapping a Temporary Object

Example 4 — Mapping multiple Windows to an object

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

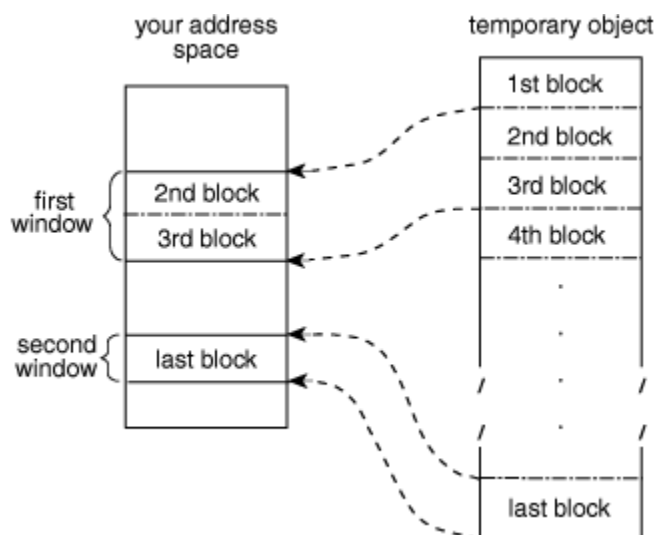


Figure 5. Mapping an Object to Multiple Windows

Example 5 — Mapping multiple objects

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

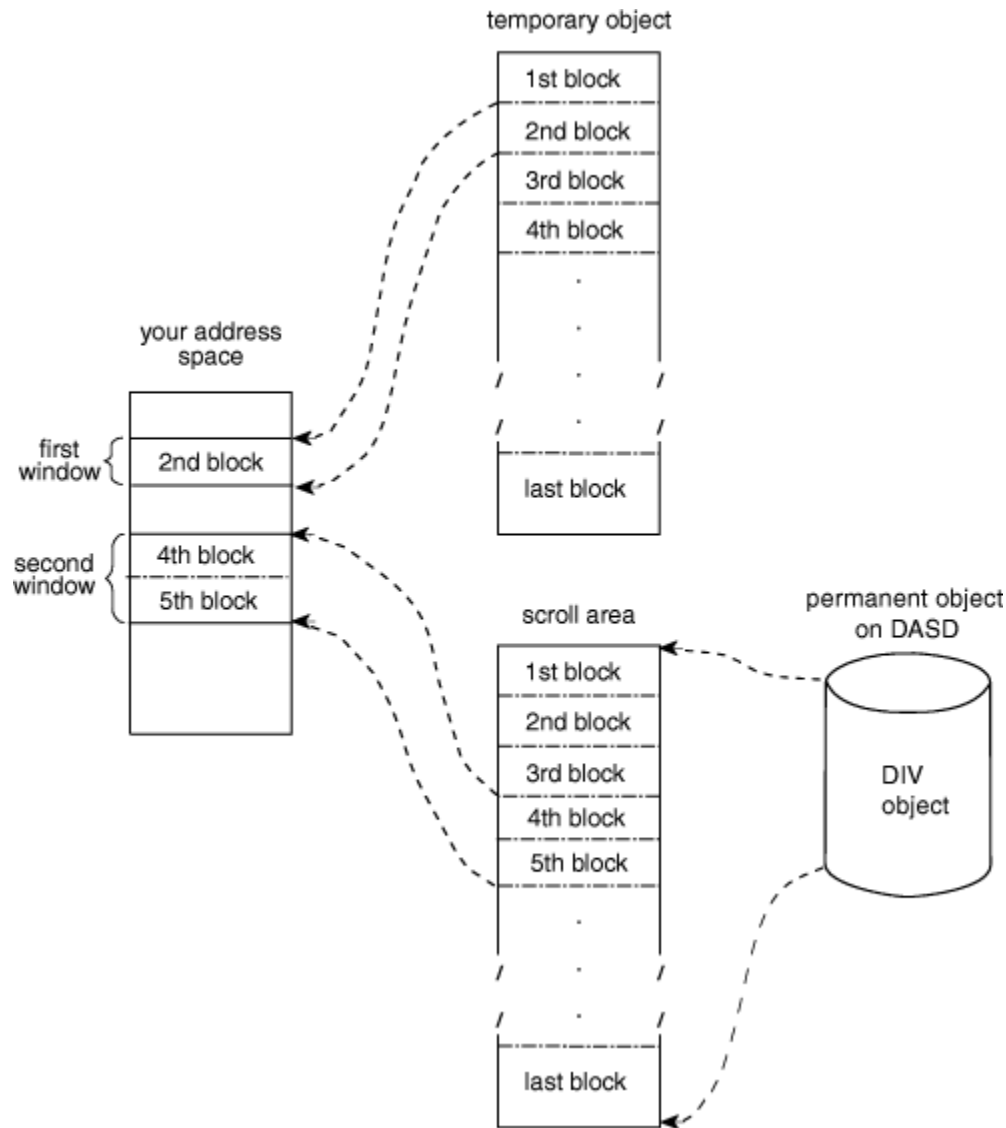


Figure 6. Mapping Multiple Objects

Access to permanent data objects

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

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

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

Access to temporary data objects

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

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

Chapter 2. Using window services

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

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

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

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

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

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

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

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

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

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

The following restrictions apply to using window services:

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

This topic explains how to do the previously described functions and contains the following subtopics:

- [“Obtaining access to a data object” on page 10](#)
- [“Defining a view of a data object” on page 11](#)
- [“Defining multiple views of an object” on page 14](#)
- [“Saving interim changes to a permanent data object” on page 15](#)
- [“Updating a temporary data object” on page 15](#)
- [“Refreshing changed data” on page 16](#)
- [“Updating a permanent object on DASD” on page 16](#)
- [“Changing a view in a window” on page 17](#)
- [“Terminating access to a data object” on page 18](#)
- [“Handling return codes and abnormal terminations” on page 18.](#)

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 22.](#)

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 DSNNAME as the value for *object_type*.

If you specify DSNNAME 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*.

Note: 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 CSREVV. Whether you use CSRVIEW or CSREVV depends on how you plan to reference the data. [“Defining the expected reference pattern” on page 13](#) describes the differences between the two services. Specify BEGIN on CSRVIEW or CSREVV as the type of operation. For descriptions of the CALL syntax and return codes from CSRVIEW or CSREVV, see [“CSRVIEW — View an object” on page 32](#) or [“CSREVV — View an object and sequentially access it” on page 19](#).

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 14](#).

Defining the disposition of a window’s contents

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

Replace option

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

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

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

Retain option

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

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

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

Defining the expected reference pattern

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

Identifying the blocks you want to view

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

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

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

Offset	Span	Resulting view in the window
0	0	view the entire object
0	1	view the first block only
1	0	view the second block through the last block
1	1	view the second block only
2	2	view the third and fourth blocks only

Extending the size of a data object

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

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

Defining multiple views of an object

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

- A non-overlapping view means that no two windows view the same block of the object. For example, a view is non-overlapping 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.

Non-overlapping views

To define multiple windows that have a non-overlapping view, call CSRIDAC once to obtain the object identifier. Then call CSRVIEW or CSREVIEW 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 non-overlapping views.

Overlapping views

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

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

Saving interim changes to a permanent data object

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

After you have a view of the object and have made changes in the window, you can save those changes in the scroll area. To save changes in the scroll area, call CSRSCOT. For a description of the CSRSCOT parameters and return codes, see [“CSRSCOT — Save object changes in a scroll area” on page 30](#).

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

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

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

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

Updating a temporary data object

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

To update a temporary object, call CSRSCOT. For a description of the CSRSCOT parameters and return codes, see [“CSRSCOT — Save object changes in a scroll area” on page 30](#).

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 26](#).

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 28](#).

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 CSREVIEW, 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 32.

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

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

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

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

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

For example, assume that a window is mapped to one block of a permanent object that has no scroll area. The window contains the character string AAA.....A and the block to which the window is mapped contains BBB.....B. If you specify a value of RETAIN, upon completion of the CALL, the window still contains AAA.....A, and the mapped block contains BBB.....B. If you specify a value of REPLACE, upon completion of the CALL, the window contents are unpredictable and the mapped block still contains BBB.....B.

- For a permanent object that has a scroll area or for a temporary object:
 - To retain the data that is currently in the window, supply a value of RETAIN for *disposition*. CSRVIEW also updates the mapped blocks of the scroll area or temporary object so that they contain the same data as the window.
 - To discard the data that is currently in the window, supply a value of REPLACE for *disposition*. Upon completion of the operation, the window contents are unpredictable.

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

CSRVIEW ignores the values you assign to the other parameters.

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

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

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

Terminating access to a data object

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

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

To terminate access to an object, call CSRIDAC and supply a 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,” on page 19](#) 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 topic defines the allowable values that you can assign to the parameter.

This topic 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 37.

This topic contains the following subtopics:

- ["CSREVIEW — View an object and sequentially access it"](#) on page 19
- ["CSRIDAC — Request or terminate access to a data object"](#) on page 22
- ["CSRREFR — Refresh an object"](#) on page 26
- ["CSRSAVE — Save changes made to a permanent object"](#) on page 28
- ["CSRSCOT — Save object changes in a scroll area"](#) on page 30
- ["CSRVIEW — View an object"](#) on page 32

CSREVIEW — View an object and sequentially access it

Call CSREVIEW 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, CSREVIEW 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 CSREVIEW 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 CSREVIEW 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 CSREVIEW, 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 CSREVIEW 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 CSREVIEW uses to obtain input values, assign appropriate values. For parameters that CSREVIEW 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 CSREVV, to:
 - *object_id*
 - *offset*
 - *span*
 - *window_name*
 - *disposition*
 - *pfcount*
- CSREVV 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 statement	Parameters
CALL CSREVV	(operation_type ,object_id ,offset ,span ,window_name ,usage ,disposition ,pfcount ,return_code ,reason_code)

operation_type

Specify BEGIN to request that CSREVV 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 CSREVV 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 CSREVV is to handle data that is in the window when you begin a view. When you specify CSREVV BEGIN and a disposition of:

REPLACE

The first time you reference a block to which the window is mapped, CSREVIEW 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 CSREVIEW 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 21](#).

,reason_code

When CSREVIEW 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 21](#).

Abend codes

CSREVIEW issues abend code X'019'. For more information, see [z/OS MVS System Codes](#).

Return codes and reason codes

When CSREVIEW 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 1 on page 21 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 1. CSREVIEW Return and Reason Codes		
Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	Meaning: The operation was successful. Action: Continue normal program execution.
00000004 (4)	xxxx0125 (293)	Meaning: The operation was successful. The service could not retain all the data that was in the scroll area, however. Action: Notify your system programmer.
00000012 (18)	xxxx000A (10)	Meaning: There is another service currently executing with the specified ID. Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.

Table 1. CSREVV Return and Reason Codes (continued)		
Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx0017 (23)	Meaning: An I/O error has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx001A (26)	Meaning: The specified range does not encompass any mapped area of the object. Action: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx001C (28)	Meaning: The object cannot be accessed at the current time. Action: Try running your program at a later time. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0040 (64)	Meaning: The specified MAP range would cause the hiperspace data-in-virtual object to be extended such that the installation data space limits would be exceeded. Action: Change the MAP range you have specified or request your system programmer to increase the installation's data space limits.
0000000C (12)	xxxx0801 (2049)	Meaning: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. Action: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	Meaning: System error — I/O driver failure. Action: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	Meaning: System error — A necessary page table could not be read into real storage. Action: Notify your system programmer.
0000000C (12)	xxx00804 (2052)	Meaning: System error — Catalog update failed. Action: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	Meaning: System error — I/O error. Action: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	Meaning: System error — I/O from a previous request has not completed. Action: Notify your system programmer.
0000002C (44)	00000004 (4)	Meaning: Window services have not been defined to your system or the link to the service failed. Action: Notify your system programmer.

CSRIDAC — Request or terminate access to a data object

Call CSRIDAC to:

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

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown. 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 statement	Parameters
CALL CSRIDAC	(operation_type ,object_type ,object_name ,scroll_area ,object_state ,access_mode ,object_size ,object_id ,high_offset ,return_code ,reason_code)

operation_type

Specifies the type of operation the service is to perform:

- To request access to an object, specify BEGIN.
- To terminate access to an object, specify END. If the object is temporary, CSRIDAC deletes it.

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

,object_type

Specifies the type of object. The types are:

DDNAME

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

DSNAME

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

TEMPSPACE

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

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

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

,object_name

Specifies the data set name of a permanent object or the DDNAME of a data definition (DD) statement that defines a permanent object.

- If *object_type* is DDNAME, *object_name* must contain the name of a DD statement.
- If *object_type* is DSNAME, *object_name* must contain the data set name of the permanent object.

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

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

,scroll_area

Specifies whether window services is to create a scroll area for the data object.

YES

Create a scroll area.

NO

Do not create a scroll area.

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

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

,object_state

Specifies the state of the object.

OLD

The object exists.

NEW

The object does not exist and window services must create it.

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

Define *object_state* as character data of length 3.

,access_mode

Specifies the type of access required.

READ

READ access.

UPDATE

UPDATE access.

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

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

,object_size

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

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

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

Define *object_size* as integer data of length 4.

,object_id

Specifies the object identifier.

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

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

Define *object_id* as character data of length 8.

,high_offset

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

Define *high_offset* as integer data of length 4.

,return_code

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

Return codes and reason codes are explained under [“Return codes and reason codes” on page 25](#).

,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 25](#).

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 2 on page 25](#) 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 2. CSRIDAC Return and Reason Codes		
Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	Meaning: The operation was successful. Action: Continue normal program execution.
00000008 (8)	00000118 (280)	Meaning: The system could not obtain enough storage to create a hiperspace for the temporary object or the scroll area. Note: Hiperspace is the name the system uses to identify the storage it uses to create a temporary object or a scroll area for a permanent object. Action: Notify your system programmer. The system programmer might have to increase the SMF limit for data spaces and hiperspace that are intended for the user.
00000008 (8)	00000119 (281)	Meaning: The system could not delete or unidentify the temporary object or the scroll area. Action: Notify your system programmer.
00000008 (8)	0000011A (282)	Meaning: The system was unable to create a new VSAM linear data set. DFP 3.1 must be running and SMS must be active. Action: Notify your system programmer.
0000000C (12)	xxxx000A (10)	Meaning: Another service currently is executing with the specified ID. Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx001C (28)	Meaning: The object cannot be accessed at the current time. Action: Try running your program at a later time. If the problem persists, notify your system programmer.

Table 2. CSRIDAC Return and Reason Codes (continued)

Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx0037 (55)	Meaning: The caller invoked ACCESS. The access is successful, but the system is issuing a warning that the data set was not allocated with a SHAREOPTIONS(1,3). Action: Notify your system programmer.
0000000C (12)	xxxx003E (62)	Meaning: The hyperspace 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 <i>object_name</i> . The value <i>rrrr</i> is the return code from dynamic allocation. The value <i>nnnn</i> is the two-byte reason code from dynamic allocation. See z/OS MVS Programming: Authorized Assembler Services Guide for dynamic allocation return and reason codes. Action: If <i>object_state</i> 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. For parameters that CSRREFR uses to obtain input values, assign values that are acceptable to CSRREFR. For parameters that CSRREFR ignores, assign any value that is valid for the particular parameter's data type.

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

CALL statement	Parameters
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” on page 27](#).

,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” on page 27](#).

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 on page 28](#) 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. CSRREFR Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	Meaning: The operation was successful. Action: Continue normal program execution.
00000008 (8)	00000152 (338)	Meaning: The system could not refresh all of the temporary object within the specified span. Action: Notify your system programmer.
0000000C (12)	xxxx000A (10)	Meaning: There is another service currently executing with the specified ID. Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	Meaning: An I/O error has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx001A (26)	Meaning: The specified range does not include any mapped block of the object. Action: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx0801 (2049)	Meaning: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. Action: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	Meaning: System error — A necessary page table could not be read into real storage. Action: Notify your system programmer.
0000000C (12)	xxxx0805 (2053)	Meaning: System error — A system error of indeterminate origin has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	Meaning: System error — I/O error. Action: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	Meaning: System error — I/O from a previous request has not completed. Action: Notify your system programmer.
0000002C (44)	00000004 (4)	Meaning: Window services have not been defined to your system or the link to the service failed. Action: Notify your system programmer.

CSRSAVE — Save changes made to a permanent object

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

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

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

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

object_id

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

Define *object_id* as character data of length 8.

,offset

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

Define *offset* as integer data of length 4.

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

,span

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

Define *span* as integer data of length 4.

,new_hi_offset

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

Define *new_hi_offset* as integer data of length 4.

,return_code

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

Return codes and reason codes are explained under [“Return codes and reason codes” on page 29](#).

,reason_code

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

Return codes and reason codes are explained under [“Return codes and reason codes” on page 29](#).

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 4 on page 30](#) 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 4. CSRSAVE Return and Reason Codes		
Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	Meaning: The operation was successful. Action: Continue normal program execution.
00000004 (4)	xxxx0807 (2055)	Meaning: Media damage may be present in allocated DASD space. The damage is beyond the currently saved portion of the object. The SAVE operation completed successfully. Action: Notify your system programmer.
00000008 (8)	xxxx0143 (323)	Meaning: You cannot use the SAVE service for a temporary object. Action: Use the scrollout (CSRSCOT) service.
0000000C (12)	xxxx000A (10)	Meaning: There is another service currently executing with the specified ID. Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	Meaning: An I/O error has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx001A (26)	Meaning: The specified range does not encompass any mapped area of the object. Action: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx0801 (2049)	Meaning: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. Action: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	Meaning: System error — I/O driver failure. Action: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	Meaning: System error — A necessary page table could not be read into real storage. Action: Notify your system programmer.
0000000C (12)	xxxx0804 (2052)	Meaning: System error — Catalog update failed. Action: Notify your system programmer.
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. 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 statement	Parameters
CALL CSRSCOT	(object_id ,offset ,span ,return_code ,reason_code)

object_id

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

Define *object_id* as character data of length 8.

,offset

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

Define *offset* as integer data of length 4.

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

,span

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

Define *span* as integer data of length 4.

,return_code

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

Return codes and reason codes are explained under [“Return codes and reason codes” on page 31](#).

,reason_code

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

Return codes and reason codes are explained under [“Return codes and reason codes” on page 31](#).

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 5 on page 32 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 5. CSRSCOT Return and Reason Codes		
Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	Meaning: The operation was successful. Action: Continue normal program execution.
00000004 (4)	xxxx0807 (2055)	Meaning: Media damage may be present in allocated DASD space. The damage is beyond the currently saved portion of the object. The SAVE operation completed successfully. Action: Notify your system programmer.
0000000C (12)	xxxx000A (10)	Meaning: There is another service currently executing with the specified ID. Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	Meaning: An I/O error has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx001A (26)	Meaning: The specified range does not encompass any mapped area of the object. Action: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx0801 (2049)	Meaning: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. Action: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	Meaning: System error — I/O driver failure. Action: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	Meaning: System error — A necessary page table could not be read into real storage. Action: Notify your system programmer.
0000000C (12)	xxxx0804 (2052)	Meaning: System error — Catalog update failed. Action: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	Meaning: System error — I/O error. Action: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	Meaning: System error — I/O from a previous request has not completed. Action: Notify your system programmer.
0000002C (44)	00000004 (4)	Meaning: Window services have not been defined to your system or the link to the service failed. Action: Notify your system programmer.

CSRVIEW — View an object

Call CSRVIEW to:

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

- End a view that you previously created through CSRVIEW or CSREVIEW and unmap the object.

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

The CSREVIEW 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. 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 CSREVIEW 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 statement	Parameters
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”](#) on page 35.

,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”](#) on page 35.

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 6 on page 35](#) 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 6. CSRVIEW Return and Reason Codes		
Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	Meaning: The operation was successful. Action: Continue normal program execution.
00000004 (4)	xxxx0125 (293)	Meaning: The operation was successful. The service could not retain all the data that was in the scroll area, however. Action: Notify your system programmer.
0000000C (12)	xxxx000A (10)	Meaning: There is another service currently executing with the specified ID. Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	Meaning: An I/O error has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx001A (26)	Meaning: The specified range does not encompass any mapped area of the object. Action: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx001C (28)	Meaning: The object cannot be accessed at the current time. Action: Try running your program at a later time. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0040 (64)	Meaning: The specified MAP range would cause the hiperspace data-in-virtual object to be extended such that the installation data space limits would be exceeded. Action: Change the MAP range you have specified or request your system programmer to increase the installation's data space limits.
0000000C (12)	xxxx0801 (2049)	Meaning: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. Action: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	Meaning: System error — I/O driver failure. Action: Notify your system programmer.

Table 6. CSRVIEW Return and Reason Codes (continued)		
Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx0803 (2051)	Meaning: System error — A necessary page table could not be read into real storage. Action: Notify your system programmer.
0000000C (12)	xxx00804 (2052)	Meaning: System error — Catalog update failed. Action: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	Meaning: System error — I/O error. Action: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	Meaning: System error — I/O from a previous request has not completed. Action: Notify your system programmer.
0000002C (44)	00000004 (4)	Meaning: Window services have not been defined to your system or the link to the service failed. Action: Notify your system programmer.

Chapter 4. Window services coding examples

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

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 in [Chapter 4, “Window services coding examples,” on page 37](#):

- [“ADA example” on page 37](#)
- [“C/370 example” on page 41](#)
- [“COBOL example” on page 43](#)
- [“FORTRAN example” on page 46](#)
- [“Pascal example” on page 50](#)
- [“PL/I example” on page 53](#)

ADA example

```

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

with EBCDIC; use EBCDIC;
with System;
with Text_IO;
with Unchecked_Conversion;
with Td_Standard; use Td_Standard;

procedure CRTPAN06 is

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

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

  function Int_To_32 is new Unchecked_Conversion
    (Integer, Integer_32);

  Orig, -- Index to indicate the 'start'
        -- of an array
  Ad, I : Integer; -- Temporary variables
  Voffset, -- Offset passed as parameter
  Voffset2, -- Offset passed as parameter
  Vobjsiz, -- Object size, as parameter
  Vwinsiz, -- Window size, as parameter
  High_Offset, -- Size of object in pages
  New_Hi_Offset, -- New max size of the object
  Return_Code, -- Return code
  Reason_Code : Integer_32; -- Reason code
  Object_Id : Str8; -- Identifying token
  Cscroll : Str3; -- Scroll area YES/NO
  Cobstate : Str3; -- Object state NEW/OLD
  Coptype : Str5; -- Operation type BEGIN/END
  Caccess : Str6; -- Access RANDOM/SEQ
  Cusage : Str6; -- Usage READ/UPDATE

```

```

Cdisp      : Str7;          -- Disposition RETAIN/REPLACE
Csptype    : Str9;          -- Object type DSNNAME/DDNAME/TEMPSPACE
Cobname    : Str7;          -- Object name
K           : constant Integer := 1024; -- One kilo-byte
Pagesize   : constant Integer := 4 * K; -- Page (4K) boundary
Offset     : constant Integer_32 := 0; -- Start of permanent object
Window_Size : constant Integer := 40;  -- Window size in pages
Num_Win_Elem : constant Integer := Window_Size*K; -- Num of 4-byte
-- elements in window
Object_Size : constant Integer := 3*Window_Size; -- Chosen object
-- size in pages
Num_Sp_Elem : constant Integer := (Window_Size+1)*K; -- Num of
-- 4-byte elements in space

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

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

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

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

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

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

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

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

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

  CSRIDAC (Coptype,
           Csptype,
           -- Set up access to the
           -- permanent object and

```

```

        Cobname,                -- request a scroll area
        Cscroll,
        Cobstate,
        Caccess,
        Vobjsiz,
        Object_Id,
        High_Offset,
        Return_Code,
        Reason_Code);

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

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

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

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

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

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

-- You can pass an array slice as a parameter to a non-Ada subprogram,
-- and because the slice is a composite object, the parameter list
-- 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 assembler\
-- 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 (Cotype,                -- Now map a window (the array)
             Object_Id,            -- to the permanent object.
             Voffset,              -- (Actually, CSRVIEW will map the
             Vwinsiz,              -- window to the blocks in the
             Sp(Orig..Num_Sp_Elem), -- scroll area and map the scroll
             Cusage,               -- area to the object.)
             Cdisp,
             Return_Code,
             Reason_Code);

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

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

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

    CSRVIEW (Cotype,                -- End the view in window
             Object_Id,
             Voffset,
```

```

        Vwinsiz,
        Sp(Orig..Num_Sp_Elem),
        Cusage,
        Cdisp,
        Return_Code,
        Reason_Code);

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

CSRVIEW (Coptype,                      -- Now map the same window
         Object_Id,                    -- to different part of the
         Voffset2,                      permanent object.
         Vwinsiz,
         Sp(Orig..Num_Sp_Elem),
         Cusage,
         Cdisp,
         Return_Code,
         Reason_Code);

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

CSRSAVE (Object_Id,                   -- Capture the view in window.
         Voffset2,                    -- Note: this time the permanent
         Vwinsiz,                    -- object is updated with the
         New_Hi_Offset,              -- changes.
         Return_Code,
         Reason_Code);

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

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

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

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

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

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

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

Coptype := "END ";
Csptype := "DDNAME ";

```

```

Cobname  := "CSRDD1 ";
Cscroll  := "YES";
Cobstate := "OLD";
Caccess  := "UPDATE";

CSRIDAC (Cotype,          -- Terminate access to the
         Csptype,         -- permanent object
         Cobname,
         Cscroll,
         Cobstate,
         Caccess,
         Vwinsiz,
         Object_Id,
         High_Offset,
         Return_Code,
         Reason_Code);

end CRTPAN06;

//ADAJOB      JOB                                00000100
//* * * * * 00000500
//* JCL USED TO COMPILE, LINK, AND EXECUTE THE ADA PROGRAM CRTPAN06 00000600
//* THAT USES DATA WINDOW SERVICES                                00000700
//* * * * * 00000800
//*JOBPARM   T=2,L=99                                00050000
//ADACOB1    EXEC PGM=IKJEFT01,DYNAMNBR=133          00055813
//SYSTSPRT   DD SYSOUT=*                             00055913
//SYSTSIN    DD *                                     00056008
//          ALLOC FI(SYSLIB) DS('SYS1.CSSLIB') SHR    00056147
//          EX 'HLQ.SEVGEXE1(ADA)' 'USERID.DWS.ADA' (MAI CRE' 00056251
//          00057008
//ADARUN     EXEC PGM=CRTPAN06,DYNAMNBR=133          00070036
//STEPLIB    DD DISP=SHR,DSN=HLQ.SEVHMOD1            00100051
//          DD DISP=SHR,DSN=USERID.LOAD               00110051
//CSRDD1     DD DSN=USERID.ADA.DWSTEST.DATA,DISP=SHR 00120051
//CONOUT     DD SYSOUT=*,                             00130013
//          DCB=(LRECL=133,RECFM=F)                   00140027

```

C/370 example

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

```

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

```

```

csridac(op_type1, object_type, object_name, scroll_area, object_state,
        access_mode,&object_size,&object_id,&high_offset,&return_code,;
        &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Define a window in a 4K region and initialize */
/* variables for CSRVIEW. Define the window for the */
/* TEMPSPACE and verify the return code and reason code. */
init_mem('0',windows,SIZE);
addr = (int) windows % 4096;
if (addr != 0) view = windows + 4096 - addr;
offset = 0; window_size = 1;
csrvview(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;
csrvview(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;
csrvview(op_type1,&object_id,&offset,&window_size,view,;
        usage, disposition, &return_code, &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* The values in the window should remain to 1's. */
for (i=0; i<4096; i++) {
    if (errflag == TRUE) printf("%d %c ", i, view[i]);
    if (view[i] != '1') errflag = TRUE;
}
/* Refresh the window to 0's. */
offset = 0; window_size = 1;
csrrefr(&object_id, &offset, &window_size,;
        &return_code, &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* The values inside the window should equal to 0's. */
for (i=0; i<4096; i++) {
    if (errflag == TRUE) printf("%d %c ", i, view[i]);
    if (view[i] != 0) errflag = TRUE;
}
/* Terminate the view to the window. */
offset = 0; window_size = 1;
csrvview(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 Hiperpace object. */
csridac(op_type2, object_type, object_name, scroll_area, object_state,
        access_mode,&object_size,&object_id,&high_offset,&return_code,;
        &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Report the status of the test. */
if (errflag) {
    printf("Test failed at line %d\n", __LINE__);
    exit(1);
}
else {
    printf("Test successful : %s\n", __FILE__);
    exit(0);
}
}
/* Functions that will be used in the program. */
/* chk_code will check return code and reason code returned from */
/* the Callable Services. It will report an error if the code(s) */
/* is not equal to 0. */

```

```

int chk_code(long int ret, long int reason, int linenumber)
{
    if (ret != 0)
        printf("return_code = %ld instead of 0 at line %d\n",
               ret, linenumber);
    if (reason != 0)
        printf("reason_code = %ld instead of 0 at line %d\n",
               reason, linenumber);
}
/* init_mem will initialize a block of memory starting at a      */
/* given location to a specified value.                          */
void init_mem(char init_val, char *low_mem, int size)
{
    int i;
    for (i=0; i<size; i++) *(low_mem+i) = init_val;
}
/*
/*-----
/* JCL USED TO COMPILE, LINK, AND, EXECUTE THE C/370 PROGRAM
/*-----
/*
//DPTTST1A JOB 'DPT04P,DPT,?,S=I','DPTTST1',MSGCLASS=H,
//          CLASS=J,NOTIFY=DPTTST1,MSGLEVEL=(1,1)
//CC        EXEC EDCC,INFILE='DPTTST1.DWS.SOURCE(DWS1)',
//          CPARM='NOOPT,SOURCE,NOSEQ,NOMAR',
//          OUTFILE='DPTTST1.DWS.OBJECT(DWS1)'
/*-----
/* LINK STEP
/*-----
//LKED      EXEC PGM=IEWL,PARM='MAP,RMODE=ANY,AMODE=31'
//SYSLIB    DD DSN=CEE.SCEELKED,DISP=SHR
//          DD DSN=SYS1.CSSLIB,DISP=SHR
//OBJECT    DD DSN=DPTTST1.DWS.OBJECT,DISP=SHR
//SYSLIN    DD *
//          ENTRY CEESTART
//          INCLUDE OBJECT(DWS1)
//          NAME DWS1(R)
//SYSLMOD   DD DSN=DPTTST1.DWS.LOAD,DISP=SHR
//SYSPRINT   DD SYSOUT=*
//SYSUT1    DD DSN=SYSUT1,UNIT=SYSDA,DISP=(NEW,DELETE,DELETE),
//          SPACE=(32000,(30,30))
/*-----
/* GO STEP. THIS STEP DEFINES A NAME FOR A PERMANENT OBJECT THAT
/* THE DDNAME OBJECT TYPE WILL REFERENCE.
/*-----
//GO        EXEC PGM=DWS1,REGION=4M
//STEPLIB   DD DSN=CEE.SCEERUN,DISP=SHR
//          DD DSN=DPTTST1.DWS.LOAD,DISP=SHR
//SYSPRINT   DD SYSOUT=*,DCB=(RECFM=VB,LRECL=125,BLKSIZE=6000)
//PLIDUMP    DD SYSOUT=*
//SYSUDUMP   DD SYSOUT=*
//DD1       DD DSN=DPTTST1.DWS.FILE1,DISP=SHR

```

COBOL example

```

IDENTIFICATION DIVISION.
*****
* Program using COBOL to create a 40-page window          *
* aligned on a page boundary. This is done by locating a  *
* page boundary within a 40*4096+4095 byte work area.     *
* The DWS interface validation routine is then called passing *
* the 40 page window.                                     *
*****
PROGRAM-ID. DWSCBSAM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
1   WORKAREA.
2   FILLER PIC X OCCURS 167935 TIMES.
PROCEDURE DIVISION.
    DISPLAY " DWSCBSAM CALLING DWSCB4K "
    CALL "DWSCB4K" USING WORKAREA
    DISPLAY " DWSCBSAM BACK FROM DWSCB4K "
    GOBACK.
-----
IDENTIFICATION DIVISION.
PROGRAM-ID. DWSCB4K.
ENVIRONMENT DIVISION.

```

```

DATA DIVISION.
WORKING-STORAGE SECTION.
1  P POINTER.
1  PR REDEFINES P PIC 9(9) COMP.
1  DUMMY PIC 9(9) COMP.
1  R     PIC 9(9) COMP.
LINKAGE SECTION.
1  INWORK PIC X(167935).
1  WINDOW.
   2 FILLER PIC X(4096) OCCURS 40 TIMES.
PROCEDURE DIVISION USING INWORK.
  SET P TO ADDRESS OF INWORK
  DIVIDE PR BY 4096
  GIVING DUMMY
  REMAINDER R
  IF R NOT EQUAL 0 THEN
  COMPUTE PR = PR + 4096 - R
  SET ADDRESS OF WINDOW TO P
  DISPLAY " DWSCBK4 CALLING DWSCB2 "
  CALL "DWSCB2" USING WINDOW.
  DISPLAY " DWSCBK4 BACK FROM  DWSCB2 "
  GOBACK.

```

```

-----
IDENTIFICATION DIVISION.
PROGRAM-ID. DWSCB2.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
* WINDOW SIZE CHOSEN TO BE 40 PAGES
1  NWINPG PIC 9(9) COMP VALUE 40.
1  NWINEL PIC 9(9) COMP.
1  NWLAST PIC 9(9) COMP.
1  NOBJPG PIC 9(9) COMP.
* WINDOWS WILL BEGIN ORIGIN-ING AT OFFSET 0 IN DATA OBJECT
1  WINOFF PIC 9(9) COMP VALUE 0.
1  RETRN1 PIC 9(9) COMP.
1  REASON PIC 9(9) COMP.
1  NEWOFF PIC 9(9) COMP.
1  OBSIZ  PIC 9(9) COMP.
1  TOKEN  PIC X(8).
1  K      PIC 9(9) COMP.
LINKAGE SECTION.
1  WINDOW.
   2 FILLER PIC X(4096) OCCURS 40 TIMES.
1  WINDOW-ARRAY REDEFINES WINDOW.
   2 A PIC S9(8) COMP OCCURS 40960 TIMES.
PROCEDURE DIVISION USING WINDOW.
  DISPLAY "Begin Data Windowing Services Interface Validation"
* WINDOW COMPOSED OF 4-BYTE ELEMENTS
  COMPUTE NWINEL = 1024 * NWINPG.
* WINDOW MAY NOT BEGIN AT ARRAY ELEMENT 1, SO LEAVE ROOM
  COMPUTE NWLAST = 1024 * NWINPG + 1023
* IN THE FOLLOWING, ARBITRARILY SET OBJECT SIZE = 3 WINDOWS WORTH
  COMPUTE NOBJPG = 3 * NWINPG
* SET UP ACCESS TO A HIPERSPACE OBJECT
  CALL "CSRIDAC" USING
    BY CONTENT
    "BEGIN",
    "TEMPSPACE",
    "MY FIRST HIPERSPACE",
    "YES",
    "NEW",
    "UPDATE",
    BY REFERENCE
    NOBJPG,
    TOKEN,
    OBSIZ,
    RETRN1,
    REASON
* PUT SOME DATA INTO THE WINDOW AREA
  MOVE ALL "DATA" TO WINDOW
* NOW VIEW SOMETHING IN THE WINDOW
  CALL "CSRVIEW" USING
    BY CONTENT
    "BEGIN",
    BY REFERENCE
    TOKEN,
    WINOFF,
    NWINPG,
    WINDOW,
    BY CONTENT

```



```

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

```

```

        WINDOW,
        BY CONTENT
        "RANDOM",
        "REPLACE",
        BY REFERENCE
        RETRN1,
        REASON
* REFRESH THE DATA IN THE WINDOW FOR THIS VIEW
  CALL "CSRREFR" USING
        TOKEN,
        WINOFF,
        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,
//  MSGCLASS=H,NOTIFY=AAUSER,
//  MSGLEVEL=(1,1),CLASS=7
//LKED EXEC PGM=IEWL,PARM='SIZE=(1024K,512K),LIST,XREF,LET,MAP',
//  REGION=1024K
//SYSLIN DD DDNAME=SYSIN
//SYSLMOD DD DSN=AAUSER.USER.LOAD(CRTCON01),DISP=SHR
//SYSLIB DD DSN=CEE.SCEELED,DISP=SHR
//*
//* FF310.OBJ HOLDS OBJECT CODE FROM THE COMPILE
//*
//MYLIB DD DSN=AAUSER.FF310.OBJ,DISP=SHR
//*
//* THE CSR STUBS ARE IN SYS1.CSSLIB
//*
//INLIB DD DSN=SYS1.CSSLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
INCLUDE MYLIB(DWSCBSAM,DWSCB4K,DWSCB2)
LIBRARY INLIB(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC)
NAME CRTCON01(R)
00010000
00020000
00030000
00040000
00080000
00090000
00110000
00120000
00140000
00150100
00150200
00150300
00151000
00151100
00151200
00151300
00152000
00170000
00230000
00231000
00240000
00250000

```

FORTRAN example

```

*****
*
*
*

```

```

*      FORTRAN EXAMPLE.  THE FORTRAN EXAMPLE IS FOLLOWED BY AN      *
*      ASSEMBLER PROGRAM CALLED ADDR.  YOU MUST LINKEDIT THIS      *
*      ASSEMBLER PROGRAM WITH THE FORTRAN PROGRAM OBJECT          *
*      CODE AND THE CSR STUBS.  THE ASSEMBLER PROGRAM ENSURES      *
*      THAT YOUR WINDOW IS ALIGNED ON A 4K BOUNDARY .              *
*                                                                    *
*****
@PROCESS DC(WINCOM)
PROGRAM CRTFON01
C
C      Test Program for Data Window Services
C
C      Window size chosen to be 40 pages
C      PARAMETER (NWINPG = 40)
C      Window composed of 4-byte elements
C      PARAMETER (NWINEL = 1024*NWINPG)
C      Window may not begin at array element 1, so leave room
C      PARAMETER (NWLAST = 1024*NWINPG+1023)
C      In the following, arbitrarily set object size = 3 windows worth
C      PARAMETER (NOBJPG = 3*NWINPG)
C      Windows will begin origin-ing at offset 0 in data object
C      INTEGER WINOFF
C      PARAMETER (WINOFF = 0)
C
C      INTEGER RETRN1, REASON, HIOFF, NEWOFF, OBSIZ, OFF
C      INTEGER ADDR, PAGE, A
C      INTEGER JUNK /-1599029040/
C      REAL*8 TOKEN
C      COMMON /WINCOM/ A(NWLAST)
C
C
C      WRITE (6, 91)
91 FORMAT('1*** Begin Data Windowing Services Interface Validation')
C
C      Set up access to a Hiperspace object
C      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"
C      PAGE = ADDR(A(1))
C      PAGE = MOD(PAGE, 4096)
C      IF (PAGE .NE. 0) PAGE = (4096 - PAGE) / 4
C      PAGE = PAGE + 1
C
C      Put data into the window
C      DO 100 K = 1, NWINEL
C          A(K+PAGE-1) = JUNK
100 CONTINUE
C
C      Now view data in the window
C      CALL CSRVIEW('BEGIN',
*                TOKEN,
*                WINOFF,
*                NWINPG,
*                A(PAGE),
*                'RANDOM',
*                'REPLACE',
*                RETRN1,
*                REASON )
C
C      Calculate a value in the window area
C      DO 101 K = 1, NWINEL
C          A(K+PAGE-1) = K
101 CONTINUE
C
C      Capture the view in the window
C      CALL CSRSCOT( TOKEN,
*                WINOFF,
*                NWINPG,
*                RETRN1,
*                REASON )
C
C      End the view in the window

```

```

      CALL CSRVIEW('END ',
*          TOKEN,
*          WINOFF,
*          NWINPG,
*          A(PAGE),
*          'RANDOM',
*          'RETAIN ',
*          RETRN1,
*          REASON )
C
C      Now view other data (2nd window's worth of data) in window
      CALL CSRVIEW('BEGIN',
*          TOKEN,
*          WINOFF + NWINPG,
*          NWINPG,
*          A(PAGE),
*          'RANDOM',
*          'REPLACE',
*          RETRN1,
*          REASON )
C
C      Calculate a new value in the window
      DO 102 K = 1, NWINEL
        A(K+PAGE-1) = -K
102  CONTINUE
C
C      Capture the view in the window
      CALL CSRSCOT( TOKEN,
*          WINOFF + NWINPG,
*          NWINPG,
*          RETRN1,
*          REASON )
C
C      Now end the current view in window
      CALL CSRVIEW('END ',
*          TOKEN,
*          WINOFF + NWINPG,
*          NWINPG,
*          A(PAGE),
*          'RANDOM',
*          'RETAIN ',
*          RETRN1,
*          REASON )
C
C      Now go back to the first view in the window
      CALL CSRVIEW('BEGIN',
*          TOKEN,
*          WINOFF,
*          NWINPG,
*          A(PAGE),
*          'RANDOM',
*          'REPLACE',
*          RETRN1,
*          REASON )
C
C      Refresh the data in the window for this view
      CALL CSRREFR( TOKEN,
*          WINOFF,
*          NWINPG,
*          RETRN1,
*          REASON )
C
C      Now end the view in the window
      CALL CSRVIEW('END ',
*          TOKEN,
*          WINOFF,
*          NWINPG,
*          A(PAGE),
*          'RANDOM',
*          'RETAIN ',
*          RETRN1,
*          REASON )
C
C      Terminate access to the Hiperspace object
      CALL CSRIDAC('END ',
*          'TEMPSPACE',
*          'MY FIRST HIPERSPACE ENDS HERE ',
*          'YES',
*          'NEW',
*          'UPDATE',
*          NOBJPG,
*          TOKEN,

```

```

*          OBSIZ,
*          RETRN1,
*          REASON )
C
    STOP
    END
*****
*
*          THIS ASSEMBLER PROGRAM ENSURES THAT YOUR WINDOW IS ALIGNED
*          ON A 4K BOUNDARY. ASSEMBLE THIS PROGRAM AND LINKEDIT THE
*          OBJECT CODE WITH THE FORTRAN CODE AND THE CSR STUBS.
*
*****
ADDR    TITLE 'LOC/ADDR Function for Fortran'
*
*          Calling Sequence:
*
*          INTEGER ADDR
*          - - -
*          L = LOC(x)
*          L = ADDR(x)
*
*          Returns address of "x" in R0, with high-order bit set to zero
*
ADDR    CSECT
        ENTRY LOC
LOC      EQU    *
        USING  *,15
        L      0,0(,1)          Get pointer to x
        N      0,MASK           Set sign bit to 0
        BR     14               Return
MASK     DC     A(X'7FFFFFFF')  Mask with high-order bit 0
        END
*****
*          JCL TO COMPILE AND LINKEDIT THE ASSEMBLER PROGRAM, THE
*          FORTRAN PROGRAM, AND THE STUBS.
*
*****
//FORTJOB JOB                                00255013
//*                                           00003100
//*                                           00003100
//* Compile and linkedit for FORTRAN        00003100
//*                                           00003100
//*                                           00003100
//VSF2CL PROC  FVPGM=FORTVS2,FVREGN=2100K,FVPDECK=NODECK, 00001000
//          FVPOLST=NOLIST,FVPOPT=0,FVTERM='SYSOUT=A',      00002000
//          PGMNAME=MAIN,PGMLIB='&&GOSET',FVLNSPC='3200,(25,6)' 00003000
//*                                           00003100
//*          PARAMETER  DEFAULT-VALUE  USAGE                00003900
//*                                           00004000
//*          FVPGM      FORTVS2        COMPILER NAME          00005000
//*          FVREGN      2100K          FORT-STEP REGION       00006000
//*          FVPDECK     NODECK         COMPILER DECK OPTION   00007000
//*          FVPOLST     NOLIST         COMPILER LIST OPTION   00008000
//*          FVPOPT      0              COMPILER OPTIMIZATION  00009000
//*          FVTERM      SYSOUT=A       FORT.SYSTERM OPERAND   00010000
//*          FVLNSPC     3200,(25,6)    FORT.SYSLIN SPACE      00011000
//*          PGMLIB      &&GOSET        LKED.SYSLMOD DSNAME     00012000
//*          PGMNAME     MAIN           LKED.SYSLMOD MEMBER NAME 00013000
//*                                           00014000
//FORT EXEC PGM=&FVPGM,REGION=&FVREGN,COND=(4,LT),          00015000
//          PARM='&FVPDECK,&FVPOLST,OPT(&FVPOPT)'          00016000
//STEPLIB DD DSN=HLLDS.FORT230.VSF2COMP,DISP=SHR           00017000
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=3429                     00018000
//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.SCEELKED,DISP=SHR                        00026000
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(200,20))                00027000
//SYSLMOD DD DSN=&PGMLIB.(&PGMNAME),DISP=(,PASS),UNIT=SYSDA, 00028000
//          SPACE=(TRK,(10,10,1),RLSE)                       00029000
//SYSLIN DD DSN=&&LOADSET,DISP=(OLD,DELETE)                  00030000
//          DD DDNAME=SYSIN                                   00040000
//PEND
//          EXEC VSF2CL,FVTERM='SYSOUT=H',
//          PGMNAME=CRTFON01,PGMLIB='WINDOW.USER.LOAD'      00003000

```

Pascal Example

```
//FORT.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTFON01),DISP=SHR
//LKED.SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR 00026000
//LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,DISP=SHR,UNIT=3380,
// VOL=SER=VM2TSO
//LKED.SYSIN DD *
LIBRARY IN(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC,ADDR)
NAME CRTFON01(R)
/*
//* The CSR stubs are available in SYS1.CSSLIB.
//* The object code for the ADDR routine is in
//* TEST.OBJ
/*
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR
// DD DSN=WINDOW.TEST.OBJ,DISP=SHR
/*
/*
*****
* JCL TO EXECUTE THE FORTRAN PROGRAM. *
* *
*****
//FON01 JOB MSGLEVEL=(1,1)
//VSF2G PROC GOPGM=MAIN,GOREGN=100K, 00001000
// GOF5DD='DDNAME=SYSIN', 00002000
// GOF6DD='SYSOUT=A', 00003000
// GOF7DD='SYSOUT=B' 00004000
/* 00005000
/* PARAMETER DEFAULT-VALUE USAGE 00007000
/* 00008000
/* GOPGM MAIN PROGRAM NAME 00009000
/* GOREGN 100K GO-STEP REGION 00010000
/* GOF5DD DDNAME=SYSIN GO.FT05F001 DD OPERAND 00011000
/* GOF6DD SYSOUT=A GO.FT06F001 DD OPERAND 00012000
/* GOF7DD SYSOUT=B GO.FT07F001 DD OPERAND 00013000
/* 00014000
/* 00015000
//GO EXEC PGM=&GOPGM,REGION=&GOREGN,COND=(4,LT) 00016000
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR 00017000
//FT05F001 DD &GOF5DD 00018000
//FT06F001 DD &GOF6DD 00019000
//FT07F001 DD &GOF7DD 00020000
// PEND
//GO EXEC VSF2G,GOPGM=CRTFON01,GOREGN=999K
//GO.STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR 00017000
// DD DSN=WINDOW.USER.LOAD,DISP=SHR,VOL=SER=VM2TSO,UNIT=3380
```

Pascal example

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

```

VOFSET2, (* Offset passed as parameter *)
VOBJSIZ, (* Object size, as parameter *)
VWINSIZ, (* Window Size, as parameter *)
HIGH_OFFSET, (* Size of object in pages *)
NEW_HI_OFFSET, (* New max size of the object *)
RETURN_CODE, (* Return code *)
REASON_CODE : INTEGER; (* Reason code *)
OBJECT_ID : REAL; (* Identifying token *)
CSCROLL : STR3; (* Scroll area YES/NO *)
COBSTATE : STR3; (* Object state NEW/OLD *)
COPTYPE : STR5; (* Operation type BEGIN/END *)
CACCESS : STR6; (* Access RANDOM/SEQ *)
CUSAGE : STR6; (* Usage READ/UPDATE *)
CDISP : STR7; (* Disposition RETAIN/REPLACE *)
CSPTYPE : STR9; (* Object type DSNNAME/DDNAME/TEMPSPACE *)
COBNAME : STR44; (* Object name *)

procedure CSRIDAC ( var OP_TYPE : STR5;
var OBJECT_TYPE : STR9;
var OBJECT_NAME : STR44;
var SCROLL_AREA : STR3;
var OBJECT_STATE : STR3;
var ACCESS_MODE : STR6;
var VOBJSIZ : INTEGER;
var OBJECT_ID : REAL;
var HIGH_OFFSET : INTEGER;
var RETURN_CODE : INTEGER;
var REASON_CODE : INTEGER); FORTRAN;

procedure CSRVIEW ( var OP_TYPE : STR5;
var OBJECT_ID : REAL;
var OFFSET : INTEGER;
var WINDOW_SIZE : INTEGER;
var WINDOW_NAME : INTEGER;
var USAGE : STR6;
var DISPOSITION : STR7;
var RETURN_CODE : INTEGER;
var REASON_CODE : INTEGER); FORTRAN;

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

procedure CSRSAVE ( var OBJECT_ID : REAL;
var OFFSET : INTEGER;
var SPAN : INTEGER;
var NEW_HI_OFFSET : INTEGER;
var RETURN_CODE : INTEGER;
var REASON_CODE : INTEGER ); FORTRAN;

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

begin
  TERMOUT(OUTPUT); (* Output to terminal *)
  WRITELN ('<< Begin Data Windowing Services Interface Validation >>');
  WRITELN;
  VOBJSIZ := OBJECT_SIZE; (* Set object size variable *)
  VOFFSET := OFFSET; (* Set offset variable to 0 *)
  VWINSIZ := WINDOW_SIZE; (* Set window size variable *)
  VOFSET2 := OFFSET+WINDOW_SIZE; (* Set offset variable to 0 *)
  COPTYPE := 'BEGIN' ;
  CSPTYPE := 'DDNAME' ;
  COBNAME := 'CSRDD1' ;
  CSCROLL := 'YES' ;
  COBSTATE := 'NEW' ;
  CACCESS := 'UPDATE' ;
  CSRIDAC (COPTYPE, (* Set up access to a *)
           CSPTYPE, (* hiperspace object *)
           COBNAME,
           CSCROLL,
           COBSTATE,
           CACCESS,
           VOBJSIZ,
           OBJECT_ID,
           HIGH_OFFSET,
           RETURN_CODE,
           REASON_CODE);
  NEW(SP); (* Allocate space *)
  AD := ADDR(SP@); (* or ORD(SP) *) (* Get address of space *)
  ORIG := AD mod PAGESIZE; (* See where space is in page *)
  if ORIG <> 0 then (* If not on page boundary *)
    ORIG := PAGESIZE-ORIG; (* then locate page boundary *)

```

Pascal Example

```

for I := 0 to NUM_WIN_ELEM-1 do      (* Put data into window *)
    SP@[4*I+ORIG] := 999999;          (* area *)
COPTYPE := 'BEGIN' ;
CUSAGE := 'RANDOM' ;
CDISP := 'REPLACE' ;
CSRVIEW (COPTYPE,                     (* Now view data in 1st *)
          OBJECT_ID,                  (* window *)
          VOFFSET,
          VWINSIZ,
          SP@[ORIG],
          CUSAGE,
          CDISP,
          RETURN_CODE,
          REASON_CODE);
for I := 0 to NUM_WIN_ELEM-1 do      (* Calculate a value in 1st *)
    SP@[4*I+ORIG] := I+1;            (* window *)
CSRSCOT( OBJECT_ID,                  (* Capture the view in 1st *)
          VOFFSET,                   (* window *)
          VWINSIZ,
          RETURN_CODE,
          REASON_CODE);
COPTYPE := 'END' ;
CUSAGE := 'RANDOM' ;
CDISP := 'RETAIN' ;
CSRVIEW (COPTYPE,                     (* End the view in 1st window *)
          OBJECT_ID,
          VOFFSET,
          VWINSIZ,
          SP@[ORIG],
          CUSAGE,
          CDISP,
          RETURN_CODE,
          REASON_CODE);
COPTYPE := 'BEGIN' ;
CUSAGE := 'RANDOM' ;
CDISP := 'REPLACE' ;
CSRVIEW (COPTYPE,                     (* Now view other data in the *)
          OBJECT_ID,                 (* 2nd window *)
          VOFSET2,
          VWINSIZ,
          SP@[ORIG],
          CUSAGE,
          CDISP,
          RETURN_CODE,
          REASON_CODE);
for I := 0 to NUM_WIN_ELEM-1 do      (* Calculate a new value in *)
    SP@[4*I+ORIG] := I-101;          (* the window *)
CSRSAVE (OBJECT_ID,
          VOFSET2,
          VWINSIZ,
          NEW_HI_OFFSET,
          RETURN_CODE,
          REASON_CODE);
COPTYPE := 'END' ;
CUSAGE := 'RANDOM' ;
CDISP := 'RETAIN' ;
CSRVIEW (COPTYPE,                     (* End the current view in *)
          OBJECT_ID,                 (* window *)
          VOFSET2,
          VWINSIZ,
          SP@[ORIG],
          CUSAGE,
          CDISP,
          RETURN_CODE,
          REASON_CODE);
COPTYPE := 'BEGIN' ;
CUSAGE := 'RANDOM' ;
CDISP := 'REPLACE' ;
CSRVIEW (COPTYPE,                     (* Now go back to the view in *)
          OBJECT_ID,                 (* the 1st window *)
          VOFFSET,
          VWINSIZ,
          SP@[ORIG],
          CUSAGE,
          CDISP,
          RETURN_CODE,
          REASON_CODE);
CSRREFR (OBJECT_ID,                  (* Refresh the data in 1st *)
          VOFFSET,                   (* window *)
          VWINSIZ,
          RETURN_CODE,
          REASON_CODE);

```



```

COPTYPE  := 'END' ;
CUSAGE   := 'RANDOM' ;
CDISP    := 'RETAIN' ;
CSRVIEW  (COPTYPE,                                (* End the view in 1st window *)
          OBJECT_ID,
          VOFFSET,
          VWINSIZ,
          SP@[ORIG],
          CUSAGE,
          CDISP,
          RETURN_CODE,
          REASON_CODE);
COPTYPE  := 'END' ;
CSPTYPE  := 'DDNAME' ;
COBNAME  := 'CSRDD1' ;
CSCROLL  := 'YES' ;
COBSTATE := 'NEW' ;
CACCESS  := 'UPDATE' ;
CSRIDAC  (COPTYPE,                                (* Terminate access to the *)
          CSPTYPE,                                (* Hiperspace object *)
          COBNAME,
          CSCROLL,
          COBSTATE,
          CACCESS,
          VWINSIZ,
          OBJECT_ID,
          HIGH_OFFSET,
          RETURN_CODE,
          REASON_CODE);
end.
*****
*                                                                 *
*          JCL to compile and linkedit                          *
*                                                                 *
*****
//PASC1JOB JOB                                00010005
//GO      EXEC PAS22CL                        00050000
//*                                              00050102
//*      Compile and linkedit for PASCAL        00050202
//*                                              00050302
//PASC.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTPAN06),DISP=SHR 00060006
//LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,DISP=SHR,UNIT=3380, 00560000
// VOL=SER=VM2TSO 00570000
//LKED.SYSIN DD * 00580000
LIBRARY IN(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC) 00590000
NAME CRTPAN06(R) 00600006
/* 00610000
//*      SYS1.CSSLIB is the source of the CSR stubs 00620002
//* 00650002
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR 00690000
*****
*                                                                 *
*          JCL to execute. A DD statement, CSRDD1, is needed to define *
*          the permanent object which already exists.                *
*                                                                 *
*****
//PASC2JOB JOB MSGLEVEL=(1,1) 00010000
//GO      EXEC PGM=CRTPAN06 00020002
//STEPLIB DD DSN=WINDOW.PASCAL22.LINKLIB, 00030000
// DISP=SHR,UNIT=3380, 00040000
// VOL=SER=VM2TSO 00050000
// DD DSN=WINDOW.USER.LOAD, 00060000
// DISP=SHR,UNIT=3380, 00070000
// VOL=SER=VM2TSO 00080000
//CSRDD1 DD DSN=DIV.TESTDS01,DISP=SHR
//OUTPUT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=133) 00090000
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=133) 00100000

```

PL/I example

```

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

```

```

CRTPLN3: PROCEDURE OPTIONS (MAIN);
DCL
(
  K INIT(1024),          /* ONE KILO-BYTE          */
  PAGESIZE INIT(4096),   /* 4K PAGE BOUNDARY       */
  OFFSET INIT(0),        /* WINDOWS STARTS         */
  WINDOW_SIZE INIT(20),  /* WINDOW SIZE IN PAGES   */
  NUM_WIN_ELEM INIT (20480), /* NUM OF 4-BYTE ELEMENTS */
  OBJECT_SIZE INIT (60)) /* CHOSEN OBJECT SIZE IN PGS */
  FIXED BIN(31);
DCL
/* 32767 IS UPPER LIMIT FOR ARRAY BOUND.
S(32767) BIN(31) FIXED BASED(SP);  /* DEFINE WORD ALIGNED SPACE */
DCL SP PTR;
DCL
(
  ORIG,                  /* START ADDRESS OF WINDOW */
  AD, I,                  /* TEMPORARY VARIABLES     */
  HIGH_OFFSET,           /* SIZE OF OBJECT IN PAGES */
  NEW_HI_OFFSET,         /* NEW MAX SIZE OF THE OBJECT */
  RETURN_CODE,           /* RETURN CODE              */
  REASON_CODE) FIXED BIN(31); /* REASON CODE              */
DCL
  OBJECT_ID CHAR(8);      /* IDENTIFYING TOKEN       */
/*****
DCL CSRIDAC ENTRY(CHAR(5), /* OP_TYPE
CHAR(9),                /* OBJECT_TYPE
CHAR(44),               /* OBJECT_NAME
CHAR(3),                /* SCROLL_AREA
CHAR(3),                /* OBJECT_STATE
CHAR(6),                /* ACCESS_MODE
FIXED BIN(31),          /* OBJECT_SIZE
CHAR(8),                /* OBJECT_ID
FIXED BIN(31),          /* HIGH_OFFSET
FIXED BIN(31),          /* RETURN_CODE
FIXED BIN(31) )        /* REASON_CODE
  OPTIONS(ASSEMBLER);
DCL CSRVIEW ENTRY(CHAR(5), /* OP_TYPE
CHAR(8),                /* OBJECT_ID
FIXED BIN(31),          /* OFFSET
FIXED BIN(31),          /* WINDOW_SIZE
FIXED BIN(31),          /* WINDOW_NAME
CHAR(6),                /* USAGE
CHAR(7),                /* DISPOSITION
FIXED BIN(31),          /* RETURN_CODE
FIXED BIN(31) )        /* REASON_CODE
  OPTIONS(ASSEMBLER);
DCL CSRSCOT ENTRY(CHAR(8), /* OBJECT_ID
FIXED BIN(31),          /* OFFSET
FIXED BIN(31),          /* SPAN
FIXED BIN(31),          /* RETURN_CODE
FIXED BIN(31) )        /* REASON_CODE
  OPTIONS(ASSEMBLER);
DCL CSRSAVE ENTRY(CHAR(8), /* OBJECT_ID
FIXED BIN(31),          /* OFFSET
FIXED BIN(31),          /* SPAN
FIXED BIN(31),          /* NEW_HI_OFFSET
FIXED BIN(31),          /* RETURN_CODE
FIXED BIN(31) )        /* REASON_CODE
  OPTIONS(ASSEMBLER);
DCL CSRREFR ENTRY(CHAR(8), /* OBJECT_ID
FIXED BIN(31),          /* OFFSET
FIXED BIN(31),          /* SPAN
FIXED BIN(31),          /* RETURN_CODE
FIXED BIN(31) )        /* REASON_CODE
  OPTIONS(ASSEMBLER);

```

```

/***** CSR00830
CSR00840
CSR00850
CSR00860
CSR00870
CSR00880
CSR00890
CSR00900
CSR00910
CSR00920
CSR00930
CSR00940
CSR00950
CSR00960
CSR00970
CSR00980
CSR00990
CSR01000
CSR01010
CSR01020
CSR01030
CSR01040
CSR01050
CSR01060
CSR01070
CSR01080
CSR01090
CSR01100
CSR01110
CSR01120
CSR01130
CSR01140
CSR01150
CSR01160
CSR01170
CSR01180
CSR01190
CSR01200
CSR01210
CSR01220
CSR01230
CSR01240
CSR01250
CSR01260
CSR01270
CSR01280
CSR01290
CSR01300
CSR01310
CSR01320
CSR01330
CSR01340
CSR01350
CSR01360
CSR01370
CSR01380
CSR01390
CSR01400
CSR01410
CSR01420
CSR01430
CSR01440
CSR01450
CSR01460
CSR01470
CSR01480
CSR01490
CSR01500
CSR01510
CSR01520
CSR01530
CSR01540
CSR01550
CSR01560
CSR01570
CSR01580
CSR01590
CSR01600
CSR01610
CSR01620
CSR01630
CSR01640
*****/

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

CALL
  CSRIDAC ('BEGIN',
           'TEMPSPACE',
           'MY FIRST HIPERSPACE',
           'YES',
           'NEW',
           'UPDATE',
           OBJECT_SIZE,
           OBJECT_ID,
           HIGH_OFFSET,
           RETURN_CODE,
           REASON_CODE);

/* SET UP ACCESS TO A HIPER-
/* SPACE OBJECT */

ALLOC S;
AD = UNSPEC(SP);
ORIG = MOD(AD,PAGESIZE);
IF ORIG ^= 0 THEN
  ORIG = (PAGESIZE-ORIG) / 4;
ORIG = ORIG + 1;

/* ALLOCATE SPACE
/* GET ADDRESS OF SPACE
/* SEE WHERE SPACE IS IN PAGE
/* IF NOT ON PAGE BOUNDARY
/* THEN LOCATE PAGE BOUNDARY

DO I = 1 TO NUM_WIN_ELEM;
  S(I+ORIG-1) = 99;
/* PUT SOME DATA INTO WINDOW
/* AREA

END;

CALL
  CSRVIEW ('BEGIN',
           OBJECT_ID,
           OFFSET,
           WINDOW_SIZE,
           S(ORIG),
           'RANDOM',
           'REPLACE',
           RETURN_CODE,
           REASON_CODE);

/* NOW VIEW DATA IN FIRST
/* WINDOW

DO I = 1 TO NUM_WIN_ELEM;
  S(I+ORIG-1) = I+1;
/* CALCULATE VALUE IN 1ST
/* WINDOW

END;

CALL
  CSRSCOT( OBJECT_ID,
           OFFSET,
           WINDOW_SIZE,
           RETURN_CODE,
           REASON_CODE);

/* CAPTURE THE VIEW IN 1ST
/* WINDOW

CALL
  CSRVIEW ('END ',
           OBJECT_ID,
           OFFSET,
           WINDOW_SIZE,
           S(ORIG),
           'RANDOM',
           'RETAIN ',
           RETURN_CODE,
           REASON_CODE);

/* END THE VIEW IN 1ST WINDOW

CALL
  CSRVIEW ('BEGIN',
           OBJECT_ID,
           OFFSET+WINDOW_SIZE,
           WINDOW_SIZE,
           S(ORIG),
           'RANDOM',
           'REPLACE',
           RETURN_CODE,
           REASON_CODE);

/* NOW VIEW OTHER DATA IN
/* 2ND WINDOW

DO I = 1 TO NUM_WIN_ELEM;
  S(I+ORIG-1) = I-101;
/* CALCULATE NEW VALUE IN
/* WINDOW

END;

CALL
  CSRSCOT( OBJECT_ID,
           OFFSET+WINDOW_SIZE,

```

```

        WINDOW_SIZE,                CSR01650
        RETURN_CODE,                CSR01670
        REASON_CODE);               CSR01680
                                     CSR01690
CALL                                     CSR01700
CSRVIEW ('END ',                    /* END THE CURRENT VIEW IN */ CSR01710
        OBJECT_ID,                  /* WINDOW */ CSR01720
        OFFSET+WINDOW_SIZE,        CSR01730
        WINDOW_SIZE,               CSR01740
        S(ORIG),                   CSR01750
        'RANDOM',                   CSR01760
        'RETAIN ',                 CSR01770
        RETURN_CODE,               CSR01780
        REASON_CODE);              CSR01790
                                     CSR01800
CALL                                     CSR01810
CSRVIEW ('BEGIN',                  /* NOW GO BACK TO THE VIEW IN */ CSR01820
        OBJECT_ID,                  /* THE 1ST WINDOW */ CSR01830
        OFFSET,                    CSR01840
        WINDOW_SIZE,               CSR01850
        S(ORIG),                   CSR01860
        'RANDOM',                   CSR01870
        'REPLACE',                 CSR01880
        RETURN_CODE,               CSR01890
        REASON_CODE);              CSR01900
                                     CSR01910
CALL                                     CSR01920
CSRREFR (OBJECT_ID,                /* REFRESH THE DATA IN 1ST */ CSR01930
        OFFSET,                    /* WINDOW */ CSR01940
        WINDOW_SIZE,               CSR01950
        RETURN_CODE,               CSR01960
        REASON_CODE);              CSR01970
                                     CSR01980
CALL                                     CSR01990
CSRVIEW ('END ',                    /* END THE VIEW IN 1ST WINDOW */ CSR02000
        OBJECT_ID,                  CSR02010
        OFFSET,                    CSR02020
        WINDOW_SIZE,               CSR02030
        S(ORIG),                   CSR02040
        'RANDOM',                   CSR02050
        'RETAIN ',                 CSR02060
        RETURN_CODE,               CSR02070
        REASON_CODE);              CSR02080
                                     CSR02090
CALL                                     CSR02100
CSRIDAC ('END ',                    /* TERMINATE ACCESS TO THE */ CSR02110
        'TEMPSPACE',                /* HIPERSPACE OBJECT */ CSR02120
        'MY FIRST HIPERSPACE ENDS HERE ', CSR02130
        'YES',                      CSR02140
        'NEW',                      CSR02150
        'UPDATE',                   CSR02160
        WINDOW_SIZE,               CSR02170
        OBJECT_ID,                  CSR02180
        HIGH_OFFSET,               CSR02190
        RETURN_CODE,               CSR02200
        REASON_CODE);              CSR02210
                                     CSR02220
FREE S;                             CSR02230
END CRTPLN3;                         CSR02260
*****
*                                     *
*                                     *
*      JCL TO COMPILE AND LINKEDIT PL/I PROGRAM.      *
*                                     *
*                                     *
*                                     *
*****
//PLIJOB   JOB                                00010007
//*                                00041001
//*  PL/I Compile and Linkedit      00042001
//*                                00043001
//*  Change all CRTPLNx to CRTPLNy  00044001
//*                                00045001
//GO      EXEC PLIXCL                      00050000
//PLI.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTPLN3),DISP=SHR 00060008
//LKED.SYSMOD DD DSN=WINDOW.USER.LOAD,UNIT=3380,VOL=SER=VM2TSO, 00070000
// DISP=SHR                             00080000
//LKED.SYSIN DD *                         00090000
//  LIBRARY IN(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC) 00100001
//  NAME CRTPLN3(R)                      00110008
/*                                00120000
//*                                00121001

```

```

//*      SYS1.CSSLIB is source of CSR stubs      00130001
//*      00190000
//LKED.IN      DD DSN=SYS1.CSSLIB,DISP=SHR      00200000
*****
*
*
*      JCL TO EXECUTE.
*
*
*
*****
//PLIRUN      JOB MSGLEVEL=(1,1)      00010000
//*
//*      EXECUTE A PL/I TESTCASE      00011001
//*      00012001
//*      00013001
//GO          EXEC PGM=CRTPLN3      00020000
//STEPLIB DD   DSN=WINDOW.USER.LOAD,DISP=SHR,      00030000
// UNIT=3380,VOL=SER=VM2TSO      00040000
//SYSLIB DD   DSN=CEE.SCEERUN,DISP=SHR      00050000
//SYSABEND DD   SYSOUT=*      00070000
//SYSLOUT DD   SYSOUT=*      00080000
//SYSPRINT DD   SYSOUT=*      00090000

```

Part 2. Reference pattern services

Chapter 5. Introduction to reference pattern services

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

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

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

Two reference pattern services are available through program CALLs:

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

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

Although reference pattern services can be used for data structures other than arrays, for simplicity, examples in Chapter 5, "Introduction to reference pattern services," on page 61 and Chapter 6, "Using reference pattern services," on page 65 use the services with arrays.

How does the system manage data?

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

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

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

An example of how the system manages data in an array

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

- | | | | |
|----------|----------|----------|----------|
| $A(1,1)$ | $A(1,2)$ | $A(1,3)$ | $A(1,4)$ |
| $A(2,1)$ | $A(2,2)$ | $A(2,3)$ | $A(2,4)$ |
| $A(3,1)$ | $A(3,2)$ | $A(3,3)$ | $A(3,4)$ |

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

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

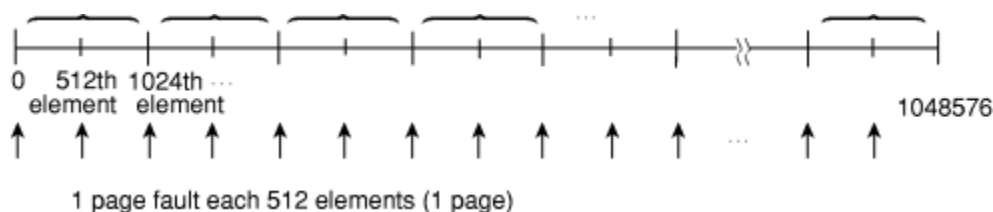
Examples in [Chapter 5, "Introduction to reference pattern services,"](#) on page 61 and [Chapter 6, "Using reference pattern services,"](#) on page 65 depict data as a horizontal string. The elements in the arrays, therefore, would look like the following:

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.

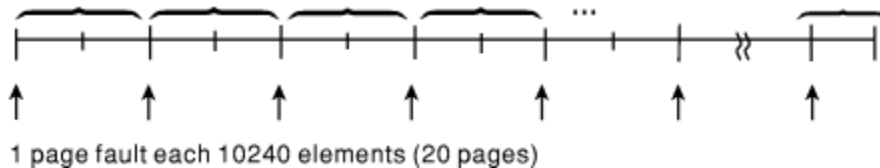
First, consider how the system brings data into central storage without information from reference pattern services. At the first reference of ARRAY1, the system takes a page fault and brings into central storage the page (of 4096 bytes) that contains the first element. After the program finishes processing 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.



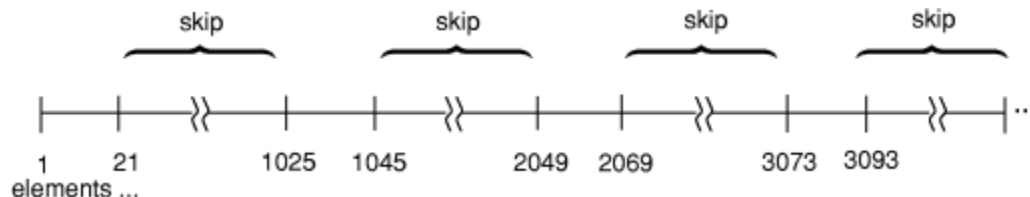
By bringing in one page at a time, the system takes 2048 page faults (8388608 divided by 4096), each page fault adding to the elapsed time of the program.

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 7 on page 64 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 7 on page 64 illustrates ARRAY1, the 1024-by-1024 array of eight-byte elements, where the program references 20 elements, then skips over the next 1004, and so forth in a forward direction throughout the array. The reference pattern includes a reference unit of 160 and a gap of 8032 bytes. The reference units begin on every other page boundary.

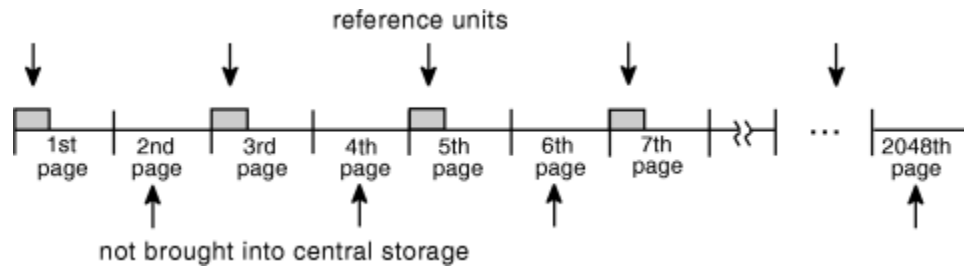
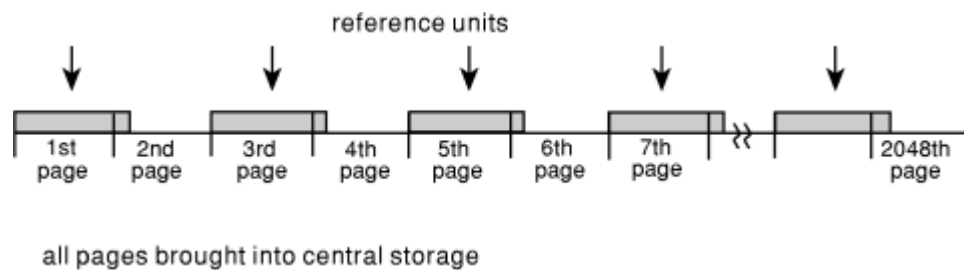


Figure 7. Illustration of a Reference Pattern with a Gap

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

Example 2

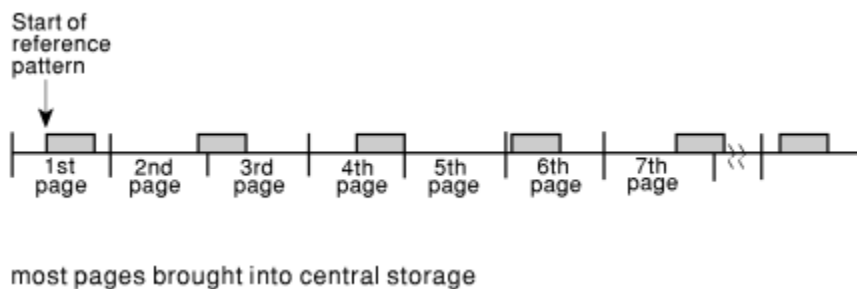
In example 2, the reference pattern includes a reference unit of 4800 bytes and a gap of 3392 bytes. The example assumes that the area to be referenced starts on a page boundary.



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

Example 3

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



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

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

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

Chapter 6. Using reference pattern services

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

Defining the reference pattern for a data area

On CSRIRP, you tell the system:

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

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

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

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

Defining the range of the area

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

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

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

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

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

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

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

These two rules are described and illustrated in [“Using CSRIRP when a gap exists” on page 67](#).

Identifying the direction of the reference

On *direction*, you specify the direction of reference through the array. Forward reference means instructions start with the element indicated by *low_address* and proceed through the range of data

specified by *size*. Backward reference means the program starts processing the high-address end of the range specified by *size* and proceeds toward the *low_address* end.

- "+1" indicates forward direction.
- "-1" indicates backward direction.

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

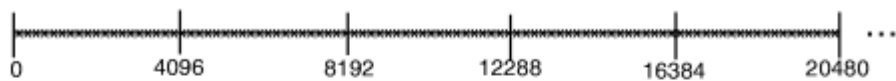
```
CSRIRP with direction of +1
```

"Using CSRIRP when a gap exists" on page 67 contains examples of forward and backward references when a gap exists.

Defining the reference pattern

Figure 8 on page 66 identifies two reference patterns that characterize most of the reference patterns that reference pattern services applies to.

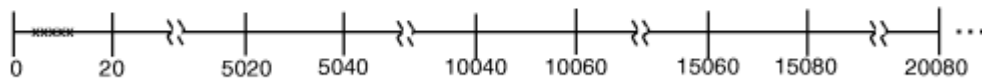
Pattern #1: No uniform gap



Characteristics of pattern:

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

Pattern #2: Uniform gap



Characteristics of pattern:

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

Figure 8. Two Typical Reference Patterns

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

- With pattern #1 where no uniform gap exists, the program uses every element, every other element, or at least most elements on each page of array data. No definable gap exists. Do not use reference pattern services if the reference pattern is irregular and includes skipping over many areas larger than a page.
 - The *unitsize* parameter identifies the reference pattern; it indicates the number of bytes you want the system to use as a reference unit. Look at logical groupings of bytes, such as one row, a number of rows, or one element, if the elements are large in size. Or, you might choose to divide the area to be referenced, and bring in that area on a certain number of page faults. Use the value 0 on *gapsize*.
 - The *units* parameter tells the system how many reference units to try to bring in on a page fault. For a reference pattern that begins on a page boundary and has no gaps, the total number of bytes the system tries to bring into central storage at a time is the value on *unitsize* times the number on *units*, rounded up to the nearest multiple of 4096. See "Choosing the number of bytes on a page fault" on page 67 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 8 on page 66 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 63 can help you understand how many bytes come into central storage at one time when a gap exists.

Using CSIRP 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 9 on page 67 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.

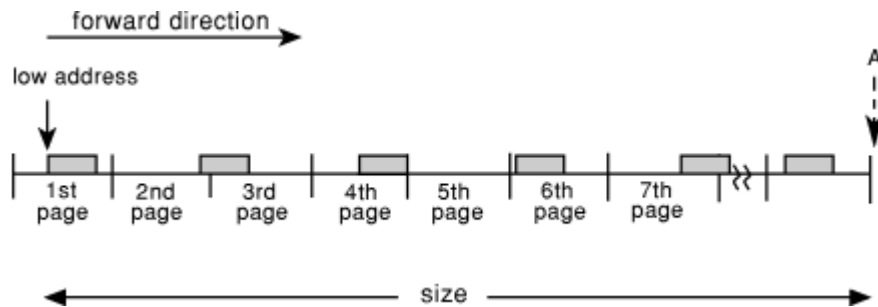


Figure 9. Illustration of Forward Direction of Reference

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

Figure 10 on page 67 illustrates backward reference through the same array as in Figure 9 on page 67. Again, the program references 2000 bytes and skips the next 5000 bytes, and so forth throughout the array. The range starts at *low_address* and ends at the point identified in the figure by **B**, where **B** must be the high-address end of a reference unit. *low_address* can be any part of a gap or reference unit.

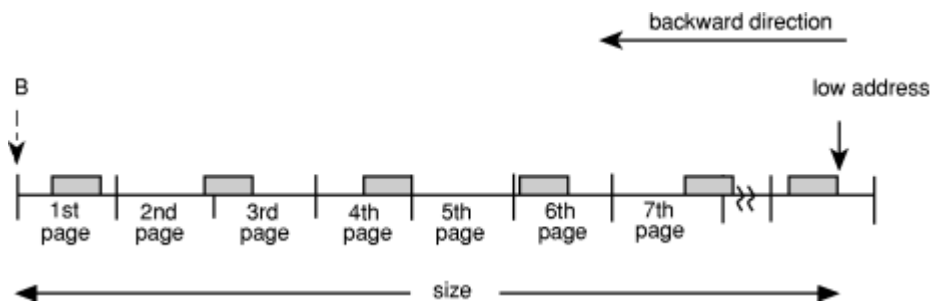


Figure 10. Illustration of Backward Direction of Reference

Choosing the number of bytes on a page fault

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

Pages do not stay in central storage if they are not referenced frequently enough and other programs need that central storage. The longer it takes for a program to begin referencing a page in central storage, the

greater the chance that the page has been moved out before being referenced. When you tell the system how many bytes it should try and bring into central at one time, you have to consider the following:

1. Contention for central storage:

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

2. Contention for processor time:

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

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

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

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

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

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

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

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

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

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

Reference pattern services affects movement of pages from auxiliary **and** expanded storage to central storage. To gain insight into the effectiveness of your reference patterns, you and your system programmer will need the kind of information that the SMF Type 30 record provides. A Type 30 record includes counts of pages moved in anticipation of your program's use of those pages. The record provides counts of pages moved between expanded and central and between auxiliary and central. It also provides elapsed time values. Use this information to calculate rates of movement in determining

whether to specify a very large number of bytes — for example, amounts greater than 204800 bytes (50 pages).

Examples of using CSRIRP to define a reference pattern

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

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

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

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

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

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

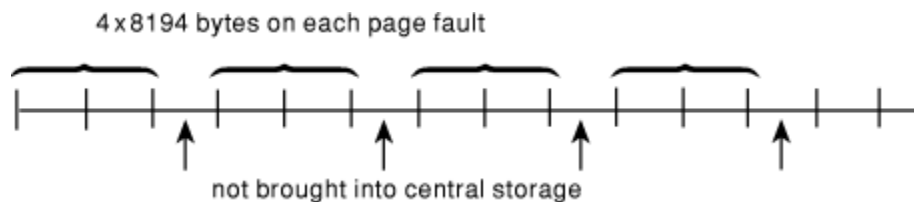
```
CSRIRP with unitsize of 4000 bytes
           gapsize of 0 bytes
           units of 50
           direction of +1
```

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

The following CSRIRP service reflects this reference pattern:

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

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



Removing the definition of the reference pattern

When a program is finished referencing the array in the way you specified on CSRIRP, use CSRRRP to remove the definition. The following example tells the system that the program in [“Defining the range of the area”](#) on page 65 has stopped referencing the array. *low_address* and *size* have the same values you coded on the CSRIRP service that defined the reference pattern for that area.

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

Handling return codes

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

When you receive a return code that indicates a problem or an unusual condition, try to correct the problem, and rerun the program. Return codes and reason codes are described in Chapter 7, [“Reference pattern services,”](#) on page 71 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 topic 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 75.

This topic contains the following subtopics:

- [“CSRIRP — Define a reference pattern”](#) on page 71
- [“CSRIRP — Remove a reference pattern”](#) on page 73.

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 CSRIRP service.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown. 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 statement	Parameters
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 CSRRRP 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 CSRRRP are:

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

CSRRRP – Remove a reference pattern

Call CSRRRP to remove the reference pattern for a data area, as specified by the CSRRRP 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 CSRRRP 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. 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 statement	Parameters
CALL CSRRRP	(low_address ,size ,return_code ,reason_code)

The parameters are explained as follows:

low_address

Specifies the beginning point of the data to be referenced.

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

,size

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

Define *size* as integer data of length 4.

,return_code

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

,reason_code

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

Return codes and reason codes

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

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

Chapter 8. Reference pattern services coding examples

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

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 topic 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 in the following topics:

- [“C/370 example” on page 75](#)
- [“COBOL example” on page 77](#)
- [“FORTRAN example” on page 81](#)
- [“Pascal example” on page 83](#)
- [“PL/I example” on page 85](#)

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 linenum);

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 rs;
    long int arraysize;
    long int direction;
    long int unitssize;
    long int gap;
    long int units;

    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__);

arraysize = m*p*kelement_size;

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

arraysize = m*n*kelement_size;

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

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

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

arraysize = n*p*kelement_size;
gap = (p-1)*kelement_size;
units = 50;
csrirp(B, &arraysize, &direction,;
      &unitsize,;
      &gap,;
      &units,;
      &rc,;
      &rsn);
chk_code(rc,rsn,__LINE__);
for (i=0; i<m; i++) {
    for (j=0; j<p; j++) {
        C[i][j] = 0;
        for (k=0; k<n; k++) {
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
        }
    }
}
arraysize = m*n*kelement_size;
csrrrp(A, &arraysize,;
      &rc,;
      &rsn);
chk_code(rc,rsn,__LINE__);
arraysize = n*p*kelement_size;
csrrrp(B, &arraysize,;

```



```

        &rc,;
        &rsn);
    chk_code(rc,rsn,__LINE__);
}

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

int chk_code(long int ret, long int reason, int linenumber)
{
    if (ret != 0)
        printf("return_code = %ld instead of 0 at line %d\n",
               ret, linenumber);
    if (reason != 0)
        printf("reason_code = %ld instead of 0 at line %d\n",
               reason, linenumber);
}

/*-----
/* JCL USED TO COMPILE, LINK, THE C/370 PROGRAM
/*-----
//CJOB JOB
//CCSTEP EXEC EDCCO,
//  CPARM='LIST,XREF,OPTIMIZE,RENT,SOURCE',
//  INFILE='REFPAT.SAMPLE.PROG(C),DISP=SHR'
//COMPILE.SYSLIN DD DSN='TEST.MPS.OBJ(C),DISP=SHR'
//COMPILE.USERLIB DD DSN=REFPAT.DECLARE.SET,DISP=SHR
//LKSTEP EXEC EDCPLO,
//  LPARM='AMOD=31,LIST,REFR,RENT,RMOD=ANY,XREF'
//PLKED.SYSIN DD DSN='TEST.MPS.OBJ(C),DISP=SHR'
//LKED.SYSLMOD DD DSN=REFPAT.USER.LOAD,DISP=SHR,
//  UNIT=3380,VOL=SER=RSMPAK
//LKED.SYSIN DD *
//  LIBRARY IN(CSRIRP,CSRIRP)
//  NAME BPGC(R)
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR
/*-----
/* JCL USED TO EXECUTE THE C/370 PROGRAM
/*-----
//CGO JOB TIME=1440,MSGLEVEL=(1,1),MSGCLASS=A
//RUN EXEC PGM=BPGC,TIME=1440
//STEPLIB DD DSN=REFPAT.USER.LOAD,DISP=SHR,
//  UNIT=3380,VOL=SER=VM2TSO
//  DD DSN=CEE.SCEERUN,DISP=SHR
//SYSPRINT DD SYSOUT=*
//PLIDUMP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*

```

COBOL example

```

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

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

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

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

* 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).

```

COBOL example

```
* 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,
        CSRFORWARD,
        UNITSIZE,
        GAP,
        UNITS,
        RETCODE,
        RSNCODE

    DISPLAY "FIFTH RETURN CODE IS "
    DISPLAY RETCODE

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

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

    DISPLAY "SIXTH RETURN CODE IS "
    DISPLAY RETCODE

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

```

COBOL example

```

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

    DISPLAY "SEVENTH RETURN CODE IS "
    DISPLAY RETCODE

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

    DISPLAY "EIGHTH RETURN CODE IS "
    DISPLAY RETCODE

    DISPLAY " BPAGE PROGRAM END "
    GOBACK.

/*-----
/* JCL USED TO COMPILE, LINK, THE COBOL PROGRAM
/*-----
//FCHANGC JOB 'D3113P,D31,?', 'FCHANG6-6756', CLASS=T,
//  MSGCLASS=H, NOTIFY=FCHANG, REGION=0K
//CCSTEP EXEC EDCCO,
//  CPARM='LIST,XREF,OPTIMIZE,RENT,SOURCE',
//  INFILE='FCHANG.PUB.TEST(C)'
//COMPILE.SYSLIN DD DSN='FCHANG.MPS.OBJ(C)', DISP=SHR'
//COMPILE.USERLIB DD DSN='FCHANG.DECLARE.SET, DISP=SHR
//LKSTEP EXEC EDCPLO,
//  LPARM='AMOD=31, LIST, REFR, RENT, RMOD=ANY, XREF'
//PLKED.SYSIN DD DSN='FCHANG.MPS.OBJ(C)', DISP=SHR'
//LKED.SYSLMOD DD DSN=RSMID.FBB4417.LINKLIB, DISP=SHR,
//  UNIT=3380, VOL=SER=RSMPAK
//LKED.SYSIN DD *
//  LIBRARY IN(CSRIRP, CSRRRP)
//  NAME BPGC(R)
//LKED.IN DD DSN=FCHANG.MPS.OBJ, DISP=SHR
/*-----
/* LINK PROGRAM
/*-----
//COBOLK JOB                                00010002
//LINKEDIT EXEC PGM=IEWL,                    00040000
//  PARM='MAP,XREF,LIST,LET,AC=1,SIZE=(1000K,100K)' 00050000
//SYSLIN DD DDNAME=SYSIN                     00051000
//SYSLMOD DD DSN=REFPAT.USER.LOAD, DISP=OLD      00052002
//SYSLIB DD DSN=CEE.SCEELKED, DISP=SHR           00053000
//MYLIB DD DSN=REFPAT.COBOL.OBJ, DISP=SHR        00053102
//CSRLIB DD DSN=SYS1.CSSLIB, DISP=SHR            00053202
//SYSPRINT DD SYSOUT=H                         00053300
//*                                              00053400
//SYSUT1 DD UNIT=SYSDA, SPACE=(TRK, (20,10))    00053500
//SYSUT2 DD UNIT=SYSDA, SPACE=(TRK, (20,10))    00053600
//SYSIN DD *                                    00053700
//  INCLUDE MYLIB(COBOL)                       00053802
//  LIBRARY CSRLIB(CSRIRP, CSRRRP)              00053901
//  NAME COBLOAD(R)                             00054002
//*                                              00055000
/*-----
/* JCL USED TO EXECUTE THE COBOL PROGRAM
/*-----
//COB2 JOB MSGLEVEL=(1,1), TIME=1440          00010000
//GO EXEC PGM=COBLOAD                         00020001
//STEPLIB DD DSN=CEE.SCEERUN, DISP=SHR         00030001
//  DD DSN=REFPAT.USER.LOAD, DISP=SHR, VOL=SER=RSMPAK, 00040001
//  UNIT=3380                                     00041001
//SYSABOUT DD SYSOUT=*                         00050000
//SYSOUT DD SYSOUT=A                           00051001
//SYSDABOUT DD SYSOUT=*                       00060000
//SYSUDUMP DD SYSOUT=*                         00070000

```

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)
PROGRAM BPAGEFOR
C
C   INCLUDE 'SYS1.SAMPLIB(CSRBPFOR)'
C
C   Multiply two arrays together - testing CSRIRP, CSRRRP services
C
C
C   INTEGER M /200/
C   INTEGER N /200/
C   INTEGER P /200/
C   PARAMETER (NKELEMENT_SIZE=4)
C   INTEGER RC,RSN
C   COMMON /WINCOM/A(200,200)
C   COMMON /WINCOM/B(200,200)
C   COMMON /WINCOM/C(200,200)
C
C   Initialize the arrays
C
C   CALL CSRIRP(A(1,1),
*           M*N*NKELEMENT_SIZE,
*           CSR_FORWARD,
*           M*NKELEMENT_SIZE,
*           0,
*           20,
*           RC,
*           RSN)
C   CALL CSRIRP(B(1,1),
*           N*P*NKELEMENT_SIZE,
*           CSR_FORWARD,
*           N*NKELEMENT_SIZE,
*           0,
*           20,
*           RC,
*           RSN)
C   DO 102 J = 1, N
C   DO 100 I = 1, M
C       A(I,J) = I + J
100 CONTINUE
102 CONTINUE
C   DO 106 J = 1, P
C   DO 104 I = 1, N
C       B(I,J) = I + J
104 CONTINUE
106 CONTINUE
C
C   CALL CSRRRP(A(1,1),
*           M*N*NKELEMENT_SIZE,
*           RC,
*           RSN)
C   CALL CSRRRP(B(1,1),
*           N*P*NKELEMENT_SIZE,
*           RC,
*           RSN)
C
C   Multiply the two arrays together
C
C   CALL CSRIRP (A(1,1),
*           M*N*NKELEMENT_SIZE,
*           CSR_FORWARD,
*           N*NKELEMENT_SIZE,
*           (N-1)*NKELEMENT_SIZE,
*           50,
*           RC,
*           RSN)
C   CALL CSRIRP (B(1,1),
*           N*P*NKELEMENT_SIZE,
*           CSR_FORWARD,
*           NKELEMENT_SIZE*N,
*           0,

```

FORTRAN example

```

      *          20,
      *          RC,
      *          RSN)
      DO 112 I = 1, M
      DO 110 J = 1, N
      DO 108 K = 1, P
      C(I,J) = C(I,J) + A(I,K) * B(K,J)
108 CONTINUE
110 CONTINUE
112 CONTINUE
      CALL CSRRRP (A(1,1),
      *          M*N*NKELEMENT_SIZE,
      *          RC,
      *          RSN)
      CALL CSRRRP (B(1,1),
      *          N*P*NKELEMENT_SIZE,
      *          RC,
      *          RSN)

      STOP
      END

```

```

*****00010000
*00020000
* THIS IS THE JCL THAT COMPILES THE PROGRAM.00030000
*00020000
*****00080000
//FORTJOB JOB00090007
// MSGCLASS=H,RDR=R,00110007
// MSGLEVEL=(1,1),CLASS=T00120000
//*00130000
//*00140000
//* COMPILE AND LINKEDIT FOR FORTRAN00150000
//*00160000
//*00170000
//*00180000
//VSF2CL PROC FVPGM=FORTVS2,FVREGN=2100K,FVPDECK=NODECK,00190000
// FVPOLST=NOLIST,FVPOPT=0,FVTERM='SYSOUT=A',00200000
// PGMNAME=MAIN,PGMLIB='&&GOSET',FVLNSPC='3200,(25,6) '00210000
//*00220000
//* COPYRIGHT: 5668-80600230000
//* (C) COPYRIGHT IBM CORP 1985, 198800240000
//* LICENSED MATERIALS - PROPERTY OF IBM00250000
//* REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-208300260000
//*00270000
//* STATUS: 02.03.00 (VV.RR.MM)00280000
//*00290000
//* PARAMETER DEFAULT-VALUE USAGE00300000
//*00310000
//* FVPGM FORTVS2 COMPILER NAME00320000
//* FVREGN 2100K FORT-STEP REGION00330000
//* FVPDECK NODECK COMPILER DECK OPTION00340000
//* FVPOLST NOLIST COMPILER LIST OPTION00350000
//* FVPOPT 0 COMPILER OPTIMIZATION00360000
//* FVTERM SYSOUT=A FORT.SYSTEM OPERAND00370000
//* FVLNSPC 3200,(25,6) FORT.SYSLIN SPACE00380000
//* PGMLIB &&GOSET LKED.SYSLMOD DSNAME00390000
//* PGMNAME MAIN LKED.SYSLMOD MEMBER NAME00400000
//*00410000
//FORT EXEC PGM=&FVPGM,REGION=&FVREGN,COND=(4,LT),00420000
// PARM='&FVPDECK,&FVPOLST,OPT(&FVPOPT) '00430000
//STEPLIB DD DSN=D24PP.FORT230.VSF2COMP,DISP=SHR00440000
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=342900450000
//SYSTEM DD &FVTERM00460000
//SYSPUNCH DD SYSOUT=B,DCB=BLKSIZE=344000470000
//SYSLIN DD DSN=&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,00480000
// SPACE=(&FVLNSPC),DCB=BLKSIZE=320000490000
//LKED EXEC PGM=HEWL,REGION=768K,COND=(4,LT),00500000
// PARM='LET,LIST,XREF '00510000
//SYSPRINT DD SYSOUT=A00520000
//SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR00530000
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(200,20))00540000
//SYSLMOD DD DSN=PGMLIB.(&PGMNAME),DISP=(,PASS),UNIT=SYSDA,00550000
// SPACE=(TRK,(10,10,1),RLSE)00560000
//SYSLIN DD DSN=&&LOADSET,DISP=(OLD,DELETE)00570000
// DD DDNAME=SYSIN00580000
// PEND00590000
// EXEC VSF2CL,FVTERM='SYSOUT=H',00600000
// PGMNAME=FORTRAN,PGMLIB='REFPAT.USER.LOAD'00680000
//FORT.SYSIN DD DSN=REFPAT.SAMPLE.PROG(FORTRAN),DISP=SHR00690000
//LKED.SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR00700000

```

```
//LKED.SYSLMOD DD DSN=REFPAT.USER.LOAD,DISP=SHR 00710007
//LKED.SYSIN DD * 00720000
INCLUDE IN(CSRIRP,CSRRRP,ADDR) 00730000
NAME BPGFORT(R) 00740006
/* 00750000
//* THE CSR STUBS ARE AVAILABLE IN SYS1.CSSLIB, 00760007
//* THE OBJ FOR THE ADDR ROUTINE IS IN TEST.OBJ 00770007
//* 00780000
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR 00790007
// DD DSN=REFPAT.TEST.OBJ,DISP=SHR 00mm0007
```

```
***** 00010000
* * 00020000
* THIS IS THE JCL I USE TO EXECUTE THE PROGRAM. * 00030000
* * 00060000
***** 00070000
//FON01 JOB MSGLEVEL=(1,1),TIME=1440 00080003
//VSF2G PROC GOPGM=MAIN,GOREGN=100K, 00090000
//* 00100000
//* 00110000
//* EXECUTE A FORTRAN TESTCASE - CHANGE ALL CRTFONXX TO CRTFONZZ 00120000
//* 00130000
// GOF5DD='DDNAME=SYSIN', 00140000
// GOF6DD='SYSOUT=A', 00150000
// GOF7DD='SYSOUT=B' 00160000
//* 00170000
//* COPYRIGHT: 5668-806 00180000
//* (C) COPYRIGHT IBM CORP 1985, 1988 00190000
//* LICENSED MATERIALS - PROPERTY OF IBM 00200000
//* REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083 00210000
//* 00220000
//* STATUS: 02.03.00 (VV.RR.MM) 00230000
//* 00240000
//* PARAMETER DEFAULT-VALUE USAGE 00250000
//* 00260000
//* GOPGM MAIN PROGRAM NAME 00270000
//* GOREGN 100K GO-STEP REGION 00280000
//* GOF5DD DDNAME=SYSIN GO.FT05F001 DD OPERAND 00290000
//* GOF6DD SYSOUT=A GO.FT06F001 DD OPERAND 00300000
//* GOF7DD SYSOUT=B GO.FT07F001 DD OPERAND 00310000
//* 00320000
//* 00330000
//GO EXEC PGM=&GOPGM,REGION=&GOREGN,COND=(4,LT) 00340000
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR 00350004
//FT05F001 DD &GOF5DD 00360000
//FT06F001 DD &GOF6DD 00370000
//FT07F001 DD &GOF7DD 00380000
// PEND 00390000
//GO EXEC VSF2G,GOPGM=BPGFORT,GOREGN=999K 00400004
//GO.STEPLIB DD DSN=WIND0W.D24PP.FORTLIB,DISP=SHR, 00410004
// VOL=SER=VM2TS0,UNIT=3380 00410104
// DD DSN=WIND0W.R40.VSF2LOAD,DISP=SHR, 00411004
// VOL=SER=VM2TS0,UNIT=3380 00412004
// DD DSN=REFPAT.USER.LOAD,DISP=SHR, 00420003
// VOL=SER=VM2TS0,UNIT=3380 00430004
```

Pascal example

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

  %include CSRBPPAS

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

  VAR
```

Pascal example

```

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
    csrrip (a(.1,1.), a_size, csr_forward,
            kelement_size*m,
            0,
            50,
            rc,
            rsn);
    csrrip (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
            b(.i,j.) := i + j;
            csrrrp (a(.1,1.), a_size,
                    rc,
                    rsn);
            csrrrp (b(.1,1.), b_size,
                    rc,
                    rsn);
/* Multiply the two arrays together */

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

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

```



```

//LKED.SYSIN DD *                                00200000
LIBRARY IN(CSRIRP,CSRRRP)                        00210005
NAME BPGPASC(R)                                  00220003
/*                                                00230000
//*      SYS1.CSSLIB IS THE SOURCE OF THE CSR STUBS 00240008
//*      00250000
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR             00260008
*****
*                                                *
*      JCL TO EXECUTE PASCAL                      *
*                                                *
*****
//PASC1JOB JOB                                00010005
//GO EXEC PAS22CL                             00050000
//*                                            00050102
//*      Compile and linkedit for PASCAL          00050202
//*                                            00050302
//PASC.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTPAN06),DISP=SHR 00060006
//LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,DISP=SHR,UNIT=3380, 00560000
// VOL=SER=VM2TSO                                00570000
//LKED.SYSIN DD *                                00580000
LIBRARY IN(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC) 00590000
NAME CRTPAN06(R)                                00600006
/*                                                00610000
//*      SYS1.CSSLIB is the source of the CSR stubs 00620002
//*      00650002
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR             00690000
*****
*                                                *
*      JCL TO COMPILE AND LINKEDIT.              *
*                                                *
*                                                *
*                                                *
*****
***** 00010000
* 00020000
* JCL TO EXECUTE. THIS ONE NEEDS A DD STATEMENT FOR THE * 00030000
* PERMANENT DIV OBJECT - CSRDD1. DATASET ALREADY EXISTS. * 00040000
* 00060000
***** 00070000
//PASCJOB JOB MSGLEVEL=(1,1),TIME=1440           00080002
//*                                            00090000
//*                                            00100000
//*      RUN A PASCAL TESTCASE - CHANGE THE NAME ON THE NEXT LINE 00110000
//*                                            00/20000
//*                                            00130000
//GO EXEC PGM=BPGPASC                             00140000
//STEPLIB DD DSN=REFPAT.USER.LOAD,               00150002
// DISP=SHR,UNIT=3380,                          00190000
// VOL=SER=VM2TSO                                00200003
//CSRDD1 DD DSN=DIV.TESTDS,DISP=SHR              00210000
//OUTPUT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=133)    00220000
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=133)  00230000
-----

```

PL/I example

```

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

```

```

the three arrays (ie. FIXED(31) is 4 bytes
                                */
                                00029713
                                00029813
                                00029913
                                00030013
                                00031013
                                00031113
                                00031213
                                00032013
                                00037013
                                00039013
                                00039113
                                00390108
                                00391808
                                00411013
                                00412013
                                00413013
                                00414013
                                00415013
                                00416013
                                00417013
                                00418013
                                00419013
                                00419113
                                00419913
                                00420013
                                00420113
                                00420213
                                00420313
                                00420413
                                00421213
                                00421313
                                00421413
                                00421513
                                00421613
                                00421713
                                00421813
                                00421913
                                00422013
                                00422113
                                00422213
                                00422313
                                00422513
                                00422613
                                00423413
                                00423613
                                00423713
                                00424513
                                00424613
                                00424713
                                00424813
                                00424913
                                00425013
                                00425133
                                00425213
                                00425313
                                00426113
                                00426213
                                00426313
                                00426413
                                00426513
                                00426613
                                00427413
                                00427513
                                00427613
                                00427713
                                00427813
                                00427913
                                00428013
                                00428113
                                00428213
                                00428313
                                00428513
                                00428613
                                00429413
                                00429613
                                00429713
                                00430513
                                01080024

/* 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 CSRIRP (A(1,1), M*N*KELEMENT_SIZE, CSR_FORWARD,
             KELEMENT_SIZE*N,
             0,
             20,
             RC,
             RSN);
CALL CSRIRP (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 CSRIRP (A(1,1), M*N*KELEMENT_SIZE, CSR_FORWARD,
             KELEMENT_SIZE*N,
             0,
             20,
             RC,
             RSN);
CALL CSRIRP (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;
  END;
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 COMPILE AND LINKEDIT.
*

```

```

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

```

Part 3. Global resource serialization latch manager services

Chapter 9. Using the latch manager services

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

This topic 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 topic contains the following subtopics:

- “ISGLCRT — Create a latch set” on page 91
- “ISGLOBT — Obtain a latch” on page 95
- “ISGLREL — Release a latch” on page 98
- “ISGLPRG — Purge a requestor from a latch set” on page 101
- “ISGLPBA — Purge a group of requestors from a group of latch sets” on page 102

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

Syntax and linkage conventions for latch manager callable services

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

CALL *routine_name(parameters)*

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

C

routine_name (parm1,parm2,...return_code)

COBOL

CALL “*routine_name*” USING *parm1,parm2,...return_code*

FORTRAN

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

PL/I

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

REXX

ADDRESS LU62 “*routine_name parm1 parm2...return_code*”

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

ISGLCRT — Create a latch set

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

ISGLOBT

Requests exclusive or shared ownership of a latch.

ISGLREL

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

ISGLPRG

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

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

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

Assign values to the following parameters:

- number_of_latches
- latch_set_name
- create_option

Latch_Create returns values in the following parameters:

- latch_set_token
- return_code

CALL statement	Parameters
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" 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 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 7. ISGLCRT Return Codes	
Return code and Equate symbol	Meaning and Action
00 (0) ISGLCRT_SUCCESS	Meaning: The Latch_Create service completed successfully. Action: None required.

Table 7. ISGLCRT Return Codes (continued)

Return code and Equate symbol	Meaning and Action
04 (4) ISGLCRT_DUPLICATE_NAME	<p>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.</p> <p>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.</p>
10 (16) ISGLCRT_NO_STORAGE	<p>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.</p> <p>Action: Specify a smaller value on the number_of_latches parameter.</p>

Examples of calls to latch manager services

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

```

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

main()
{
    const int numberOfLatches = 16; /* in this example we create 16
                                   latches */
    ISGLM_LSNM_type latchSetName
        = "EXAMPLE.ONE_LATCH_SET_NAME";
    ISGLM_LSTK_type latchSetToken; /* set up 48-byte latch set name */
    /* latch set token - output from
    create and input to obtain,
    release, and purge */
    int returnCode = 0; /* return code from services */

    const int latchNumber = 6; /* in this example we obtain latch
                                six */
    ISGLM_LRID_type requestorID = "123"; /* requestor ID - output from
    obtain and input to purge */
    int ECB = 0; /* ECB used for latch obtain
    service */
    ISGLM_EADDR_type ECBaddress = &ECB; /* pointer to ECB */
    ISGLM_LTK_type latchToken; /* latch token - output from
    obtain and input to release */

    union {
        double alignment; /* force double word alignment */
        ISGLM_WA_type area; /* set up work area */
    } work;

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

/*****
/* create a latch set with 16 latches
*****/
    isglcrt(numberOfLatches
        ,latchSetName
        ,ISGLCRT_PRIVATE
        ,&latchSetToken;
        ,&returnCode);

/*****
/* obtain latch
*****/
    isglobt(latchSetToken
        ,latchNumber
        ,requestorID
        ,ISGLOBT_SYNC /* suspend until granted */
        ,ISGLOBT_EXCLUSIVE /* access option (exclusive) */
        ,&ECBaddress /* required, but not used */
        ,&latchToken /* identifies request */

```

```

        ,&work.area
        ,&returnCode);

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

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

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

    isglprg(latchSetToken
            ,requestorID
            ,&returnCode);

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

```

ISGLOBT – Obtain a latch

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

ISGLCRT

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

ISGLREL

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

ISGLPRG

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

In the following description of Latch_Obtain:

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

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

Assign values to the following parameters:

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

Latch_Obtain returns values in the following parameters:

- latch_set_token
- return_code

Latch_Obtain uses the following parameter for temporary storage:

- work_area

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

The parameters are explained as follows:

latch_set_token

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

,latch_number

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

,requestor_ID

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

Recovery routines can purge all granted and pending requests for a particular requestor (identified by a requestor_id) within a specific latch set. When specifying the requestor_ID on Latch_Obtain, consider which latches would be purged if the Latch_Purge service were to be called with the specified requestor_ID. For more information about the Latch_Purge service, see [“ISGLPRG — Purge a requestor from a latch set”](#) on page 101.

,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.

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

Table 8. ISGLOBT Return Codes (continued)

Return code and Equate Symbol	Meaning and Action
04 (4) ISGLOBT_CONTENTION	<p>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.</p> <p>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.</p>

Example

See “Examples of calls to latch manager services” on page 94 for an example of how to call Latch_Obtain in C language.

ISGLREL — Release a latch

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

ISGLCRT

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

ISGLOBT

Requests exclusive or shared control of a latch.

ISGLPRG

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

In the following description of Latch_Release:

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

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

Assign values to the following parameters:

- latch_set_token
- latch_token
- release_option

Latch_Release returns a value in the following parameter:

- return_code

Latch_Release uses the following parameter for temporary storage:

- work_area

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

The parameters are explained as follows:

latch_set_token

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

,latch_token

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

,release_option

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

ISGLREL_UNCOND (value of 0)

Abend the requestor:

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

ISGLREL_COND (value of 1)

Return control to the requestor:

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

,work_area

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

,return_code

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

ABEND codes

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

Return codes

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

Table 9. ISGLREL Return Codes	
Return code and Equate Symbol	Meaning and Action
00 (0) ISGLREL_SUCCESS	<p>Meaning: The Latch_Release service completed successfully. The caller released ownership of the specified latch request.</p> <p>Action: None.</p>
04 (4) ISGLREL_NOT_OWNED_ECB_REQUEST	<p>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.</p> <p>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).</p>
08 (8) ISGLREL_STILL_SUSPENDED	<p>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.</p> <p>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.</p> <p>Action:</p> <ul style="list-style-type: none"> • 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.
0C (12) ISGLREL_INCORRECT_LATCH_TOKEN	<p>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.</p> <p>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.</p> <p>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.</p>

Example

See “[Examples of calls to latch manager services](#)” on page 94 for an example of how to call Latch_Release in C language.

ISGLPRG — Purge a requestor from a latch set

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

ISGLCRT

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

ISGLOBT

Requests exclusive or shared control of a latch.

ISGLREL

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

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

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

Assign values to the following parameters:

- latch_set_token
- requestor_ID

Latch_Purge returns a value in the return_code parameter.

CALL statement	Parameters
CALL ISGLPRG	(latch_set_token ,requestor_ID ,return_code)

The parameters are explained as follows:

latch_set_token

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

,requestor_ID

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

,return_code

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

ABEND codes

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

Return codes

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

Table 10. ISGLPRG Return Codes

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

Example

See “Examples of calls to latch manager services” on page 94 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 statement	Parameters
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 11. ISGLPBA Return Codes	
Return code and Equate Symbol	Meaning and Action
00 (0) ISGLPRG_SUCCESS	Meaning: The Latch_Purge_by_Address_Space service completed successfully. Action: None.
04 (4) ISGLPRG_DAMAGE_DETECTED	Meaning: Program error. While purging all requests for a particular requestor from a latch set, the latch manager found incorrect data in one or more latches. The latch manager tries to purge the latches that contain incorrect data, but the damage might prevent the latch manager from purging those latches. The latch manager purges the remaining latches (those with <i>correct</i> data) for the specified requestor. Action: Take a dump and check for a storage overlay. If your application can continue without the resources serialized by the damaged latches, no action is required.

Part 4. Resource recovery services (RRS)

Chapter 10. Using protected resources

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

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

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

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

Resource recovery programs

Three programs work together to protect resources:

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

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

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

- A database manager, such as DB2®
- A program, such as IMS/ESA® Transaction Manager, that accepts work from an end user or another system and manages that work

Note: The resource manager in resource recovery is different from an RTM resource manager, which is related to the operating system's recovery termination management (RTM) and runs during termination processing.

- Sync-point manager: The sync-point manager coordinates changes to protected resources, so that all changes are made or no changes are made. The z/OS sync-point manager is recoverable resource management services (RRMS). Three MVS components provide RRMS function; because resource recovery services (RRS) provides the sync-point services, most technical information uses RRS rather than RRMS.

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

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

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

Two-phase commit protocol

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

The phases of the protocol are:

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

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

Once a commit decision is hardened, the application changes are considered committed. If there is a failure after this point, the resource manager will make the changes during restart. Before this point, a failure causes the resource manager to back out the changes during restart.

- Phase 2: In the second phase, the resource managers commit or back out the changes.

Resource recovery process

For a look at the resource recovery process, think of a person who requests an automated teller machine (ATM) to transfer money from a savings account to a checking account. The application program receives the person's input from the ATM. Each account is in a different database. Each database has its own resource manager. The sync-point manager is RRS. [Figure 11 on page 108](#) shows how the ATM application, resource managers, and RRS work together

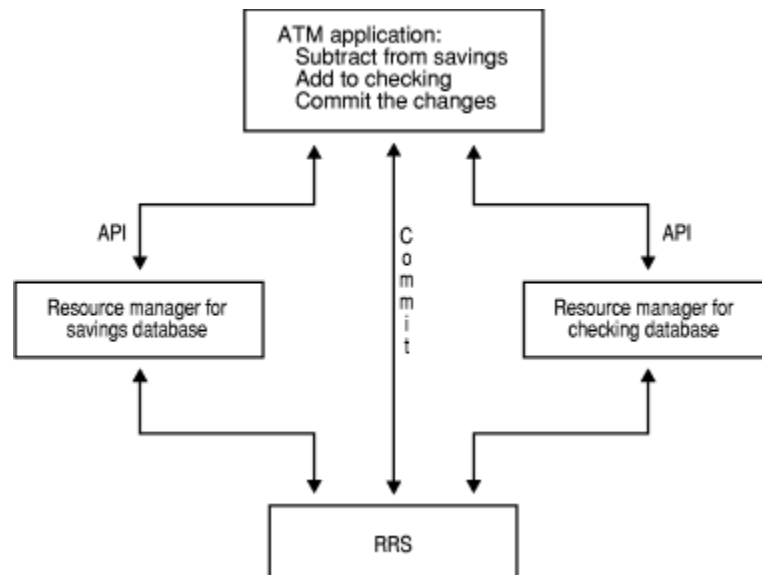


Figure 11. ATM Transaction

The actions required to process the ATM transaction are:

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

[Figure 12 on page 109](#) 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 108](#).

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 12 on page 109](#), both resource managers vote YES.
8. In response, RRS notifies the resource managers to commit the changes, that is, to make the changes permanently in the databases.
9. The resource managers complete the commit and return OK to RRS.
10. RRS gives a return code to the application program, indicating that all changes were made in the databases.

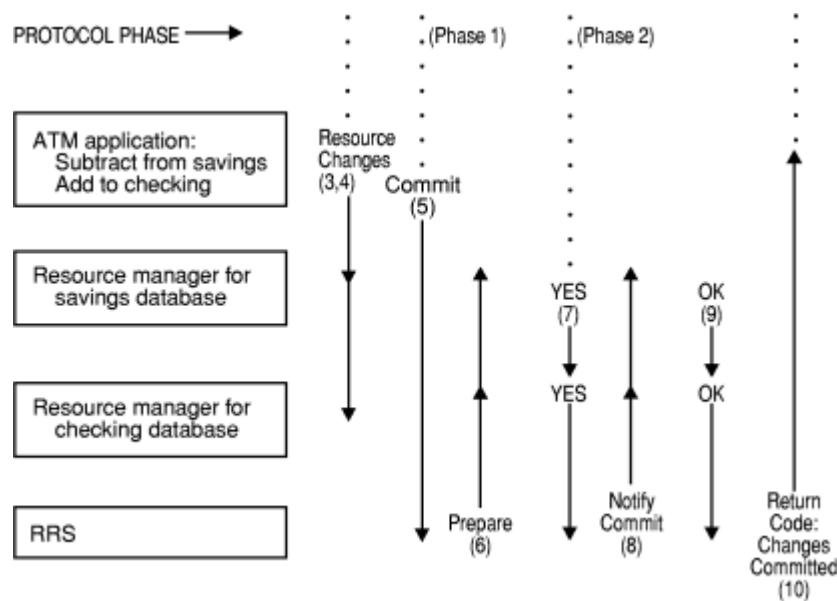


Figure 12. Two-Phase Commit Actions

If the ATM user decides not to transfer the money and presses a NO selection, the application requests backout, instead of commit, in step 6. In this case, the changes are backed out and are not actually made in any database. See [Figure 13 on page 109](#).

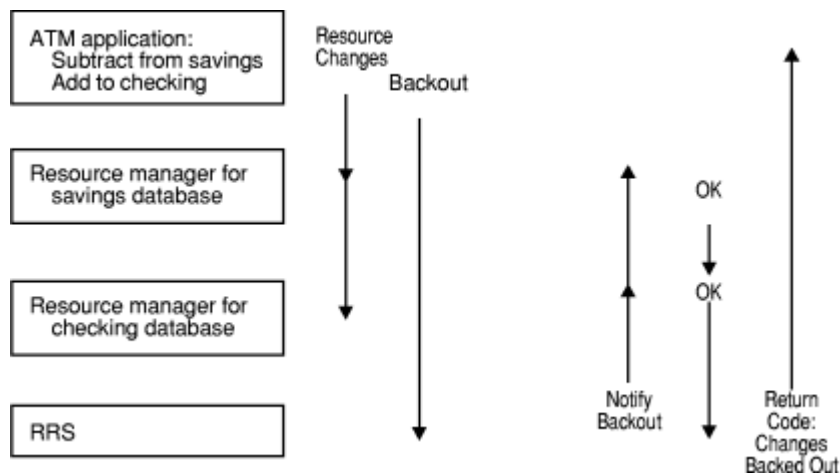


Figure 13. Backout – Application Request

Or if a resource manager cannot make the change to its database, the resource manager votes NO during prepare. If **any** resource manager votes NO, all of the changes are backed out. See Figure 14 on page 110.

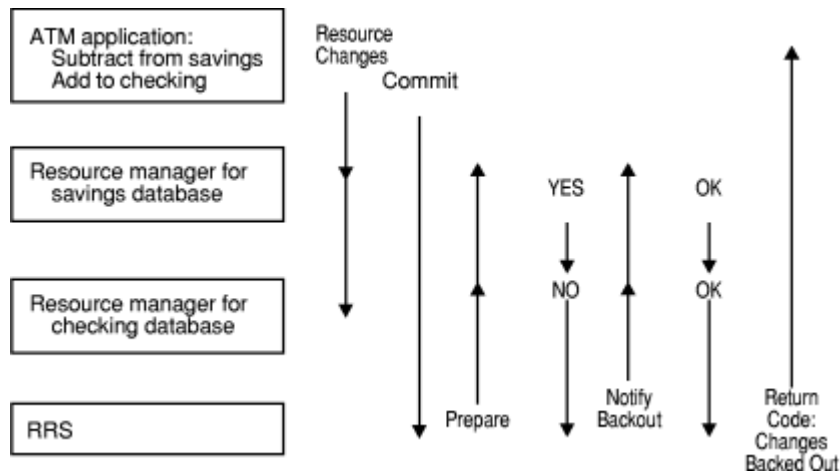


Figure 14. Backout – Resource Manager Votes NO

Requesting resource protection and recovery

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

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

For details about the calls, see “Application_Backout_UR (SRRBACK)” on page 111 and “Application_Commit_UR (SRRCMIT)” on page 114.

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 15 on page 110 illustrates distributed resource recovery.

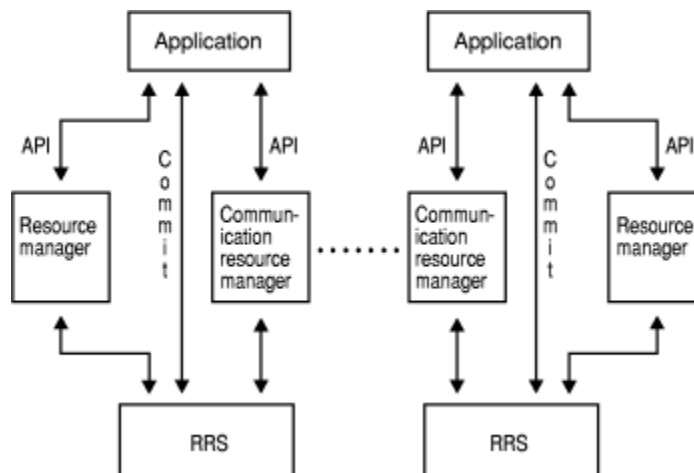


Figure 15. Transaction – Distributed Resource Recovery

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:

Requirement	Details
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 linking 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 SRRBACK 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 37 or [Chapter 8, “Reference pattern services coding examples,”](#) on page 75.

High level language (HLL) definitions

The high level language (HLL) definitions for the callable service are:

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

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

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

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

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

Restrictions

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

The UR cannot be in local transaction mode.

Input register information

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

Output register information

When control returns to the caller, the GPRs contain:

Register Contents

0-1

Used as work registers by the system

2-13

Unchanged

14

Used as a work register by the system

15

Return code

When control returns to the caller, the ARs contain:

Register Contents

0-1

Used as work registers by the system

2-13

Unchanged

14-15

Used as work registers by the system

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

Performance implications

None.

Syntax

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

CALL statement	Parameters
CALL SRRBACK	(return_code)

Parameters

The parameters are explained as follows:

return_code

Returned parameter

- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Application_Backout_UR service.

ABEND codes

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

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

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

Return codes

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

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
0	0	Code: RR_OK Meaning: Successful completion. The resource managers returned their resources to the values they had before the UR was processed. Action: None.
12D	301	Code: RR_BACKED_OUT_OUTCOME_PENDING Meaning: Environmental error. The backout was not completed, for one of the following reasons: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> Display a warning message to the end user. Write an exception entry into an output log. Abnormally end the application because the resource manager will not allow any further changes to the resource until the situation is resolved.
12E	302	Code: RR_BACKED_OUT_OUTCOME_MIXED Meaning: Environmental error. RRS requested that the resource managers back out the changes to the resources. However, one or more resources were changed. Action: Same as the action for return code 12D (301).

Example

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

```

:
CALL SRRBACK(RETCODE)
:

```

Application_Commit_UR (SRRCMIT)

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

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

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

Description

Environment

The requirements for the caller are:

Requirement

Minimum authorization:

Details

Problem state, any PSW key

Requirement	Details
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 linking 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 37](#) or [Chapter 8, "Reference pattern services coding examples," on page 75](#).

High level language (HLL) definitions

The high level language (HLL) definitions for the callable service are:

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

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

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

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

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

Restrictions

The state of the UR that represents the changes must be **in-reset** or **in-flight**.

The UR cannot be in local transaction mode.

Input register information

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

Output register information

When control returns to the caller, the GPRs contain:

Register

Contents

0-1

Used as work registers by the system

2-13

Unchanged

14

Used as a work register by the system

15

Return code

When control returns to the caller, the ARs contain:

Register

Contents

0-1

Used as work registers by the system

2-13

Unchanged

14-15

Used as work registers by the system

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

Performance implications

None.

Syntax

Write the call as shown in the syntax diagram. You must code the parameter in the CALL statement as shown.

CALL statement	Parameters
CALL SRRCMIT	(return_code)

Parameters

The parameters are explained as follows:

return_code

Returned parameter

- Type: Integer
- Length: 4 bytes

Contains the return code from the Application_Commit_UR service.

ABEND codes

The call might result in an abend X'5C4' with a reason code of X'00160000' through X'00160012'. See [z/OS MVS System Codes](#) for the explanations and actions.

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

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

Return codes

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

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
0	0	Code: RR_OK Meaning: Successful completion. The changes to all protected resources have been made permanent. Action: None.
65	101	Code: RR_COMMITTED_OUTCOME_PENDING Meaning: Environmental error. The commit was not completed: <ul style="list-style-type: none"> • RRS requested that the resource managers make the changes to the resources permanent. However, the state of one or more of the resources is not known. Action: The action by an application depends on the system environment. Some possible actions are: <ul style="list-style-type: none"> • 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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
66	102	Code: RR_COMMITTED_OUTCOME_MIXED Meaning: Environmental error. RRS requested that the resource managers make the changes to the resources permanent. One or more resources were changed, but one or more were not changed. Action: Same as the action for return code 65 (101).
C8	200	Code: RR_PROGRAM_STATE_CHECK Meaning: Environmental error. The commit failed. The resource managers did not make the changes to the resources because one of the following occurred: <ul style="list-style-type: none"> • A resource on the same system as the application is not in the proper state for a commit. • A protected conversation is not in the required state: send, send pending, defer receive, defer allocate, sync_point, sync_point send, sync_point deallocate. • A protected conversation is in send state. The communications manager started sending the basic conversation logical record, but did not finish sending it. Action: Initiate an action by a resource manager to get its resource to a committable state, then call Application_Commit_UR again. For example, if the application has allocated a protected conversation through APPPC/MVS, and the conversation is in receive state, the application gets this return code. It then must use APPC/MVS services to change the conversation to send state before issuing the commit request again.
12C	300	Code: RR_BACKED_OUT Meaning: Environmental error. The commit failed. The resource managers backed out the changes, returning the resources to the values they had before the UR was processed. Action: Same as the action for return code 65 (101).
12D	301	Code: RR_BACKED_OUT_OUTCOME_PENDING Meaning: Environmental error. The commit failed for one of the following reasons: <ul style="list-style-type: none"> • RRS requested that the resource managers back out the changes to the resources. However, the state of one or more of the resources is not known. • RRS is not active. Action: Same as the action for return code 65 (101).
12E	302	Code: RR_BACKED_OUT_OUTCOME_MIXED Meaning: Environmental error. The commit failed. RRS requested that the resource managers back out the changes to the resources. One or more resources were backed out, but one or more were changed. Action: Same as the action for return code 65 (101).

Example

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

```

:
CALL SRRCMIT(RETCODE)
:

```

Additional callable services

Additional callable services that an authorized resource manager can use to request resource recovery services can be found in *z/OS MVS Programming: Resource Recovery*.

Part 5. CEA TSO/E address space services

Chapter 11. Introduction to CEA TSO/E address space services

The z/OS CEA TSO/E address space manager provides services to programmatically start and manage TSO/E address spaces and provides a communications mechanism for use between the caller and the programs running in these managed address spaces.

CEA TSO/E address space services allow callers to:

- Start a new TSO/E address space.
- End a TSO/E address space started by CEA.
- Send an attention interrupt to a TSO/E address space started by CEA.
- Obtain information about a TSO/E address space started by CEA.
- Obtain information about all the TSO/E address spaces that CEA started for an application.
- Ping a TSO/E address space that was started by CEA to prevent the address space from ending because it has been idle too long.

Two versions of TSO/E Address Space Services are available: Version 1 and Version 2.

Version 1 TSOASMGR services

Version 1 TSOASMGR services allow a calling application to create sessions on only the system on which the function was invoked (the local system). Version 1 exploiters of TSO/E Address Space Services use `msgsnd()` and `msgrcv()` functions to directly receive data to and from the z/OS UNIX queue.

Version 2 TSOASMGR services

Version 2 TSOASMGR services allow a calling application to create and log in to a TSO/E address space on a different system in the sysplex (a remote system). With the Version 2 request structures, two new CEA provided APIs, `CEAmgsnd` and `CEAmgrcv`, are used to perform message send and message receive. Callers using the new APIs use the CEA connection handle.

CEA TSO/E address space manager components

The CEA TSO/E address space manager includes the common event adapter (CEA) component of z/OS. The CEA component provides the framework and manages the resources for the TSO/E address spaces. [Table 12 on page 121](#) describes the components that are included in the CEA TSO/E address space manager.

Table 12. CEA TSO/E address space manager components	
Component	Description
CEA address space	<p>The CEA TSO/E address space manager is integrated into the CEA address space infrastructure. The function is started automatically when CEA is started.</p> <p>Attention : If the CEA address space ends, all the TSO/E sessions that are created by CEA also end. Callers are not notified that the CEA address space is ended. Instead, when a caller attempts to invoke the CEA TSO/E address space services or use the z/OS UNIX message queue, the request fails.</p>

Table 12. CEA TSO/E address space manager components (continued)	
Component	Description
Session table	<p>When the CEA TSO/E address space manager starts a new TSO/E address space, the attributes of the address space and the resources that are obtained are stored in an internal session table. The entry exists for the life of the session and is removed when the TSO/E address space ends.</p> <p>To display the contents of the session table, use the MODIFY CEA, DIAG, SESSTABLE command. For more information about the command, see the topic about displaying the CEA TSO/E address space information in <i>z/OS MVS System Commands</i>.</p>
z/OS UNIX message queue	The CEA TSO/E address space manager creates and manages a z/OS UNIX message queue, which is used to facilitate communication between the caller and the TSO/E address space. For more information about the z/OS UNIX message queue, see “Communicating with programs running in the TSO/E address spaces” on page 124.
CEATsoRequest API	The CEA TSO/E address space manager provides the CEATsoRequest API, which is a 64-bit C-language-based API that callers can use to request TSO/E address space services. Use this API with Version 1 TSOASMGR services. For more information about the API, see Chapter 12, “Using CEA TSO/E address space services,” on page 129.
CEAmsgsnd API	The CEA TSO/E address space manager provides the CEAmsgsnd API, which is a 64-bit C-language-based API that callers can use to request TSO/E address space services. Use this API with Version 2 TSOASMGR services. For more information about the API, see Chapter 12, “Using CEA TSO/E address space services,” on page 129.
CEAmsgrcv API	The CEA TSO/E address space manager provides the CEAmsgrcv API, which is a 64-bit C-language-based API that callers can use to request TSO/E address space services. Use this API with Version 2 TSOASMGR services. For more information about the API, see Chapter 12, “Using CEA TSO/E address space services,” on page 129.

System prerequisites for the CEA TSO/E address space services

Table 13 on page 122 describes the system prerequisites for using the CEA TSO/E address space services.

Table 13. System prerequisites	
Prerequisite	Description
CEA must be active.	<p>The CEA TSO/E address space manager runs in the CEA address space, which is started automatically during z/OS initialization. If your installation stopped CEA, restart it. Otherwise, the services are not enabled.</p> <p>To determine whether the CEA address space is active, enter the following z/OS system console command:</p> <p>D A,CEA</p>

Table 13. System prerequisites (continued)

Prerequisite	Description
The TRUSTED attribute must be assigned to the CEA started task.	<p>To allow the CEA TSO/E address space manager to access or create any resource that it needs, the CEA started task requires the TRUSTED(YES) attribute to be set on the RDEFINE STARTED CEA.** definition.</p> <p>If the TRUSTED attribute is not assigned to the CEA started task, the CEA TSO/E address space manager services might not be operational. For example, the services cannot create or access z/OS UNIX message queues.</p> <p>For more information about the RACF® TRUSTED attribute, see the topic on associating started procedures and jobs with user IDs in <i>z/OS Security Server RACF System Programmer's Guide</i>, and the topic on using started procedures in <i>z/OS Security Server RACF Security Administrator's Guide</i>.</p>
The CEA address space must be started in full function mode.	Because the CEATsoRequest API requires z/OS UNIX System Services, CEA must be started in full function mode. For information about starting CEA in full function mode, see the topic about customizing CEA in <i>z/OS Planning for Installation</i> .
The external security manager (ESM) must have sysplex-wide scope.	<p>To create address spaces on other systems in the sysplex, ensure that the security identities of the caller are the same on each system. Your installation must ensure that the REALM class contains a SAFDFLT profile with an application name. In a RACF system, issue a command similar to the following command:</p> <pre>RDEFINE REALM SAFDFLT APPLDATA('racf.ceatsoasmgr')</pre>
Callers must be authorized to SAF resource profile CEA.CEATSO.TSOREQUEST.	To access the CEATsoRequest API, callers must be authorized by their security product to SAF resource profile CEA.CEATSO.TSOREQUEST in the SERVAUTH class.
<p>Ensure that callers are authorized to the following SAF resource profiles to allow them to send data to <i>systemname</i>:</p> <p>CEA.CEATSO.FLOW.<i>systemname</i></p>	<p>To flow data between different systems in the sysplex, ensure that the caller is authorized by the external security manager (ESM). Because the security database is sysplex wide in scope, CEA can check for both local and remote permissions on the system that initiated the request. For example, to flow data between System A and System B, the following profiles must permit CEA:</p> <ul style="list-style-type: none"> • CEA.CEATSO.FLOW.SYSTEMA • CEA.CEATSO.FLOW.SYSTEMB
Users must be authorized to the appropriate resources.	The user ID of the user for whom the caller is requesting TSO/E address space services must be authorized to use TSO/E, OMVS, and any other resources the address space requires.

Working with TSO/E address spaces started by CEA

By default, the CEA TSO/E address space manager can create a maximum of 10 concurrent address spaces for a single user, and can create a maximum of 50 concurrent TSO/E address spaces. These settings are configurable through the MAXSESSIONS and MAXSESSPERUSER operands provided for the TSOASMGR statement in the CEAPRMxx parmlib member. For more information, see *z/OS MVS Initialization and Tuning Reference*.

You can use the same processes that you use to work with other TSO/E address spaces when working with the TSO/E address spaces that are created by the CEA TSO/E address space manager. For example,

you can issue the D TS z/OS console command to display information about TSO/E address spaces, or you can issue the C u=userid,A=asid console command to cancel a TSO/E address space. For the display command, the TSO/E address spaces will appear in the list, indistinguishable from the other TSO/E address spaces. Note that TSO/E sessions started by CEA do not add to the count for the total maximum sessions for VTAM®.

You can also display information about these TSO/E address spaces using SDSF, a REXX EXEC, or a CLIST. Note that the application identifier that was specified when the TSO/E session was started is displayed where you would typically expect to see a terminal ID.

For example, if the CEA TSO/E address space manager starts a TSO/E session for the z/OSMF ISPF task, which has an application identifier equal to IZUIS, and you issue the REXX EXEC depicted in [Figure 16](#) on [page 124](#), you will obtain the results depicted in [Figure 17](#) on [page 124](#):

```
/* REXX */
trace all
myapp = sysvar('systemid')
say myapp
exit 0
```

Figure 16. Sample REXX EXEC

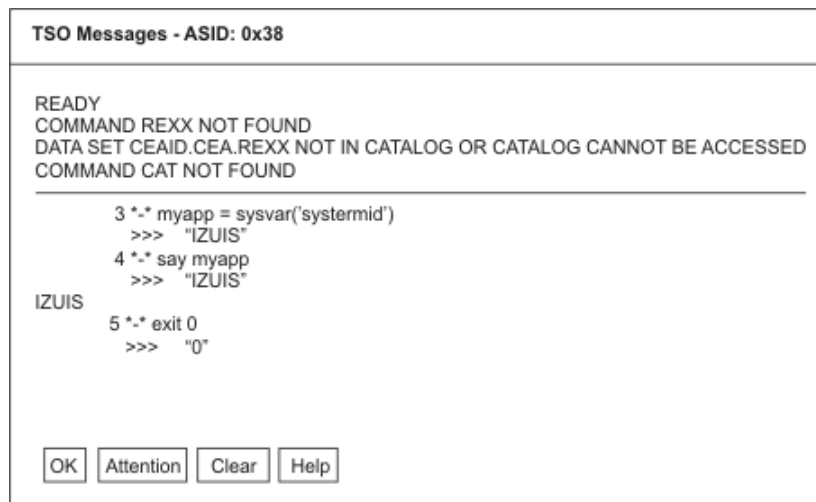


Figure 17. Example illustrating that the REXX SYSTERMID is the same as the z/OSMF ISPF application identifier

The following rules apply when working with remote TSO/E address spaces:

- When CEA sends data to a queue, it is sent to the remote queue.
- When CEA retrieves data, it is retrieved from the queue on the same system as the caller. This rule applies to all callers.

It is possible for callers to use the new Version 2 APIs without first creating the TSO/E address space on a remote system. In this situation, processing can determine that the TSO/E session is on the same system and will create only a single queue. To use the new APIs in this way, however, the caller must invoke the CEATSORequest () with a Version 2 CEATsoRequestStruct_s. Set the **systemname** field to either blank or the local system name.

Communicating with programs running in the TSO/E address spaces

A z/OS UNIX message queue is the mechanism the CEA TSO/E address space manager uses for allowing communications between the caller and TSO/E, ISPF, and other programs running in the TSO/E address space. To communicate with the TSO/E address space, callers must read data from and write data to the message queue.

The CEA TSO/E address space manager creates a z/OS UNIX message queue for each TSO/E address space when the TSO/E address space is started, and anchors the message queue in the session table for the duration of the session. The CEA TSO/E address space manager deletes the message queue when the TSO/E address space ends.

Messages that typically are written to a 3270-type terminal are translated to UTF-8, converted to a JSON format, and written to the z/OS UNIX message queue along with identifying header information and a message type identifier. For a list of the message type identifiers, see [Table 14 on page 125](#).

<i>Table 14. Message type identifiers</i>	
Message Type ID	Description
1	Control data for the client.
2	TSO/E data for the client.
3	ISPF data for the client.
4 thru 32768	Reserved for IBM.
32769	Control TSO/E data from the client.
32770	TSO/E data from the client.
32771	ISPF data from the client.
32772 thru 65535	Reserved for IBM.
65536 and above	Available for use by applications.

For information about the JSON format used for TSO/E messages, see “JSON format for TSO/E messages” on page 125. For the JSON format used for ISPF messages, see the topic about JSON data structures and variables used to communicate between ISPF and a client in [z/OS ISPF Services Guide](#).

JSON format for TSO/E messages

TSO/E messages are written to the z/OS UNIX message queue using message type identifiers 2 and 32770 and are formatted as follows:

```
{“message-type”:{“VERSION”:“JSON-version”,“data-type”:“data-value”}}
```

where:

message-type

Keyword that identifies the type of TSO/E message. [Table 15 on page 125](#) lists and describes the message types that can be used for message type identifiers 2 and 32770.

<i>Table 15. Message types</i>		
Message Type	Description	Message Type ID
TSO MESSAGE	Indicates that the system has created data or a message to be displayed on the client. The caller should read the message and display it accordingly.	2
TSO PROMPT	Indicates that the system requires a response from the client.	2
TSO RESPONSE	Indicates that a response was created by the client in response to a prompt. Callers should use this keyword when writing a response to the message queue.	32770

JSON-version

A four-digit number that identifies the JSON version used to format the message.

data-type

Keyword that describes the type of data included in the *data-value* variable. Table 16 on page 126 lists and describes the data types that can be used for each TSO/E message type.

Table 16. Data types		
Data Type	Description	Message Type
DATA	Indicates that the data included in the <i>data-value</i> variable is either a message from the system or a response from the client. For this data type, the <i>data-value</i> variable is a character string that can contain up to 32,767 bytes.	TSO MESSAGE and TSO RESPONSE
HIDDEN	Indicates whether the client should hide or mask the response. For this data type, the <i>data-value</i> variable is a Boolean that can have the value of either TRUE or FALSE. When TRUE, this tells the client to hide or mask the response as it is entered. Otherwise, the response will display as it is entered.	TSO PROMPT
ACTION	Indicates that the caller would like to interrupt or end a process that is in progress. For this data type, specify ATTN as the value for the <i>data-value</i> variable. Callers should use the CEATsoRequest API to issue the CeaTsoAttn request type before using a message to issue an attention interrupt. Use this data type only if the CeaTsoAttn request fails.	TSO RESPONSE

Sample TSO/E messages written to the z/OS UNIX message queue

Figure 18 on page 126 provides an example that illustrates how TSO/E messages appear on the z/OS UNIX message queue.

Note: The message type identifiers are not part of the JSON structure. They are included for illustration purposes only.

```
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"IKJ56455I
IBMUSER LOGON IN PROGRESS AT 03:46:24 ON OCTOBER 12, 2011"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"IKJ56951I NO BROADCAST MESSAGES
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"READY "}}}
2 {"TSO PROMPT":{"VERSION":"0100","HIDDEN":"FALSE"}}
32770 {"TSO RESPONSE":{"VERSION":"0100","DATA":"TIME"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"IKJ56650I TIME-03:46:50 AM.
CPU-00:00:00 SERVICE-775140 SESSION-00:00:26 OCTOBER 12,2011"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"READY "}}}
2 {"TSO PROMPT":{"VERSION":"0100","HIDDEN":"FALSE"}}
32770 {"TSO RESPONSE":{"VERSION":"0100","DATA":"ALLOC DA"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"IKJ56700A ENTER DATA SET NAME
OR * -
2 {"TSO PROMPT":{"VERSION":"0100","HIDDEN":"FALSE"}}
32770 {"TSO RESPONSE":{"VERSION":"0100","DATA":"'sys1.brodcast'"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"IKJ56225I DATA SET SYS1.BROADCAST
ALREADY IN USE, TRY LATER+"}}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"IKJ56225I DATA SET IS ALLOCATED
TO ANOTHER JOB OR USER"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"READY "}}}
2 {"TSO PROMPT":{"VERSION":"0100","HIDDEN":"FALSE"}}
32770 {"TSO RESPONSE":{"VERSION":"0100","DATA":"LOGOFF"}}
```

Figure 18. Sample TSO/E messages written to the queue

Reconnecting to CEA TSO/E address spaces

When a user requests to end a TSO/E session created by CEA, if the caller has not set the abnormal logoff flag (CEATSO_ABLOGOFF) or the no reconnect flag (CEATSO_NORECONN), the CEA TSO/E address space manager can intercept that request and place the session in a dormant state instead of ending it.

A *dormant TSO/E session* is a session that has been deactivated for communication through its message queue but remains available at a TSO/E READY prompt for a period of time so that the user can reconnect to it. Reconnecting to a dormant session is faster and uses fewer resources than constructing a new session because the session resources are retained and reused when the user reconnects to the session.

To enable the CEA reconnect feature, which is disabled by default, specify non-zero values for the RECONSESSIONS and RECONTIME statements in the TSOASMGR parmlib statement in the CEAPRMxx parmlib member. The RECONSESSIONS statement indicates how many dormant sessions can be created for each user, and the RECONTIME statement indicates the amount of time a dormant session remains a candidate for reconnection.

Important: Only Version 1 sessions honor this setting in the CEAPRMxx member. Version 2 sessions do not support the ability to reconnect. If you plan to allow the creation of version 2 remote sessions in the sysplex, you must modify the CEAPRMxx member to turn off the reconnection capability.

The CEA TSO/E address space manager can create a maximum of three dormant sessions per user and can keep a dormant session available for reconnection for a maximum of 23 hours, 59 minutes, and 59 seconds. The settings you specify for the TSOASMGR parmlib statement affect all of the TSO/E sessions that are managed by the CEA TSO/E address space manager. For more information about the TSOASMGR parmlib statement, see the topic about the CEAPRMxx parmlib member in [z/OS MVS Initialization and Tuning Reference](#).

When the CEA reconnect feature is enabled, to reconnect to a dormant session, the user must do the following:

- Request to start a new TSO/E session before the specified RECONTIME expires. After the RECONTIME expires, the session remains in a dormant state until CEA ends it; however, the session is no longer a candidate for reconnection.
- Use the same security credentials and logon parameters that were used for the dormant session.

If no dormant sessions are available that satisfy these requirements, the CEA TSO/E address space manager will create a new address space for the user.

Dormant TSO/E sessions do not interfere with the maximum number of sessions allowed. That is, if a user tries to create a new session and the number of active and dormant sessions equal the maximum allowed, the CEA TSO/E address space manager will end a dormant session and create a new session for the user.

Idle time versus RECONTIME

Each dormant TSO/E session has an idle application time, which is not adjustable, and a reconnect time (RECONTIME). The idle time cannot exceed 15 minutes. Otherwise, the CEA TSO/E address space manager will end the session regardless of reconnect time. To prevent your dormant sessions from ending because of idle time, issue a ping request at least once every 15 minutes, which informs CEA that all of the sessions for your application are still active. For more information, see [“CeaTsoPing - Sending a ping on behalf of an application”](#) on page 137.

TSO/E LOGON RECONNECT operand versus CEA reconnect

The TSO/E LOGON command is not supported for CEA-managed TSO/E sessions, and the capability provided by the TSO/E LOGON RECONNECT operand is different from the CEA reconnect feature. For more information about the TSO/E LOGON RECONNECT operand, see the topic about LOGON command operands in [z/OS TSO/E Command Reference](#).

Chapter 12. Using CEA TSO/E address space services

To use CEA TSO/E address space services, you issue CALLs from high-level language programs that invoke the CEATsoRequest API. The API is a 64-bit C-language based interface that the CEA TSO/E address space manager uses to receive requests from callers and to determine what action to take to process the request.

The CEATsoRequest API supports the following request types:

- **CeaTsoStart.** Start a TSO/E address space.
- **CeaTsoAttn.** Send an attention interrupt to a TSO/E address space started by CEA.
- **CeaTsoEnd.** End a TSO/E address space started by CEA.
- **CeaTsoPing.** Ping a TSO/E address space that was started by CEA to prevent the address space from ending because it has been idle too long.
- **CeaTsoQuery.** Obtain information about a specific TSO/E address space started by CEA.
- **CeaTsoQueryApp.** Obtain information about all the TSO/E address spaces that CEA started for an application.

For more details about the request types, see [“Understanding the request types” on page 134](#).

Invoking the CEATsoRequest API

The format to use to call the CEATsoRequest API follows:

```
#include <ceaytsor.h>
#include <ceaxrdef.h>

int32_t CEATsoRequest(CEATsoRequestStruct_t* RequestStruct,
                     CEATsoQueryStruct_t* QueryStruct,
                     CEATsoError_t* ErrorStruct)
```

The call format is the same for each request type. The only difference is the fields that are required for each structure. For a description of each parameter and all the possible fields that can be included in each structure, see [“Parameters” on page 129](#). For a list of the fields that are required for each request type, see [“Understanding the request types” on page 134](#).

The CEATsoRequest API is used as a dynamically loaded library. The file ceasapit.x, which exists in /usr/lib, contains the sidedeck needed to link your program to the DLL. The contents of the file are depicted in [Figure 19 on page 129](#).

```
IMPORT CODE64, 'ceasapit.dll', 'CEATsoRequest'
```

Figure 19. Contents included in the ceasapit.x file

To compile your programs, the following header files are required: ceaytsor.h and ceaxrdef.h. The header files are stored in partitioned data set SYS1.SIEAHDV. The contents of the header files are provided in [“CEAYTSOR header file” on page 162](#) and [“CEAXRDEF header file” on page 165](#).

Parameters

RequestStruct

Pointer to the CEATsoRequestStruct structure. The layout of the CEATsoRequestStruct structure follows:

```
struct CEATsoRequestStruct_s {
    char          ceatso_eyecatcher[8];
```

```

uint32_t    ceatso_version;
uint32_t    ceatso_requesttype;
char        ceatso_userid[8];
uint32_t    ceatso_asid;
char        ceatso_logonproc[8];
char        ceatso_command[80];
uint16_t    ceatso_numqueryreq;
uint16_t    ceatso_numqueryrslt;
uint32_t    ceatso_duration;
uint32_t    ceatso_msgqueueid;
uint16_t    ceatso_charset;
uint16_t    ceatso_codepage;
uint16_t    ceatso_screenrows;
uint16_t    ceatso_screencols;
char        ceatso_account[40];
char        ceatso_group[8];
char        ceatso_region[7];
char        ceatso_instance[1];
char        ceatso_apptag[8];
char        ceatso_stoken[8];
uint32_t    ceatso_ascbaddr;
uint16_t    ceatso_flags;
uint16_t    ceatso_index;
char        ceatso_systemname[8];
CEAconn_t   ceatso_connhandle;
char        rsvd1[64];
};
typedef struct CEATsoRequestStruct_s  CEATsoRequestStruct_t;

```

The fields in the CEATsoRequestStruct structure are explained as follows:

ceatso_eyecatcher

Eye catcher. Specify 'CEAYTSOR'.

ceatso_version

Structure version number.

ceatso_requesttype

Type of request. Specify one of the following values:

- CeaTsoStart
- CeaTsoAttn
- CeaTsoEnd
- CeaTsoPing
- CeaTsoQuery
- CeaTsoQueryApp

For more details about each request type, see [“Understanding the request types” on page 134](#).

ceatso_userid

User ID of the authenticated user for which the TSO/E address space was created.

ceatso_asid

The address space ID (ASID) for the TSO/E address space.

ceatso_logonproc

Name of the TSO/E logon procedure to use to log onto the TSO/E address space.

ceatso_command

Unused.

ceatso_numqueryreq

Maximum number of sessions to query.

ceatso_numqueryrslt

Number of sessions found that satisfy the query.

ceatso_duration

Unused.

ceatso_msgqueueid

The ID of the z/OS UNIX message queue that is used for communications between the caller and the TSO/E session.

ceatso_charset

Character set to use for the caller's TSO/E address space. This value is used by the applications running in the TSO/E address space to convert messages and responses from UTF-8 to EBCDIC. The default character set, which is 697 decimal, will be used if zero is specified as the value.

ceatso_codepage

Codepage to use for the caller's TSO/E address space. This value is used by the applications running in the TSO/E address space to convert messages and responses from UTF-8 to EBCDIC. The default codepage, which is 1047 decimal, will be used if zero is specified as the value.

ceatso_screenrows

Number of rows to be displayed on the screen. The default number of rows, which is 24, will be used if zero is specified as the value.

ceatso_screencols

Number of columns to be displayed on the screen. The default number of columns, which is 80, will be used if zero is specified as the value.

ceatso_account

TSO/E account number.

ceatso_group

TSO/E group name.

ceatso_region

Region size used for the TSO/E address space.

ceatso_instance

Number of active TSO/E address spaces that were started by CEA for the corresponding user ID. In the session table, this value is stored with the oldest TSO/E session entry created for the user.

ceatso_apptag

Identifies the application that is responsible for creating the TSO/E address space.

ceatso_stoken

A token that uniquely identifies the TSO/E address space.

ceatso_ascbaddr

Address of the address space control block that was created for the TSO/E address space.

ceatso_flags

When ending a TSO/E session, you can set the following flags:

- **CEATSO_ABLOGOFF (0x8000)**. If this flag is set, the CANCEL command will be issued to end the TSO/E session regardless of whether the CEA reconnect feature is enabled. Otherwise, the LOGOFF command will be issued or the TSO/E session will be placed in a dormant state as a candidate for reconnection.
- **CEATSO_NORECONN (0x4000)**. If this flag is set, the CEA TSO/E address space manager will end the TSO/E session even if the CEA reconnect feature is enabled. That is, if the client allows users to set this flag, users can force the CEA TSO/E address space manager to end a TSO/E session even if your installation has enabled the reconnect feature. For more information about the reconnect feature, see [“Reconnecting to CEA TSO/E address spaces” on page 127](#).

When starting a TSO/E session, the CEA TSO/E address space manager sets the CEATSO_RECONNECTD (0x2000) flag if the user was connected to a dormant TSO/E session instead of a new session.

ceatso_index

The index value, STOKEN, and ASID together identify the TSO/E address space to the CEA TSO/E address space services.

ceatso_systemname

The system name.

ceatso_connhandle

The connection handle.

rsvd1

Reserved for future use.

QueryStruct

Pointer to the CEATsoQueryStruct structure. This structure is used to return query results for the CeaTsoQuery and CeaTsoQueryApp request types. The layout of the CEATsoQueryStruct structure follows:

```
struct CEATsoQueryStruct_s{
    char        ceatsoq_eyecatcher[8];
    uint32_t    ceatsoq_version;
    uint32_t    ceatsoq_requesttype;
    char        ceatsoq_userid[8];
    uint32_t    ceatsoq_asid;
    char        ceatsoq_logonproc[8];
    char        ceatsoq_command[80];
    uint16_t    ceatsoq_numqueryreq;
    uint16_t    ceatsoq_numqueryrslt;
    uint32_t    ceatsoq_duration;
    uint32_t    ceatsoq_msgqueueid;
    uint16_t    ceatsoq_charset;
    uint16_t    ceatsoq_codepage;
    uint16_t    ceatsoq_screenrows;
    uint16_t    ceatsoq_screencols;
    char        ceatsoq_account[40];
    char        ceatsoq_group[8];
    char        ceatsoq_region[7];
    char        ceatsoq_instance[1];
    char        ceatsoq_apptag[8];
    char        ceatsoq_stoken[8];
    uint32_t    ceatsoq_ascbaddr;
    uint16_t    ceatsoq_flags;
    uint16_t    ceatsoq_index;
    char        rsvd1[8];
};
typedef struct CEATsoQueryStruct_s  CEATsoQueryStruct_t;
```

The fields in the CEATsoQueryStruct structure are explained as follows:

ceatso_eyecatcher

Eye catcher. The value is 'CEAYTSOQ'.

ceatso_version

Structure version number.

ceatso_requesttype

Type of request. The CeaTsoQueryStruct returns results for the CeaTsoQuery and CeaTsoQueryApp request types. For more details about each request type, see [“Understanding the request types” on page 134](#).

ceatso_userid

User ID of the authenticated user for which the TSO/E address space was created.

ceatso_asid

The address space ID (ASID) for the TSO/E address space.

ceatso_logonproc

Name of the TSO/E logon procedure to use to log onto the TSO/E address space.

ceatso_command

Unused.

ceatso_numqueryreq

Maximum number of sessions to query.

ceatso_numqueryrslt

Number of sessions found that satisfy the query.

ceatso_duration

Unused.

ceatso_msgqueueid

The ID of the z/OS UNIX message queue that is used for communications between the caller and the TSO/E session.

ceatso_charset

Character set to use for the caller's TSO/E address space. This value is used by the applications running in the TSO/E address space to convert messages and responses from UTF-8 to EBCDIC. The default character set, which is 697 decimal, will be used if zero is specified as the value.

ceatso_codepage

Codepage to use for the caller's TSO/E address space. This value is used by the applications running in the TSO/E address space to convert messages and responses from UTF-8 to EBCDIC. The default codepage, which is 1047 decimal, will be used if zero is specified as the value.

ceatso_screenrows

Number of rows to be displayed on the screen. The default number of rows, which is 24, will be used if zero is specified as the value.

ceatso_screencols

Number of columns to be displayed on the screen. The default number of columns, which is 80, will be used if zero is specified as the value.

ceatso_account

TSO/E account number.

ceatso_group

TSO/E group name.

ceatso_region

Region size used for the TSO/E address space.

ceatso_instance

Number of active TSO/E address spaces that were started by CEA for the corresponding user ID. In the session table, this value is stored with the oldest TSO/E session entry created for the user.

ceatso_apptag

Identifies the application that is responsible for creating the TSO/E address space.

ceatso_stoken

A token that uniquely identifies the TSO/E address space.

ceatso_ascbaddr

Address of the address space control block that was created for the TSO/E address space.

ceatso_flags

When ending a TSO/E session, you can set the following flags:

- **CEATSO_ABLOGOFF (0x8000)**. If this flag is set, the CANCEL command will be issued to end the TSO/E session regardless of whether the CEA reconnect feature is enabled. Otherwise, the LOGOFF command will be issued or the TSO/E session will be placed in a dormant state as a candidate for reconnection.
- **CEATSO_NORECONN (0x4000)**. If this flag is set, the CEA TSO/E address space manager will end the TSO/E session even if the CEA reconnect feature is enabled. That is, if the client allows users to set this flag, users can force the CEA TSO/E address space manager to end a TSO/E session even if your installation has enabled the reconnect feature. For more information about the reconnect feature, see [“Reconnecting to CEA TSO/E address spaces” on page 127](#).

When starting a TSO/E session, the CEA TSO/E address space manager sets the CEATSO_RECONNECTD (0x2000) flag if the user was connected to a dormant TSO/E session instead of a new session.

ceatso_index

The index value, STOKEN, and ASID together identify the TSO/E address space to the CEA TSO/E address space services.

rsvd1

Reserved for future use.

ErrorStruct

Pointer to the CEATsoErrorStruct structure. This structure contains information about the results of the request. The layout of the CEATsoErrorStruct structure follows:

```
struct CEATsoError_s {
    char          eyeCatcher[8];
    uint32_t      version;
    int32_t       returnCode;
    uint32_t      reasonCode;
    CEATsoDiag_t  diag;
};
typedef struct CEATsoError_s CEATsoError_t;
```

The fields in the CEATsoErrorStruct structure are explained as follows:

eyeCatcher

Eye catcher. Specify 'CEAIERRO'.

version

Structure version number.

returnCode

Return code. For more information about return codes, see [“Return codes” on page 148](#).

reasonCode

Reason code. For more information about reason codes, see [“Reason codes” on page 149](#).

diag

Diagnostic codes, which are mapped by a CEATsoDiag_t structure. This structure can contain up to four diagnostic codes that provide more details about the failure. For more information about diagnostic codes, see [“Diagnostic codes” on page 158](#).

Requirements for callers

To send requests to the API, the environment of the caller must satisfy the following requirements:

- **Minimum authorization:** Problem state
- **Dispatchable unit mode:** Task
- **Cross memory mode:** PASN=HASN=SASN
- **AMODE:** 64-bit
- **ASC mode:** Primary
- **Interrupt status:** Enabled for I/O and external interrupts
- **Locks:** No locks held
- **Linkage:** Uses standard C linkage conventions
- **Library path (LIBPATH):** Must be set to include /usr/lib

Understanding the request types

This section describes the request types that are provided by the CEATsoRequest API. For a description of the API, including the call format and parameters, see [“Invoking the CEATsoRequest API” on page 129](#).

CeaTsoStart - Starting a new TSO/E session

Use the CeaTsoStart request type to start a new TSO/E address space or to reconnect to a dormant TSO/E session. When you start a new TSO/E address space, a z/OS UNIX message queue is also created to enable communication between the caller and the TSO/E address space. When you reconnect to a TSO/E session, the existing message queue is reused.

The TSO/E address space is started or reconnected to using the security environment of the caller. If there is task-level security, it is used for the address space. Otherwise, the address space security environment

is used. The user tokens (UTOKENs) from both environments are saved and are used to verify subsequent requests.

Table 17 on page 135 lists the input callers must provide for each structure used for this request type and the output that will be provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields listed for each structure, see [“Parameters” on page 129](#).

Table 17. Input and output for each structure used for the CeaTsoStart request type		
Structure	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version • ceatso_requesttype • ceatso_logonproc • ceatso_charset • ceatso_codepage • ceatso_screenrows • ceatso_screencols • ceatso_account • ceatso_group • ceatso_region • ceatso_apptag 	<p>If the return code is CEASUCCESS, the following fields are returned:</p> <ul style="list-style-type: none"> • ceatso_userid • ceatso_asid • ceatso_msgqueueid • ceatso_stoken • ceatso_index • ceatso_flags. The value is <i>tsor_reconnected</i> if the CEA TSO/E address space manager connected the user to a dormant TSO/E session.
CeaTsoQueryStruct	Not used for this request type.	Not used for this request type.
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

CeaTsoAttn - Sending an attention interrupt to a TSO/E session

Use the CeaTsoAttn request type to send an attention interrupt to a TSO/E address space started by CEA. An attention interrupt allows you to interrupt or end a process that is taking place. This request type is useful if the client is stuck at a prompt or if you submitted a request to which the system is not responding.

To perform this request, the CEA TSO/E address space manager extracts the caller's security UTOKEN from the caller's environment and uses it when needed.

Table 18 on page 136 lists the input callers must provide for each structure used for this request type and the output that will be provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields listed for each structure, see [“Parameters” on page 129](#).

Note: Create the CEATsoRequest structure with the system name of the session to which you want to send an attention interrupt. This will send an ATTN to the session on that system with the corresponding connection handle.

Table 18. Input and output for each structure used for the CeaTsoAttn request type

Structure	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version • ceatso_requesttype • ceatso_asid • ceatso_apptag • ceatso_stoken • ceatso_index 	None
CeaTsoQueryStruct	Not used for this request type.	Not used for this request type.
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

CeaTsoEnd - Ending a TSO/E session

Use the CeaTsoEnd request type to end a TSO/E address space started by CEA or to place the session into a dormant state. When you end a TSO/E address space, all of the associated resources are returned to the system, including the z/OS UNIX message queue that was used for communicating with the session.

If the CEA reconnect feature is enabled and the caller has not set the CEATSO_ABLOGOFF flag (0x8000) or the CEATSO_NORECONN flag (0x4000), the CEA TSO/E address space manager will intercept the CeaTsoEnd request and place the TSO/E session in a dormant state instead of ending it. In this case, some of the session resources are retained and reused when the user reconnects to the session. For more information about the reconnect feature, see [“Reconnecting to CEA TSO/E address spaces” on page 127](#).

To perform the CeaTsoEnd request, the CEA TSO/E address space manager extracts the caller's security UTOKEN from the caller's environment and uses it when needed.

Table 19 on page 137 lists the input callers must provide for each structure used for this request type and the output that will be provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields listed for each structure, see [“Parameters” on page 129](#).

Note: Create the CEATsoRequest structure with the name of the system to end. Code Version 2 in the version field of the request structure (CEATsoRequestStruct_s). The session that is associated with the connection handle will end.

Table 19. Input and output for each structure used for the CeaTsoEnd request type

Structure	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version • ceatso_requesttype • ceatso_asid • ceatso_apptag • ceatso_stoken • ceatso_index Optional input: <ul style="list-style-type: none"> • ceatso_flags 	None
CeaTsoQueryStruct	Not used for this request type.	Not used for this request type.
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

CeaTsoPing - Sending a ping on behalf of an application

Each TSO/E session has an idle application time that the CEA TSO/E address space manager uses to determine if the application that is associated with the session is active. If the idle application time is 15 minutes, the application is considered to be inactive. In which case, the CEA TSO/E address space manager ends all the CEA-managed TSO/E sessions for that application that have the same application identifier.

To prevent TSO/E sessions from ending because of idle application time, callers can use the CeaTsoPing request type to issue a ping request at least once every 15 minutes. Doing so informs CEA that the application is still active, and causes the CEA TSO/E address space manager to reset the idle application time for all the CEA-managed TSO/E sessions that have the same application identifier.

To perform this request, the CEA TSO/E address space manager extracts the caller's security UTOKEN from the caller's environment and uses it when needed.

Table 20 on page 138 lists the input callers must provide for each structure used for this request type and the output that will be provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields listed for each structure, see [“Parameters” on page 129](#).

Note: Create the CEATsoREquest structure with the name of the system to ping. Use a Version 2 level of the request structure. CEA will ping the sessions with that system name and ping the corresponding sessions on the local system. You cannot use a wildcard for the system name to globally ping all systems everywhere. You must ping each remote system explicitly.

<i>Table 20. Input and output for each structure used for the CeaTsoPing request type</i>		
Structure	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version • ceatso_requesttype • ceatso_asid • ceatso_apptag • ceatso_stoken • ceatso_index 	None
CeaTsoQueryStruct	Not used for this request type.	Not used for this request type.
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

CeaTsoQuery - Querying the TSO/E address spaces

Use the CeaTsoQuery request type to obtain information from the CEA TSO/E address space manager about a TSO/E address space started by CEA. Two levels of the request structure, Version 1 and Version 2, are supported.

To perform this request, the CEA TSO/E address space manager extracts the caller's security UTOKEN from the caller's environment and uses it when needed.

Table 21 on page 138 lists the input that callers must provide for each structure that is used for Version 1 of this request type and the output that is provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields that are listed for each structure, see “Parameters” on page 129.

<i>Table 21. Input and output for each structure that is used for Version 1 of the CeaTsoQuery request type</i>		
Structure, Version 1	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version <p>Note: The version of this structure must match that of CeaTsoQueryStruct. Otherwise, the request will fail.</p> <ul style="list-style-type: none"> • ceatso_requesttype • ceatso_asid • ceatso_apptag • ceatso_stoken • ceatso_index 	None

Table 21. Input and output for each structure that is used for Version 1 of the CeaTsoQuery request type (continued)

Structure, Version 1	Required Input	Output
CeaTsoQueryStruct	<ul style="list-style-type: none"> • eyecatcher • ceatsoq_version <p>Note: The version of this structure must match that of CeaTsoRequestStruct. Otherwise, the request will fail.</p>	<p>If the return code is CEASUCCESS, the following fields are returned:</p> <ul style="list-style-type: none"> • ceatsoq_userid • ceatsoq_asid • ceatsoq_logonproc • ceatsoq_msgqueueid • ceatsoq_charset • ceatsoq_codepage • ceatsoq_screenrows • ceatsoq_screencols • ceatsoq_account • ceatsoq_group • ceatsoq_region • ceatsoq_apptag • ceatsoq_stoken • ceatsoq_index
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

Table 22 on page 140 lists the input that callers must provide for each structure that is used for Version 2 of this request type and the output that is provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields that are listed for each structure, see “Parameters” on page 129.

Table 22. Input and output for each structure that is used for Version 2 of the CeaTsoQuery request type

Structure, Version 2	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version <p>Note: The version of this structure must match that of CeaTsoQueryStruct. Otherwise, the request will fail.</p> <ul style="list-style-type: none"> • ceatso_requesttype • ceatso_connhandle • ceatsor_flags <p>Note: This field must be cleared.</p> <ul style="list-style-type: none"> • ceatsor_systemname <p>Note: This field must be cleared.</p>	None
CeaTsoQueryStruct	<ul style="list-style-type: none"> • eyecatcher • ceatsoq_version <p>Note: The version of this structure must match that of CeaTsoRequestStruct. Otherwise, the request will fail.</p>	<p>If the return code is CEASUCCESS, the following fields are returned:</p> <ul style="list-style-type: none"> • ceatsoq_userid • ceatsoq_asid <p>Note: This is the address space identifier of the address space that was created on the system that is indicated by ceatso_systemname.</p> <ul style="list-style-type: none"> • ceatsoq_logonproc • ceatsoq_charset • ceatsoq_codepage • ceatsoq_screenrows • ceatsoq_screencols • ceatsoq_account • ceatsoq_group • ceatsoq_region • ceatsoq_apptag • ceatsoq_stoken • ceatsoq_index • ceatsoq_systemname <p>Note: This is the system on which the session or address space was created.</p> <ul style="list-style-type: none"> • ceatsoq_connhandle

Table 22. Input and output for each structure that is used for Version 2 of the CeaTsoQuery request type (continued)

Structure, Version 2	Required Input	Output
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

CeaTsoQueryApp - Querying TSO/E sessions by application

Use the CeaTsoQueryApp request type to obtain information from the CEA TSO/E address space manager about all the TSO/E address spaces that CEA started that are associated with a specific application identifier. Two levels of the request structure, Version 1 and Version 2, are supported.

To perform this request, the CEA TSO/E address space manager extracts the caller's security UTOKEN from the caller's environment and uses it when needed.

Table 23 on page 141 lists the input that callers must provide for each structure that is used for Version 1 of this request type and the output that is provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields that are listed for each structure, see “Parameters” on page 129.



Attention: It is the caller's responsibility to free the storage that is associated with the query structures that are returned.

Table 23. Input and output for each structure used for Version 1 of the CeaTsoQueryApp request type

Structure, Version 1	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version • ceatso_requesttype • ceatso_numqueryreq • ceatso_apptag 	<p>If the return code is CEASUCCESS, the following field is returned:</p> <ul style="list-style-type: none"> • ceatso_numqueryrslt

Table 23. Input and output for each structure used for Version 1 of the CeaTsoQueryApp request type (continued)

Structure, Version 1	Required Input	Output
CeaTsoQueryStruct	None	<p>If the return code is CEASUCCESS, an array of query structures are allocated and the following fields are returned for each:</p> <ul style="list-style-type: none"> • eyecatcher • ceatsoq_version • ceatsoq_userid • ceatsoq_asid • ceatsoq_logonproc • ceatsoq_msgqueueid • ceatsoq_charset • ceatsoq_codepage • ceatsoq_screenrows • ceatsoq_screencols • ceatsoq_account • ceatsoq_group • ceatsoq_region • ceatsoq_apptag • ceatsoq_stoken • ceatsoq_index
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

Table 24 on page 142 lists the input that callers must provide for each structure that is used for Version 2 of this request type and the output that is provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields that are listed for each structure, see “Parameters” on page 129.



Attention: It is the caller's responsibility to free the storage that is associated with the query structures that are returned.

Table 24. Input and output for each structure used for Version 2 of the CeaTsoQueryApp request type

Structure, Version 2	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version • ceatso_requesttype • ceatso_numqueryreq • ceatso_flags • ceatso_systemname • ceatso_apptag 	<p>If the return code is CEASUCCESS, the following field is returned:</p> <ul style="list-style-type: none"> • ceatso_numqueryrslt

Table 24. Input and output for each structure used for Version 2 of the CeaTsoQueryApp request type (continued)

Structure, Version 2	Required Input	Output
CeaTsoQueryStruct	None	<p>If the return code is CEASUCCESS, an array of query structures are allocated and the following fields are returned for each:</p> <ul style="list-style-type: none"> • eyecatcher • ceatsoq_version • ceatsoq_userid • ceatsoq_asid • ceatsoq_logonproc • ceatsoq_charset • ceatsoq_codepage • ceatsoq_screenrows • ceatsoq_screencols • ceatsoq_account • ceatsoq_group • ceatsoq_region • ceatsoq_apptag • ceatsoq_stoken • ceatsoq_index • ceatsoq_connhandle • ceatsoq_systemname
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

Invoking the CEAMsgsnd API

This API provides the caller with the ability to send data to the TSO/E address space that was created with a prior CEATsoRequest(requettype=start) invocation.

To enable easy conversion from the msgsnd operation to CEA processing, the new CEAMsgsnd API uses many of the same parameters.

```
Int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

The format to use to call the CEAMsgsnd API follows:

```
int32_t CEAMsgsnd(CEAconn_t *connecthandle,
                  CEAMsgopt_t *options,
                  const void *msgp,
                  size_t msgsz,
                  int msgflg,
                  CEATsoError_t *ceaerro);
```

This API is intended to be used to send information to the TSO/E address space that is associated with the connection handle. If an existing application is to be converted to use this API, it is intended to be used in place of `msgsnd` with minor modifications as described in the parameters descriptions.

Parameters

connecthandle

Pointer to a connection handle which is returned by the `CEATsoRequest(requesttype=Start)` that created the TSO/E address space. This is the handle to the session to which data will be sent. This opaque object requires no manipulation by the caller.

Direction: Input

options

Pointer to the option structure which is constructed by the caller and used to guide specific behaviors for CEA in performance of the API function. This structure, although required, is reserved by IBM and is not used. It is expected that the structure is provided and completely initialized to zeros.

Direction: Input

msgp

Pointer to the structure which is the buffer where the message to be sent is to be placed. This buffer structure is the same format as required by the `msgsnd()` function. The caller must allocate storage for this structure.

Direction: Input

msgsz

The value is the size of the buffer to which the sent data is placed.

Direction: Input

msgflg

The message flag field contains the value of flags as defined by z/OS UNIX services. It is used by the message services in the same manner as the `msgsnd()` operation.

Direction: Input

ceaerro

Pointer to the structure which contains the results of the call to the services. It is expected that the caller:

- Allocates the storage for this structure
- Sets the eyecatcher
- Sets the version

The `ceaerro` parameter can return z/OS UNIX `errno` and `errnojrs`. Definitions of the `#defines` for these values are in the header file `ceaytsor.h`.

Direction: Input/Output

Requirements for callers

To send requests to the API, the environment of the caller must satisfy the following requirements, which are the same as those for `CEATsoRequest()`.

- **Minimum authorization:** Problem state
- **Dispatchable unit mode:** Task
- **Cross memory mode:** PASN=HASN=SASN
- **AMODE:** 64-bit
- **ASC mode:** Primary
- **Interrupt status:** Enabled for I/O and external interrupts

- **Locks:** No locks held
- **Linkage:** Uses standard C linkage conventions
- **Library path (LIBPATH):** Must be set to include /usr/lib

Restrictions

None defined.

Examples

SYS1.SAMPLIB(CEASAMP)

Invoking the CEAMsgrcv API

This API provides the caller with the ability to receive data from the TSO/E address space created with a prior CEATsoRequest(requesttype=start) invocation. A companion API, CEAMsgsnd, is added to allow the caller to send data to that address space.

To enable easy conversion from the msgrcv operation to CEA processing, the new CEAMsgrcv API uses many of the same parameters.

The format to use to call the CEAMsgrcv API follows:

```
int32_t CEAMsgrcv(CEAconn_t *connecthandle,
    CEAMsgopt_t *options,
    void *msgp,
    size_t msgsz,
    long int msgtype,
    int msgflg,
    CEATsoError_t *ceaerro);
```

This API retrieves messages from the TSO/E address space that is associated with the connection handle. If an existing application is to be converted to use this API, the API can be used in place of msgrcv with minor modifications as described in the parameter descriptions.

Parameters

connecthandle

Pointer to the connection handle which is returned by the CEATsoRequest(requesttype=Start) that created the TSO/E address space the caller would like to receive data from. This is an opaque object that requires no manipulation by the caller.

Direction: Input

options

Pointer to the option structure which is constructed by the caller and used to guide specific behaviors for CEA in performance of the receive function. This structure, although required, is reserved, but not used. It is expected that the structure is provided and completely initialized to zeros.

Direction: Input

msgp

Pointer to the structure which is the buffer where the message to be returned is to be placed. This buffer structure is the same format as required by the function msgrcv(). The caller must allocate storage for this structure.

Direction: Input/Output

msgsz

This value is the size of the buffer where the data from the operation is to be placed.

Direction: Input

msgtype

This value is the type of message that the caller would like the function to retrieve and place into the buffer provided.

Direction: Input

msgflg

The message flag field contains the value of flags as defined by the existing services. It is used by the message services in the same manner as required by the msgrcv operation.

Direction: Input

ceaerro

Pointer to the structure that contains the results of the call to the services. It is expected that the caller:

- Allocates the storage for this structure
- Sets the eyecatcher
- Sets the version

The ceaerro parameter can return z/OS UNIX errno and errnojrs. Definitions of the #defines for these values are in the header file ceaytsor.h.

Direction: Input/Output

Requirements for callers

To send requests to the API, the environment of the caller must satisfy the following requirements, which are the same as those for CEATsoRequest().

- **Minimum authorization:** Problem state
- **Dispatchable unit mode:** Task
- **Cross memory mode:** PASN=HASN=SASN
- **AMODE:** 64-bit
- **ASC mode:** Primary
- **Interrupt status:** Enabled for I/O and external interrupts
- **Locks:** No locks held
- **Linkage:** Uses standard C linkage conventions
- **Library path (LIBPATH):** Must be set to include /usr/lib

Restrictions

None defined.

Examples

SYS1.SAMPLIB(CEASAMP)

Invoking the CEAWSNDT API

This API is used by callers running in a TSO/E address space that was created by a caller using the TSOASMgr services. That is, this address space must have been started by a CEATSORequest() invocation.

To enable easy conversion from the msgsnd operation to CEA processing, the new CEAWSNDT API uses many of the same parameters.

```
Int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

The format to use to call the CEAWSNDT API follows:

```
int32_t CEAWSNDT(
    CEAMsgopt_t **options,
    const void **msgp,
    int msgsz,
    int msgflg,
    CEATsoError_t **ceaerro);
```

This API is intended to be used to send information in JSON format to the TSO/E address space that is associated with the connection handle. If an existing application is to be converted to use this API, it is intended to be used in place of msgsnd with minor modifications as described in the parameter descriptions.

Parameters

options

Pointer to pointer to the option structure which is constructed by the caller and used to guide specific behaviors for CEA in performance of the API function. This structure, although required, is reserved, but not used. It is expected that the structure is provided and completely initialized to zeros.

Direction: Input

msgp

Pointer to pointer to the structure which is the buffer where the message to be sent is to be placed. This buffer structure is the same format as required by the msgsnd() function. The caller must allocate storage for this structure.

Direction: Input

msgsz

The size of the buffer where the data for the operation is to be placed.

Direction: Input

msgflg

The message flag field contains the value of flags as defined by z/OS UNIX services. It is used by the message services in the same manner as required by the msgsnd() operation.

Direction: Input

ceaerro

Pointer to pointer of the structure that contains the results of the call to the services. It is expected that the caller:

- Allocates the storage for this structure
- Sets the eyecatcher
- Sets the version

The ceaerro parameter can return z/OS UNIX errno and errnojrs. Definitions of the #defines for these values are in the header file ceaytsor.h.

Direction: Input/Output

Requirements for callers

To send requests to the API, the environment of the caller must satisfy the following requirements.

- **Minimum authorization:** Problem
- **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
- **Linkage:** SYS1.CSSLIB(CEACSS)

Restrictions

None defined.

Examples

SYS1.SAMPLIB(CEAWAPI)

Return, reason, and diagnostic codes

When the CEATsoRequest API returns control to your program, the CEATsoErrorStruct structure contains the return, reason, and diagnostic codes that you can use to identify more information about any errors that occurred.

The codes the API returns are described in the following sections:

- [“Return codes” on page 148](#)
- [“Reason codes” on page 149](#)
- [“Diagnostic codes” on page 158](#)

Return codes

Table 25 on page 148 lists and describes the return codes that are typically returned after the CEATsoRequest API processes a request.

Table 25. Return codes	
Hexadecimal Return Code	Equate Symbol, Meaning, and Action
FFFFFFFF	<p>Equate symbol: CEAFAILURE</p> <p>Meaning: One or more errors occurred during CEATSOREQUEST processing.</p> <p>Action: Check the reason and diagnostic codes to obtain additional information, and correct any errors.</p>
FFFFFFFE	<p>Equate symbol: CEA_ENVIRONMENTAL_ERROR</p> <p>Meaning: The requested API is not available on this system.</p> <p>Action: Contact system programmer to find the system with the correct level of TSO Address space manager services.</p>

Table 25. Return codes (continued)

Hexadecimal Return Code	Equate Symbol, Meaning, and Action
00000000	<p>Equate symbol: CEASUCCESS</p> <p>Meaning: No errors occurred during CEATSOREQUEST processing. The meaning of a CEASUCCESS return code for each request type follows:</p> <ul style="list-style-type: none"> • CeaTsoStart. A new TSO/E address space was started, or the user was connected to a dormant TSO/E session. The caller can now read from and write to the z/OS UNIX message queue. • CeaTsoAttn. The attention interrupt request was sent to the specified TSO/E address space. • CeaTsoEnd. The specified TSO/E address space was ended or placed into a dormant state. If the session was ended, all associated resources were returned to the system. Otherwise, the resources were retained so that they can be reused when the user reconnects to the session. • CeaTsoPing. The ping request was performed, and the timestamp for the specified TSO/E session was updated. • CeaTsoQuery. The query completed with no errors. • CeaTsoQueryApp. The query by application completed with no errors. An array of query structures were allocated and populated with information about the sessions. <p>Action: None.</p>
00000004	<p>Equate symbol: CEAWARNING</p> <p>Meaning: One or more warnings occurred during CEATSOREQUEST processing.</p> <p>Action: Check the reason and diagnostic codes to obtain additional information, and correct any errors.</p>

Reason codes

Table 26 on page 149 lists and describes the reason codes that are typically returned after the CEATsoRequest API processes a request. Additional reason codes might also be returned from services that obtained an unexpected error. Those reason codes are not listed in the table.

Table 26. Reason codes

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
0452x	<p>Equate symbol: UNIXSDDFailure</p> <p>Meaning: z/OS UNIX service to set the dub default value has failed. As a result, the message queues that are constructed on remote systems cannot be created in the correct security context. CEA cannot process remote TSOASMgr requests.</p> <p>Action: Internal Error. Determine why the z/OS UNIX service has failed. CEA ctraces the errno, errnojr in the dump. Reg 2 also contains the ret code, reg3 has the rsn code.</p>

Table 26. Reason codes (continued)

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
0453x	<p>Equate symbol: BadConnHandle</p> <p>Meaning: An attempt was made to use an unrecognized connection handle. Either the eyecatcher or length is not valid.</p> <p>Action: Internal Error. This is a version mismatch problem with the connection handle or a bad pointer problem with the caller.</p>
0454x	<p>Equate symbol: BadASAElement</p> <p>Meaning: The element on the work queue is not as expected.</p> <p>Action: Internal Error. Contact IBM Support</p>
0455x	<p>Equate symbol: BadCurrentElement</p> <p>Meaning: The element on the work queue is not as expected.</p> <p>Action: Internal Error. Contact IBM Support.</p>
0456x	<p>Equate symbol: AlreadyFree</p> <p>Meaning: Storage block was freed for a second time.</p> <p>Action: Internal Error. Contact IBM Support.</p>
0457	<p>Equate symbol: QueuePermissionErr</p> <p>Meaning: z/OS UNIX found permission errors with queues</p> <p>Action: Internal Error. Contact IBM Support.</p>
118	<p>Equate symbol: CEANOTYETIMPLEMENTED</p> <p>Meaning: The requested function is not available on this system.</p> <p>Action: Contact the system programmer to find a system with the correct level of TSO Address space manager services.</p>
34C	<p>Equate symbol: CEAOBJECTTYPEBADVERSION</p> <p>Meaning: The version of the parameter structure is not supported.</p> <p>Action: Reissue the API with the correct version.</p>
1000	<p>Equate symbol: CEATSOMSGQSERVICEFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: z/OS UNIX message queue processing failed.</p> <p>Action: Ensure that the CEA started task is TRUSTED. For more information about the RACF TRUSTED attribute, see the topic on associating started procedures and jobs with user IDs in <i>z/OS Security Server RACF System Programmer's Guide</i>, and the topic on using started procedures in <i>z/OS Security Server RACF Security Administrator's Guide</i>.</p>

Table 26. Reason codes (continued)

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
1001	<p>Equate symbol: CEATSONOUSERIDFOUND</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: An input user ID value was expected, but not received.</p> <p>Action: Specify a user ID.</p>
1002	<p>Equate symbol: CEATSOMATCHMISSING</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: A user ID was expected, but not found in the session table.</p> <p>Action: Ensure that the user ID, STOKEN, and index specified are valid.</p>
1003	<p>Equate symbol: CEATSOSTOKENMISSING</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: An input STOKEN value was expected, but not received.</p> <p>Action: Specify a STOKEN.</p>
1004	<p>Equate symbol: CEATSOINDEXOUTOFRANGE</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: Input table index is too big or too small for the session table.</p> <p>Action: Specify a valid index. The index for the TSO/E address space should be between 1 and 2000.</p>
1005	<p>Equate symbol: CEATSOSTartFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: CEA could not create a TSO/E address space.</p> <p>Action: Ensure that sufficient system resources are available to create the TSO/E address space, and verify that the user is authorized to create address spaces.</p>
1006	<p>Equate symbol: CEATSOATTNFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: CEA could not issue a TSO/E attention interrupt.</p> <p>Action: Check the diagnostic codes to obtain additional information, and correct any errors.</p>
1007	<p>Equate symbol: CEATSOENDFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: CEA could not end a TSO/E address space.</p> <p>Action: Check the diagnostic codes to obtain additional information, and correct any errors.</p>

Table 26. Reason codes (continued)

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
1008	<p>Equate symbol: CEATSOQUERYFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: An attempt to query the session table failed.</p> <p>Action: Ensure that the input values you specified are valid. If the input values are valid, check the diagnostic codes to obtain additional information. Correct any errors.</p>
1009	<p>Equate symbol: CEATSOQUERYAPPFFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: An attempt to query the session table for the TSO/E sessions that are associated with a specific application failed.</p> <p>Action: Ensure that the application identifier you specified is valid. If the application identifier is valid, check the diagnostic codes to obtain additional information. Correct any errors.</p>
100A	<p>Equate symbol: CEATSOPINGFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: Ping processing failed. Typically, this error occurs when the ping request is not issued from the security environment where the TSO/E address space was started or the user is not authorized to the application identified when the TSO/E address space was created.</p> <p>Note that the TSO/E address space is started or reconnected to using the security environment of the caller. If there is task-level security, it is used for the address space. Otherwise, the address space security environment is used. The user tokens (UTOKENs) from both environments are saved and are used to verify subsequent requests.</p> <p>Action: Issue the ping request from the security environment that was used when the TSO/E address space was started, and ensure that the user is authorized to the application specified when the address space was created.</p>
100B	<p>Equate symbol: CEATSOENDSENDLOGOFFFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The CANCEL command was issued to end the TSO/E address space because the LOGOFF command failed.</p> <p>Action: None.</p>
100C	<p>Equate symbol: CEATSOBadAmode</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The call was invoked in the wrong AMODE. AMODE 64 is required.</p> <p>Action: Invoke the API in AMODE 64.</p>
100D	<p>Equate symbol: CEATSODisabled</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The dispatchable unit is not enabled.</p> <p>Action: Ensure that the dispatchable unit is enabled.</p>

Table 26. Reason codes (continued)

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
100E	<p>Equate symbol: CEATSONotTaskMode</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The CEATsoRequest API was not invoked under task mode. The dispatchable unit mode must be task.</p> <p>Action: Ensure that the dispatchable unit is a task.</p>
100F	<p>Equate symbol: CEATSOFRSet</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The CEATsoRequest API was invoked under a functional recovery routine (FRR). No FRRs are allowed.</p> <p>Action: Ensure that no FRRs are invoked in your environment.</p>
1010	<p>Equate symbol: CEATSOLocked</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The caller is holding a system lock. No system locks are allowed.</p> <p>Action: Release the lock.</p>
1011	<p>Equate symbol: CEATSOXMMMode</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The CEATsoRequest API was invoked while running cross memory mode, which is not allowed. The API must be invoked in primary mode.</p> <p>Action: Invoke the API in primary mode.</p>
1013	<p>Equate symbol: CEATsoReqStructFieldBad</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: Input provided for a field in the CEATsoRequestStruct structure is not valid.</p> <p>Action: To identify the field that is not valid, see the diagnostic codes.</p>
1014	<p>Equate symbol: CEATsoBadQueryEyecatcher</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The eye catcher specified for the query structure is not valid. The expected value is CEAYTSOQ.</p> <p>Action: Specify CEAYTSOQ as the value for the eye catcher field.</p>
1015	<p>Equate symbol: CEATsoBadQueryVersion</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The version specified for the query structure is not valid.</p> <p>Action: Specify a valid version number. The version numbers allowed are specified in the ceaytsor.h header file.</p>
1016	<p>Equate symbol: CEABadCommRequest</p> <p>Meaning: Communications server called w/bad request.</p> <p>Action: Internal error.</p>

Table 26. Reason codes (continued)

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
1017	Equate symbol: CEABADBRANCHFORCOMMSRVR Meaning: Bad branchabend for non-owned request to server Action: Internal error.
1018	Equate symbol: CEACOMMServerABENDED Meaning: Communications queue task error Action: Internal error.
101A	Equate symbol: CEAXCFSENDPROBLEM Meaning: Unable to use XCF services Action: Verify that XCF services are available in the sysplex between the two systems involved in the call.
101B	Equate symbol: CEANOTTARGETSYSTEM Meaning: Unable to find system requested. Action: Specify your request with the correct name of the target system and try again.
101c	Equate symbol: CEAXCFFAILURE Meaning: Failure with XCF services Action: XCF services are unavailable. Consult your installation.
101D	Equate symbol: CEAMSGTYPEERROR Meaning: Message received with unknown type Action: Internal error.
101E	Equate symbol: CEAXCFRECVPROBLEM Meaning: Unexpected receive failure return code on XCF receive. Action: Internal error. Check the CEA CTRACE. Determine the status of XCF services.
101F	Equate symbol: CEAXCFRCVNONE Meaning: No messages received from XCF Action: Retry the request. It is possible the target system is no longer there. Receive returncode and reasoncodes are in diag1 and diag2 respectively.
1020	Equate symbol: CEAXCFRCVFAILURE Meaning: Unusual failure with XCF Receive Action: Determine the status of XCF services.

Table 26. Reason codes (continued)

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
1021	<p>Equate symbol: CEAXCFTIMEOUT</p> <p>Meaning: XCF timed out waiting on response</p> <p>Action: Diag1 has the response code 1 from XCF</p>
1022	<p>Equate symbol: CEAXCFDOWNLEVEL</p> <p>Meaning: XCF down level response</p> <p>Action: Diag1 has response code 1 from XCF; Diag2 has response code 2 from XCF</p>
1023	<p>Equate symbol: CEANOTARGET</p> <p>Meaning: No target system for XCF request</p> <p>Action: Most likely cause is the system is not online, or CEAXCF server is not running on that system. Diag1 has response code 1 from XCF; Diag2 has response code 2 from XCF.</p>
1024	<p>Equate symbol: CEAXCFUNEXPECTED</p> <p>Meaning: Unexpected XCF condition</p> <p>Action: Diag1 has response code 1 from XCF; Diag2 has response code 2 from XCF</p>
1025	<p>Equate symbol: CEANOREMOTEAUTH</p> <p>Meaning: No Authority to launcher on remote system.</p> <p>Action: Check security sessions of the caller. Diag1 has RACF RC1; Diag2 has SAF RC; Diag2 has SAF RSN code</p>
1026	<p>Equate symbol: CEAGETSECURITYOBJECFAILED</p> <p>Meaning: Cannot get local security object</p> <p>Action: Diag1 has RACF RC1; Diag2 has SAF RC; Diag2 has SAF RSN code</p>
1027	<p>Equate symbol: CEAREMOTEREQUESTNOTACCEPTED</p> <p>Meaning: Request not recognized for remote processing</p> <p>Action: The requested action can not be performed on the target (remote) system, adjust the application accordingly.</p>
1028	<p>Equate symbol: CEABADLAUNCHREQUEST</p> <p>Meaning: Internal PC request code not recognized.</p> <p>Action: Internal error.</p>
1029	<p>Equate symbol: CEATSOBadRequestVersion</p> <p>Meaning: Expected V2 request, not V1.</p> <p>Action: Specify the request again with the correct version.</p>

Table 26. Reason codes (continued)

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
102A	Equate symbol: CEAUNIXSERVICEFAILED Meaning: z/OS UNIX service failed. Action: , check diag1 and diag2 for the z/OS UNIX errno and errnoJr.
102B	Equate symbol: CEATSOMSGSENDFAILED Meaning: Problems with the ENQ environment. Action: Diag1 indicated ENQ or DEQ; Diag2 indicates method returncode
102C	Equate symbol: CEASESSIONLOOKUPFAILED Meaning: User is not known to TSOASMGR Action: Specify the request with a known user.
102D	Equate symbol: CEASESSIONQUERYFAILED Meaning: User is not known to the TSOASMGR. Action: Specify the request with a known user.
102E	Equate symbol: CEANOREMOTEID Meaning: Unable to create remote security identity. The userid is not known on that system. Action: Diag1 has RACF rc; diag2 SAF RC; diag2 SAF rsn
102F	Reserved
1030	Equate symbol: CEASENDBADENV Meaning: Recovery entered for CEAsndmsg Action: Internal error.
1031	Equate symbol: CEAXCFSEND2PROBLEM Meaning: Unable to send a msgsnd() to remote Action: Check for other trace entries in the CEA CTRACE.
1032	Equate symbol: CEAXCFRECV2PROBLEM Meaning: Unable to send a msgrcv() to remote Action: Check for other trace entries in the CEA CTRACE
1033	Reserved
1034	Equate symbol: CEATSOWRAPPERERROR Meaning: Recovery entered for wrapper modules for TSO Action: Internal error.

Table 26. Reason codes (continued)

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
1035	<p>Equate symbol: CEABADCONNHANDLE</p> <p>Meaning: Connection handle bad and unusable.</p> <p>Action: Specify request with a good connection handle.</p>
1036	<p>Equate symbol: CEABADCALLER</p> <p>Meaning: Send service was called in an incorrect state.</p> <p>Action: Internal error</p>
1037	<p>Equate symbol: CEANODATAINTBUFF</p> <p>Meaning: Internal buffer error in CEA's XCF Communication task.</p> <p>Action: Internal error</p>
1038	<p>Equate symbol: CEANOANSAREA</p> <p>Meaning: Internal buffer error in answer area in the COMM task.</p> <p>Action: Internal error.</p>
1039	<p>Equate symbol: CEABADLOCALSECURITYID</p> <p>Meaning: The security identifier for the security database on this system is not available.</p> <p>Action: This most likely is due to the SAFDFLT profile in the REALM class not be appropriately established. See sample security setup JCL CEASEC member. Consult Security administrator for proper setup for your installation.</p>
103A	<p>Equate symbol: CEATARGETSECBAD</p> <p>Meaning: The security identifier for the security database on the remote system is not available.</p> <p>Action: This most likely is due to the SAFDFLT profile in the REALM class not be appropriately established. See sample security setup JCL CEASEC member. Consult Security administrator for proper setup for your installation.</p>
103B	<p>Equate symbol: CEASECMISMATCH</p> <p>Meaning: The system has determined that the security database being used is not the same on both systems involved in the remote TSOASMGR conversation.</p> <p>Action: You are not able to create a session on the remote system. Please consult your security administrator if you believe this is in error.</p>
103c	<p>Equate symbol: CEABADMSGOPTIONS</p> <p>Meaning: The data structure has been filled out incorrectly. For the initial release of the structure, it should be zero filled.</p>

<i>Table 26. Reason codes (continued)</i>	
Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
103D	Equate symbol: CEARCVBADENV Meaning: Recovery entered for the CEArcvmsg() execution. Action: Internal Error.
103E	Equate symbol: CEAXCFSEND3PROBLEM Meaning: CEA Unable to send a message to itself on another system. Action: Internal Error. Contact IBM Support.
103F	Equate symbol: CEATSOBADQUERYRESULT Meaning: When requesting a QUERY or QUERYAPP request, the version numbers of the TSOResult parameter and the QueryResult parameter must be the same. A version 2 TSOResult structure must use a version 2 QueryResult structure. Action: Adjust caller code.
1040	Equate symbol: CEAFailDELETE Meaning: Unable to delete the user ACEE on cleanup Action: Internal Error. Contact IBM Support.
1041	Equate symbol: UNIXUNDUBFAIL Meaning: Unable to undub the UNIX environment to change security environment. Action: Internal Error. Contact IBM Support.

Diagnostic codes

Table 27 on page 159 lists and describes the diagnostic codes that are typically returned after the CEATsoRequest API processes a request. Additional diagnostic codes might also be returned from services that obtained an unexpected error. Those diagnostic codes are not listed in the table.

Table 27. Diagnostic code

Hexadecimal Diagnostic Code	Equate Symbol and Meaning
04	<p>Equate symbol: kCEATsoBadRacRouteExtr</p> <p>Meaning: The TSO/E address space was not started because an error occurred while trying to authenticate the caller. The CEA TSO/E address space service could not complete one of the following actions:</p> <ul style="list-style-type: none"> • Extract the security identity of the caller. • Log the caller into TSO/E. • Authorize the caller to a required resource. <p>The following fields are returned in the CEATsoErrorStruct structure:</p> <ul style="list-style-type: none"> • diag2 contains the SAF return code from RACRoute returned in R15. • diag3 contains the RACF or installation return code from the SAF parameter list. • diag4 contains the RACF or installation exit reason code from the SAF parameter list. <p>Note that a value is not always returned in diag2, diag3, and diag4.</p>
05	<p>Equate symbol: kCEATsoBadRacRouteCreate</p> <p>Meaning: An error was encountered when requesting verification of the newly created security identity.</p> <p>The following fields are returned in the CEATsoErrorStruct structure:</p> <ul style="list-style-type: none"> • diag2 contains the SAF return code from RACRoute returned in R15. • diag3 contains the RACF or installation return code from the SAF parameter list. • diag4 contains the RACF or installation exit reason code from the SAF parameter list.
0A	<p>Equate symbol: kCEATsoBadAddSession</p> <p>Meaning: Unable to create a new TSO/E address space.</p> <p>The return code received from the TSO/E session is provided in the diag2 field of the CEATsoErrorStruct structure. If the value in the diag2 field is 15, this indicates that the CEA TSO/E address space manager has created the maximum number of TSO/E sessions allowed. In that case, a message is also written to the z/OS console indicating that the session limit has been reached and user requests cannot be processed.</p>
0B	<p>Equate symbol: kCEATsoBadQuerySession</p> <p>Meaning: Unable to query the attributes of TSO/E sessions that are associated with a specific application.</p> <p>The return code received from the method is provided in the diag2 field of the CEATsoErrorStruct structure.</p>

Table 27. Diagnostic code (continued)

Hexadecimal Diagnostic Code	Equate Symbol and Meaning
0C	<p>Equate symbol: kCEATsoBadASCBStoken</p> <p>Meaning: Unable to issue an attention interrupt or query the session table for information about the TSO/E address space because the STOKEN could not be found.</p>
0D	<p>Equate symbol: kCEATsoBadSessIndex</p> <p>Meaning: The value provided in the ceatso_index field in the CeaTsoRequestStruct is zero, which is not valid. The index must be greater than or equal to one.</p>
0F	<p>Equate symbol: kCEATsoBadLOGONMGCRE</p> <p>Meaning: The MGCRE service used to issue the start command to start a TSO/E address space failed.</p> <p>The register where MGCRE returned its return code is provided in the diag2 field of the CEATsoErrorStruct structure. In this case, the value in the diag2 field is R15 (register 15).</p>
10	<p>Equate symbol: SESS_SESSIONNOLONGERINTABLE</p> <p>Meaning: The TSO/E session no longer exists in the session table.</p>
11	<p>Equate symbol: kCEATsoBadSessENQreq</p> <p>Meaning: Unable to acquire the ENQ on the session table.</p> <p>The return code received from the method is provided in the diag2 field of the CEATsoErrorStruct structure.</p>
13	<p>Equate symbol: kCEATsoBadSessUpdateLastRef</p> <p>Meaning: The ping request failed because the CEA TSO/E address space manager was unable to update the last reference timestamp for that session.</p> <p>The return code received from the method is provided in the diag2 field of the CEATsoErrorStruct structure.</p>
14	<p>Equate symbol: kCEATsoBadQuerySessionForApptag</p> <p>Meaning: Unable to query the session table for the specified application identifier because an error occurred.</p> <p>The return code received from the method is provided in the diag2 field of the CEATsoErrorStruct structure.</p>
15	<p>Equate symbol: kCEATsoBadNumEntries</p> <p>Meaning: The number of entries found that match the query exceeds the maximum number of sessions that can be queried or exceeds the number of entries the query structure can accommodate.</p> <p>The number of entries found is provided in the diag2 field of the CEATsoErrorStruct structure.</p>

Table 27. Diagnostic code (continued)

Hexadecimal Diagnostic Code	Equate Symbol and Meaning
17/23	Equate symbol: kCEATsoBadAppTag Meaning:
18	Equate symbol: kCEATsoBadResmgrAdd Meaning: Unable to set the end of memory resource manager; an ABEND dump was taken.
19	Equate symbol: kCEATsoBadQueryAllSessions Meaning: Unable to perform a query of all TSO/E sessions in the session table. You must search for a specific TSO/E session, or search for TSO/E sessions that are associated with a specific application identifier. The return code received from the method is provided in the diag2 field of the CEATsoErrorStruct structure.
1A	Equate symbol: kCEATsoBadApptag Meaning: The value contained in the application identifier field is not valid.
1B	Equate symbol: kCEATsoBaduserid Meaning: The value contained in the user ID field is not valid.
1C	Equate symbol: kCEATsoBadlogonproc Meaning: The value contained in the logon procedure field is not valid.
1F	Equate symbol: kCEATsoBadscreenrows Meaning: The number of screen rows specified is out of range. The minimum number of screen rows is 24, and the maximum is 204.
20	Equate symbol: kCEATsoBadscreencols Meaning: The number of screen columns specified is out of range. The minimum number of screen columns is 80, and the maximum is 160.
21	Equate symbol: kCEATsoBadaccount Meaning: The value contained in the account field is not valid.
22	Equate symbol: kCEATsoBadgroup Meaning: The value contained in the TSO/E group name field is not valid.
23	Equate symbol: kCEATsoBadregion Meaning: The value contained in the TSO/E region size field is not valid.
26	Equate symbol: kCEATsoBadCharsetCodepage Meaning: The value contained in the codepage field is not valid because no match was found in the Coded Character Set Identifiers (CCSID) table.

Table 27. Diagnostic code (continued)	
Hexadecimal Diagnostic Code	Equate Symbol and Meaning
27	Equate symbol: kCEATsoBadregionsize Meaning: The value contained in the region size field is not valid because it exceeds the maximum allowable region size of 2,096,128.
28/40	Equate symbol: kCEATsoBadSystemnamechars Meaning:
29/41	Equate symbol: kCEATsoFlagsForQuery Meaning:

CEAYTSOR header file

For the C programmer, include file ceaytsor.h defines the structures, functions, and macros used for the CEATsoRequest API. The header file is stored in partitioned data set SYS1.SIEAHDRV, and contains the following information.

```
#ifndef __ceaytsor__
#define __ceaytsor__

/***** START OF SPECIFICATIONS *****/
*
* DESCRIPTIVE NAME:  CEA TsoRequest structures
*
* ACRONYM:  CEAYTSOR
*
* STRUCT NAME:  None
*
* LABEL PREFIX:  None
*
* COMPONENT ID:  Common Event Adpater (CEA)
*
*****/

/****PROPRIETARY_STATEMENT*****/
/*
/*
/* LICENSED MATERIALS - PROPERTY OF IBM
/* COPYRIGHT IBM CORP. 2011, 2012
/*
/* STATUS= HBB7770
/*
*****/
/****END_OF_PROPRIETARY_STATEMENT*****/
/*
/*01* EXTERNAL CLASSIFICATION: PI
/*01* END OF EXTERNAL CLASSIFICATION:
/*
*****/

/* $Id: ieac1as2.ide, ieapr, osnp_v1r13.5 1.9 12/01/24 17:16:48 $ */

/*****
* FUNCTION:
*
*
* This header file defines the structures, functions
* and macros used for CEATsoRequest() API.
*
* This support requires the setting of _XOPEN_SOURCE_EXTENDED
*
* RESTRICTIONS:
*   None
*
* CHANGE-ACTIVITY:
```

```

*
****END OF SPECIFICATIONS*****/

/*****
Constants
*****/
#define CEATSOREQUEST_CURRENTVERSION 1
#define CEATSOQUERY_CURRENTVERSION 1
#define CEATSOERROR_CURRENTVERSION 1
#define CEATSODIAG_CURRENTVERSION 1
#define CEATSOREQUEST_EYECATCHER "CEAYTSOR"
#define CEATSOQUERY_EYECATCHER "CEAYTSOQ"
#define CEATSOERROR_EYECATCHER "CEAIERRQ"

/*****
CONSTANTS ceatso_requesttype;
These are the request types used in the CEATsoRequest structure
*****/
#define CeaTsoStart 1
#define CeaTsoEnd 2
#define CeaTsoQuery 3
#define CeaTsoAttn 4
#define CeaTsoPing 5
#define CeaTsoQueryApp 6

/*****
CONSTANTS ceatso_flags
These are the flag values used in the CEATsoRequest structure
*****/
#define CEATSO_ABLGLOGOFF 0x8000 // Use Cancel to end the TSO session
#define CEATSO_NOREUSE 0x4000 // Do not reconnect an existing session

/*****
CEATsoRequestStruct_t

eyeCatcher - "CEAYTSOR"
version - CEATSOQUERY_CURRENTVERSION
request - request - uses CeaTso* constants

*****/

struct CEATsoRequestStruct_s {
    char ceatso_eyecatcher[8]; /* eye catcher: CEAYTSOR */
    uint32_t ceatso_version; /* version number */
    uint32_t ceatso_requesttype; /* which type request */
    char ceatso_userid[8]; /* tso id */
    uint32_t ceatso_asid; /* tso asid */
    char ceatso_logonproc[8]; /* logon proc name */
    char ceatso_command[80]; /* unused */
    uint16_t ceatso_numqueryreq; /* caller num max query */
    uint16_t ceatso_numqueryrslt; /* actual num query */
    uint32_t ceatso_duration; /* unused */
    uint32_t ceatso_msgqueueid; /* msg queue id */
    uint16_t ceatso_charset; /* callers character set */
    uint16_t ceatso_codepage; /* callers code page */
    uint16_t ceatso_screenrows; /* screen rows */
    uint16_t ceatso_screencols; /* screen cols */
    char ceatso_account[40]; /* tso account number */
    char ceatso_group[8]; /* tso group name */
    char ceatso_region[7]; /* tso region size */
    char ceatso_instance[1]; /* tso instance number */
    char ceatso_apptag[8]; /* identity of caller */
    char ceatso_stoken[8]; /* tso asid stoken */
    uint32_t ceatso_ascbaddr; /* tso ascb address */
    uint16_t ceatso_flags; /* tso request flags */
    uint16_t ceatso_index; /* tso session index */
    char rsvd1[8]; /* reserved space */
};
typedef struct CEATsoRequestStruct_s CEATsoRequestStruct_t;

/*****
CEATsoQueryStruct_t*

This structure is used to return Query results for the CEATsoRequest
CeaTsoQuery

eyeCatcher - "CEAYTSOQ"
version - 1

*****/
struct CEATsoQueryStruct_s {
    char ceatsoq_eyecatcher[8]; /* eye catcher: CEAYTSOQ */

```

```

uint32_t      ceatsoq_version;      /* version number      */
uint32_t      ceatsoq_requesttype;  /* which type request  */
char          ceatsoq_userid[8];    /* tso id              */
uint32_t      ceatsoq_asid;         /* tso asid            */
char          ceatsoq_logonproc[8]; /* logon proc name     */
char          ceatsoq_command[80];  /* tso command         */
uint16_t      ceatsoq_numqueryreq;  /* caller num max query */
uint16_t      ceatsoq_numqueryrslt; /* actual num query    */
uint32_t      ceatsoq_duration;     /* duration            */
uint32_t      ceatsoq_msgqueueid;   /* msg queue id       */
uint16_t      ceatsoq_charset;      /* callers character set */
uint16_t      ceatsoq_codepage;     /* callers code page   */
uint16_t      ceatsoq_screenrows;   /* screen rows         */
uint16_t      ceatsoq_screencols;    /* screen cols         */
char          ceatsoq_account[40];  /* tso account number  */
char          ceatsoq_group[8];     /* tso group name      */
char          ceatsoq_region[7];    /* tso region size     */
char          ceatsoq_instance[1];  /* tso instance number */
char          ceatsoq_apptag[8];    /* identity of caller  */
char          ceatsoq_stoken[8];    /* tso asid stoken     */
uint32_t      ceatsoq_ascbaddr;     /* tso ascb address    */
uint16_t      ceatsoq_flags;        /* tso request flags   */
uint16_t      ceatsoq_index;        /* tso session index   */
char          rsvd1[8];             /* reserved space      */
};
typedef struct CEATsoQueryStruct_s CEATsoQueryStruct_t;

/*****
CEATsoDiag_t

version      - version of CEADiag_t
flags        - diagnostic flags
offset       - offset point to additional information
rsvd         - reserved for future use
diag1        - Used to hold return codes
diag2        - from system REXX scripts
diag3        - or other things outside of
diag4        - CEA control
rsvd2        - reserved for future use
messageArea  - Contains any output messages relating to error codes

* This structure is part of CEAError, doesn't get its own eyecatcher
*****/

struct CEATsoDiag_s {
    uint8_t  version;
    uint8_t  flags1;
    uint16_t offset;
    uint8_t  diagid;
    char     rsvd[3];
    uint32_t diag1;
    uint32_t diag2;
    uint32_t diag3;
    uint32_t diag4;
    char     rsvd2[16];
    char     messageArea[256];
    char     rsvd3[256];
};
typedef struct CEATsoDiag_s CEATsoDiag_t;

/*****
CEAError_t

eyeCatcher - "CEAIERRO"
version    - version of CEAError_t
returnCode - function return code - duplicate of function return value
reasonCode - further explanation of a return code.
diag       - further explanation of a reason code.
*****/

struct CEATsoError_s {
    char     eyeCatcher[8];
    uint32_t version;
    int32_t  returnCode;
    uint32_t reasonCode;
    CEATsoDiag_t diag;
};
typedef struct CEATsoError_s CEATsoError_t;

/*****
Function prototype CEATsoRequest
*****/

```



```

#ifdef __cplusplus
extern "C" {
#endif
int32_t CEATsoRequest(CEATsoRequestStruct_t*,
                     CEATsoQueryStruct_t*,
                     CEATsoError_t*);

#ifdef __cplusplus
}
#endif

/*****
Diag Values

These are the possible values that can be returned in the Diag1
field in the CEATsoError_t Diag structure returned from the
CEATsoRequest API

Note: Some duplication of codes exist but codes are unique per API
Request Type
*****/

#define kCEATsoBadRacRouteExtr 0X0004 //0004
#define kCEATsoBadRacRouteCreate 0X0005 //0005
#define kCEATsoBadAddSession 0X000A //0010
#define kCEATsoBadQuerySession 0X000B //0011
#define kCEATsoBadASCBStoken 0X000C //0012
#define kCEATsoBadSessIndex 0X000D //0013
#define kCEATsoBadRemoveSessEntry 0X000E //0014
#define kCEATsoBadLogonMGCRE 0X000F //0015
#define kCEATsoSessionNotFound 0X0010 //0016
#define kCEATsoBadSessENQreq 0X0011 //0017
#define kCEATsoBadSessDEQreq 0X0012 //0018
#define kCEATsoBadSessUpdateLR 0X0013 //0019
#define kCEATsoBadQuerySessApptag 0X0014 //0020
#define kCEATsoBadNumEntries 0X0015 //0021
#define kCEATsoBadMsgQDelete 0X0016 //0022
#define kCEATsoBadAppTag 0X0017 //0023
#define kCEATsoBadWiComCreate 0X0017 //0023
#define kCEATsoBadResmgrAdd 0X0018 //0024
#define kCEATsoBadQueryAllSessions 0X0019 //0025
#define kCEATsoBadApptag 0X001A //0026
#define kCEATsoBaduserid 0X001B //0027
#define kCEATsoBadlogonproc 0X001C //0028
#define kCEATsoBadcharset 0X001D //0029
#define kCEATsoBadcodepage 0X001E //0030
#define kCEATsoBadscreenrows 0X001F //0031
#define kCEATsoBadscreencols 0X0020 //0032
#define kCEATsoBadaccount 0X0021 //0033
#define kCEATsoBadgroup 0X0022 //0034
#define kCEATsoBadregion 0X0023 //0035
#define kCEATsoBadQueryEyecatcher 0X0024 //0036
#define kCEATsoBadQueryVersion 0X0025 //0037
#define kCEATsoBadCharsetCodepage 0X0026 //0038
#define kCEATsoBadregionsize 0X0027 //0039

#endif /* __ceaytsor__ */

```

CEAXRDEF header file

For the C programmer, include file ceaxrdef.h defines the return codes and reason codes that are associated with the CEA TSO/E address space manager services. The header file is stored in partitioned data set SYS1.SIEAHDV, and contains the following information.

```

#ifndef __ceaxrdef__
#define __ceaxrdef__

/***** START OF SPECIFICATIONS *****/
*
* DESCRIPTIVE NAME: CEA reason code definitions
*
* ACRONYM: CEAXRDEF
*
* STRUCT NAME: None
*
* LABEL PREFIX: None
*

```

```

* COMPONENT ID: Common Event Adapter (CEA)
*
*****/

/* $Id: ieac1as2.ide, ieapr, osnp_v1r13.5 1.9 12/01/24 17:16:48 $ */

/****PROPRIETARY_STATEMENT*****/
/*
/*
/* LICENSED MATERIALS - PROPERTY OF IBM
/* COPYRIGHT IBM CORP. 2011, 2012
/*
/* STATUS= HBB7770
/*
/****END_OF_PROPRIETARY_STATEMENT*****/
/*
/*01* EXTERNAL CLASSIFICATION: PI
/*01* END OF EXTERNAL CLASSIFICATION:
/*
/*****/

/*****
* ceaxrdef.h header file
* -----
* This header file defines the reason codes associated with
* the Common Event Adapter (a.k.a. CEAS) client code.
*
*
* CHANGE-ACTIVITY:
*
****END OF SPECIFICATIONS*****/

// Completion Codes
#define CEASUCCESS      0
#define CEAFailure      -1
#define CEAWARNING      4

// Reason Codes
#define CEAUNAVAIL      0x100 //256
#define CEADUPPLICATENAME 0x101 //257
#define CEANOCONNAUTH 0x102 //258
#define CEANOACCESS 0x103 //259
#define CEABADPID 0x104 //260
#define CEABADHANDLE 0x105 //261
#define CEADUPESUB 0x106 //262
#define CEADUPHANDLER 0x107 //263
#define CEANOSUBSCRIBE 0x108 //264
#define CEANOMATCH 0x109 //265
#define CEASALLBUFF 0x10A //266
#define CEANODATA 0x10B //267
#define CEADATATRUNC 0x10C //268 //returned on warning
#define CEAEVENTSMISSED 0x10D //269 //returned on warning
#define CEANOSUBAUTH 0x10E //270
#define CEABADPROTOCOL 0x10F //271
#define CEACOMMFAILURE 0x110 //272
#define CEASYSTEMFAILURE 0x111 //273
#define CEAINVALIDCLIENT 0x112 //274
#define CEASOFTWAREFAILURE 0x113 //275
#define CEABADHANDLEPTR 0x114 //276
#define CEASECURITYFAILURE 0x115 //277
#define CEAINVALIDCOMMAND 0x116 //278
#define CEAMAXCLIENTSCONNECTED 0x117 //279
#define CEANOTYETIMPLEMENTED 0x118 //280
#define CEABADREGVERSION 0x119 //281
#define CEAEFFFAILURE 0x11A //282
#define CEADYNEXFAILURE 0x11B //283
#define CEAEVENTSLOSTTRUNC 0x11C //284
#define CEAUSSSHUTDOWN 0x011D //285
#define CEANOENFEXITRTN 0x011E //286
#define CEASYSOPFORCEUNSUBSCRIBE 0x011F //287
#define CEASYSOPFORCEDISCONNECT 0x0120 //288
#define CEAFORCEMINMODE 0x0121 //289
#define CEAUSSNOTACTIVE 0x0122 //290
#define CEAMAXWTOSUBSCRIBED 0x0123 //291
#define CEAMAXEVENTSSUB 0x0124 //292
#define CEAMAXXSUBCONNECTED 0x0125 //293
#define CEAMAXPGMSUBSCRIBED 0x0126 //294

#define CEANONAME 0x0200 //512
#define CEAINVALIDPDM 0x0201 //513
#define CEABADCONNVERSION 0x0202 //514
#define CEANOTRECOGNIZED 0x0203 //515

```

```

#define CEANOTYPE 0x0204 //516
#define CEABADENFCODE 0x0205 //517
#define CEABADRETVERSION 0x0206 //518
#define CEABADEVENTVERSION 0x0207 //519
#define CEAINVALIDFORM 0x0208 //520
#define CEAINVALIDMODE 0x0209 //521
#define CEAHANDLERNOTFOUND 0x020A //522
#define CEAHANDLERNOTREENT 0x020B //523
#define CEAINVALIDHANDLER 0x020C //524
#define CEACONNECTNOTDEFSEC 0x020D //525
#define CEAEVENTNOTDEFSEC 0x020E //526
#define CEABADCLIENTNAME 0x020F //527
#define CEAINVALIDMSGID 0x0210 //528
#define CEABADADDRESS 0x0211 //529
#define CEAEVENTNOTALPHANUM 0x0212 //530
#define CEAEVENTHASBLANKS 0x0213 //531
#define CEAMAXTHRUPUTREACHED 0x0214 //532
#define CEABADQMASK 0x0215 //533
#define CEABADBITCOMPARE 0x0216 //534
#define CEAMAXENFX 0x0217 //535
#define CEAREJECTENFX 0x0218 //536
#define CEATYPEENFXNOTSUPPORTED 0x0219 //537

#define CEAREQUESTNOTRECOGNIZED 0x0300 //768
#define CEAREQUESTNOTIMPLEMENTED 0x0301 //769
#define CEAPROPERTYSTRUCTBADPTR 0x0302 //770
#define CEAPROPERTYSTRUCTBADEYE 0x0303 //771
#define CEAPROPERTYSTRUCTBADVERSION 0x0304 //772
#define CEAPROPERTYBADRESOURCE 0x0305 //773
#define CEAPROPERTYNOMATCH 0x0306 //774
#define CEAPROPERTYSTRUCTEMPTY 0x0307 //775
#define CEAENVBAD 0x0308 //776
#define CEAFILTERSTRUCTBADEYE 0x0309 //777
#define CEAFILTERSTRUCTBADVERSION 0x030A //778
#define CEAFILTERBADRESOURCE 0x030B //779
#define CEAFILTERNOMATCH 0x030C //780
#define CEABADPARMPTR 0x030D //781
#define CEABADSSISUBSYSTEM 0x030E //782
#define CEABADSSICALL 0x030F //783
#define CEANOS54 0x0310 //784
#define CEABADSSIENV 0x0311 //785
#define CEAENVBADSSI 0x0312 //786
#define CEANOFILTFORVERBOSE 0x0313 //787
#define CEAUNABLETOALLOCATE 0x0314 //788
#define CEANOTJOBSTERSEELEMENT 0x0315 //789
#define CEAJOBCHAINBROKEN 0x0316 //790
#define CEABADDATENV 0x0317 //791
#define CEASYSOUTCHAINBROKEN 0x0318 //792
#define CEANOTSYSOUTHRELEMENT 0x0319 //793
#define CEABADFREETR 0x031A //794
#define CEABADFREETR 0x031B //795
#define CEABADFREETR 0x031C //796
#define CEANABLETOFREE 0x031D //797
#define CEABADIEFQRY 0x031E //798
#define CEASSCHAINBROKEN 0x031F //799
#define CEAENVBADJSQY 0x0320 //800
#define CEABADFILTEROPER 0x0321 //801
#define CEABADS54SUBSYSTEM 0x0322 //802
#define CEABADS54CALL 0x0323 //803
#define CEANOS54 0x0324 //804
#define CEABADS54ENV 0x0325 //805
#define CEAENVBAD54 0x0326 //806
#define CEABADS54STOR 0x0327 //807
#define CEATIMEOUTMAXIMUMEXCEEDED 0x0328 //808
#define CEANEEDSYSOUTFILTER 0x0329 //809
#define CEABUFFERTOOLARGE 0x032A //810
#define CEACMDSDIAGRCSET 0x032B //811
#define CEACMDSDAXREXXRCSET 0x032C //812
#define CEANOINSTRAUTH 0x032D //813
#define CEATOOMUCHDATA 0x032E //814
#define CEAFILTERNOTSUPPORTED 0x032F //815
#define CEAPRIMARYTYPEMISMATCH 0x0330 //816
#define CEABADSUBSYSTEM 0x0331 //817
#define CEANABLETOALLOCATE2 0x0332 //818
#define CEABADBUFFER 0x0333 //819
#define CEATIMEOUTLESSTHANMINIMUM 0x0334 //820
#define CEACMDSSYNTAXERROR 0x0335 //821
#define CEACMDSHALTERROR 0x0336 //822
#define CEACMDSUNINITERROR 0x0337 //823
#define CEAFILTERBADCOMBO 0x0338 //824
#define CEACMDSTIMEDOUT 0x0339 //825
#define CEALLREQBLOCKSINUSE 0x033A //826

```

```

#define CEAIPRQCLIENTABENDED 0x033B //827
#define CEAIPRQARGSCANNOTACCESS 0x033C //828
#define CEAPLISTCANNOTACCESS 0x033D //829
#define CEAIPRQSERVERABENDED 0x033E //830
#define CEANOTACTIVE 0x033F //831
#define CEABADIPQSERVERRC 0x0340 //832
#define CEAMEMORYALLOCATION 0x0341 //833
#define CEASDDIREMPTY 0x0342 //834
#define CEAAADDFAILED 0x0343 //835
#define CEAINCIDENTSTRUCTBADEYE 0x0344 //836
#define CEAINCIDENTSTRUCTBADVERSION 0x0345 //837
#define CEAEERRORSTRUCTBADEYE 0x0346 //838
#define CEAEERRORSTRUCTBADVERSION 0x0347 //839
#define CEAINCINAMESTRUCTBADEYE 0x0348 //840
#define CEABADBRANCHFORIPCSSVR 0x0349 //841
#define CEABADENVFORMAR 0x034A //842
#define CEAOBJECTTYPEBADEYE 0x034B //843
#define CEAOBJECTTYPEBADVERSION 0x034C //844
#define CEAPROBNOTYPEBADEYE 0x034D //845
#define CEAPROBNOTYPEBADVERSION 0x034E //846
#define CEAMAXINSTANCENOSUPPORT 0x034F //847
#define CEAPDWKEYSTRUCTBADEYE 0x0350 //848
#define CEADIAGSTRUCTBADVERSION 0x0351 //849
#define CEADAEDSNNOTAVAILABLE 0x0352 //850
#define CEACANTFINDCOUNTRYCODE 0x0353 //851
#define CEACANTFINDBRANCHCODE 0x0354 //852
#define CEABADPARMLIST 0x0355 //853
#define CEABADPARM 0x0356 //854
#define CEAGENPREPAREDSSNFAIL 0x0357 //855
#define CEAREXXENVEERROR 0x0358 //856
#define CEAXREXXERROR 0x0359 //857
#define CEAINTERNALBUFFEROVERRUN 0x035A //858
#define CEABADTIMEOUTPTR 0x035B //859
#define CEABADOUTPUTBUFFERPTR 0x035C //860
#define CEABADOUTPUTBUFFERLENPTR 0x035D //861
#define CEABADERRORPTR 0x035E //862
#define CEARECOVERYFAILURE 0x035F //863
#define CEABADACRO 0x0360 //864
#define CEABADVER 0x0361 //865
#define CEADMPINCIDENTNOTFOUND 0x0362 //866
#define CEAINVALIDINCIDENTKEY 0x0363 //867
#define CEABADERRO 0x0364 //868
#define CEASYSREXXNOTACTIVE 0x0365 //869
#define CEASYSREXXBADENVIRONMENT 0x0366 //870
#define CEAXEJECTIMEOUT 0x0367 //871
#define CEASYSREXXOVERLOADED 0x0368 //872
#define CEADATABADEYE 0x0369 //873
#define CEADATABADVERSION 0x036A //874
#define CEASYSDUMPBADYE 0x036B //875
#define CEASYSDUMPBADVERSION 0x036C //876
#define CEAINCIDENTSTRUCTBADTYPE 0x036D //877
#define CEAMIGLIBNOTAPFAUTH 0x036E //878
#define CEANOSAFOPERLOGSNAP 0x036F //879
#define CEALOGGERNOTAVAIL 0x0370 //880
#define CEABADALLOCNEW 0x0371 //881
#define CEATERSEBADALLOC1 0x0372 //882
#define CEABADIXGCONN 0x0373 //883
#define CEABADIXGBRWSESTART 0x0374 //884
#define CEABADIXGBRWSEREAD 0x0375 //885
#define CEANOSNAPSHOT 0x0376 //886
#define CEAPDWBOBJECTNOTFOUND 0x0377 //887
#define CEAPDWBDIAGDATAEMPTY 0x0378 //888
#define CEAWRONGIBMPMRFORMAT 0x0379 //889
#define CEABADLEVELOFFPREPARATION 0x037A //890
#define CEADAESYPTOMNOTVALID 0x037B //891
#define CEADAESYPTOMNOTFOUND 0x037C //892
#define CEAIPCSERROR 0x037D //893
#define CEASDDIROPENERROR 0x037E //894
#define CEAXMLINITFAILURE 0x037F //895
#define CEAXMLPARSEFAILURE 0x0380 //896
#define CEAXMLTERMFAILURE 0x0381 //897
#define CEAXMLTAGSTOODEEP 0x0382 //898
#define CEAXMLPARMSBADEYE 0x0383 //899
#define CEADATASPACEBADPTR 0x0384 //900
#define CEAPREPREOBJINUSE 0x0385 //901
#define CEAPREPREENQERR 0x0386 //902
#define CEACKSTBADREQ 0x0387 //903
#define CEACKSTBUFLN 0x0388 //904
#define CEACKSTIGGCSICALLABEND 0x0389 //905
#define CEACKSTBADCONTROLBLOCK 0x038A //906
#define CEACKSTINVALIDSIZETYPE 0x038B //907
#define CEACKSTINVALIDALLOCVALUE 0x038C //908

```

```

#define CEACKSTINVALIDIGGCSIENTRY 0X038D //909
#define CEACKSTIGGCSICALLFAIL 0X038E //910
#define CEACKSTUCBSCANFAIL 0X038F //911
#define CEACKSTUCBSCANABND 0X0390 //912

#define CEASETINCFSELBADEYE 0X0393 //915
#define CEASETINCFSELBADVERSION 0X0394 //916
#define CEASETINCFVALBADEYE 0X0395 //917
#define CEASETINCFVALBADVERSION 0X0396 //918
#define CEASETINCFVALDATATRUNC 0X0397 //919
#define CEAMIGRATEDDATASETS 0X0398 //920
#define CEAMIGRATEDDATASETSWHSMERR 0X0399 //921

#define CEATMSGQSERVICEFAILED 0X1000 //4096
#define CEATSONOUSERIDFOUND 0X1001 //4097
#define CEATSOMATCHMISSING 0X1002 //4098
#define CEATSOSTOKENMISSING 0X1003 //4099
#define CEATSOINDEXOUTOFRANGE 0X1004 //4100
#define CEATSOStartFAILED 0X1005 //4101
#define CEATSOATTNFAILED 0X1006 //4102
#define CEATSOENDFAILED 0X1007 //4103
#define CEATSOQUERYFAILED 0X1008 //4104
#define CEATSOQUERYAPPFALLED 0X1009 //4105
#define CEATSOPINGFAILED 0X100A //4106
#define CEATSOENDSENDLOGOFFFAILED 0X100B //4107
#define CEATSOBADAMODE 0X100C //4108
#define CEATSODISABLED 0X100D //4109
#define CEATSONOTTASKMODE 0X100E //4110
#define CEATSOFRSET 0X100F //4111
#define CEATSOLOCKED 0X1010 //4112
#define CEATSOXMMODE 0X1011 //4113
#define CEATSOSESSTBLDSPFAILED 0X1012 //4114
#define CEATSOREQSTRUCTFIELDDBAD 0X1013 //4115
#define CEATSOBADQUERYEYECATCHER 0X1014 //4116
#define CEATSOBADQUERYVERSION 0X1015 //4117
#endif /* __ceaxrdef__ */

```

Programming example

The following example shows how to invoke the CEATsoRequest API from a C program. For a sample compile job that you can use to compile this sample program, see [“Sample compile job” on page 183](#).

```

/*****
/*
/* CEASAMPT.c Sample code to demonstrate the
/* CEATsoRequest() API for CEA HBB7780
/* CEA TSO ADDRESS SPACE MANAGER
/*
/*
/* Classification: Unclassified
/*
/* Copyright: (C) Copyright IBM Corp. 2011, 2012
/* Licensed Materials - Property of IBM
/*
/*
/* Change History:
/* $1.0 20110314 CYL: Initial Version
/* $1.1 20111015 PDA2: Sample Program
/*
*****/

#define _XOPEN_SOURCE
#define _POSIX1_SOURCE 2

#define SESS_SESSIONNOLONGERINTABLE 16
#define SESS_MATCHMISSING 11
#define SESS_INDEXOUTOFRANGE 13
#define kMaximumSessions 50

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <errno.h>

```

```

#include <env.h>
#include <iconv.h>
#include <sys/msg.h>
#include <sys/types.h>
#include <time.h>

#include "ceaytsor.h"
#include "ceaxrdef.h"
#include "ceayinci.h"
void init_expected_values( void );
void init_ceatso_struct( void );
void print_request_struct( void );
void print_query_struct( void );
void print_error_struct( void );
int send_message( void );
int check_message( int, int );
int verify_messages( int, int );
int verify_attn_messages( int, int );
void save_required_members( void );
void init_required_members( void );
void set_required_members( void );

#define NUMVARS 56

struct message_queue_s {
    long int message_type;
    char message_text[200];
};
typedef struct message_queue_s message_queue_t;

int error_counter; /* Total errors */

CEATsoRequestStruct_t ceatso_request;
CEATsoQueryStruct_t ceatso_query;
CEATsoError_t ceatso_error;

char userid[8];
uint32_t asid;
char apptag[8];
uint32_t ascbaddr;
int index_value; /* Save index value */
char stoken[8]; /* Stoken buffer */
char *stoken_ptr; /* Stoken pointer */
char *ptr;

message_queue_t message_queue;
int message_id;
size_t message_size;
char message_text[200];
int wait_seconds; /* Msg receive time */
int sleep_time;

char *tso_cmd_ptr;
char tso_cmd[80] =
    "{\"TSO RESPONSE\":{\"VERSION\":\"0100\",\"DATA\":{\"ALLOC DA\"}}\"";

int32_t expected_rc;
uint32_t expected_rsn;
uint32_t expected_diag1;
uint32_t expected_diag2;
uint32_t expected_diag3;
uint32_t expected_diag4;
uint32_t reason_mask;
int CeaTsoSamp1( void );

int main() {
    int rc; /* Return code */

    CeaTsoSamp1(); /* Invoke the sample code */

    return 0;
}

/*****/
/**
** Routine to initialize the expected return code,
** reason code and diag codes.
**
**/
/*****/
void init_expected_values( void ) {

```

```

    expected_rc      = CEASUCCESS;
    expected_rsn     = 0;
    expected_diag1   = 0;
    expected_diag2   = 0;
    expected_diag3   = 0;
    expected_diag4   = 0;

    return;
}

/*****
**
** Routine to initialize the CEA TSO request structure
** query structure and error structure for API call
**
**
*****/
void init_ceatso_struct( void ) {

    /* Initialize CEA TSO Request structure for CEATsoRequest() */
    memset(&ceatso_request, '\0', sizeof(CEATsoRequestStruct_t));

    strcpy(ceatso_request.ceatso_eyecatcher, CEATSOREQUEST_EYECATCHER);
    ceatso_request.ceatso_version = CEATSOREQUEST_CURRENTVERSION;
    ceatso_request.ceatso_requesttype = 0;

    /*
    */
    ceatso_request.ceatso_asid = 0;

    /*
    */
    strcpy(ceatso_request.ceatso_userid, "IBMUSER ");
    strcpy(ceatso_request.ceatso_logonproc, "OMVS0803");
    memset(&ceatso_request.ceatso_command, ' ', 80);

    /*
    */
    ceatso_request.ceatso_numqueryreq = 12;
    ceatso_request.ceatso_numqueryrslt = 12;
    ceatso_request.ceatso_duration = 0;
    ceatso_request.ceatso_msgqueueid = 0;

    /*
    */
    ceatso_request.ceatso_charset = 697;
    ceatso_request.ceatso_codepage = 1047;
    ceatso_request.ceatso_screenrows = 24;
    ceatso_request.ceatso_screencols = 80;
    memset(ceatso_request.ceatso_account, '0', 40);
    memset(ceatso_request.ceatso_group, ' ', 8);
    strcpy(ceatso_request.ceatso_region, "2000000");

    /*
    */
    memset(ceatso_request.ceatso_instance, ' ', 1);

    /*
    */
    strcpy(ceatso_request.ceatso_apptag, "IZUIS ");
    ceatso_request.ceatso_flags = CEATSO_ABLOGOFF;

    /*
    */
    memset(ceatso_request.ceatso_stoken, 0xFF, 8);
    ceatso_request.ceatso_ascbaddr = 0;

    ceatso_request.ceatso_index = 0;

    /*
    */

```

```

/* Initialize the CEA TSO Query structure for CEATsoRequest() */
memset(&ceatso_query, '\0', sizeof(CEATsoQueryStruct_t));

strcpy(ceatso_query.ceatsoq_eyecatcher, CEATSOQUERY_EYECATCHER);

memset(&ceatso_request.ceatso_command, ' ', 40);

/* Initialize the CEA TSO Error structure for CEATsoRequest() */
memset(&ceatso_error, 0x00, sizeof(CEATsoError_t));

strcpy(ceatso_error.eyeCatcher, CEAINCT_EYE_CEAIERRO);

ceatso_error.version = CEAIERRO_CURRENTVERSION;

return;
}

/*****
**/
/** Routine to print out the CEATsoRequest structure          **/
/** used by CEATsoRequest( ) API.                             **/
/**                                                           **/
*****/
void print_request_struct( void ) {
    int i;

    printf("\n\nCEATsoRequest structure\n\n");

    printf("sizeof(CEATsoRequestStruct_t) = %d\n",
           sizeof(CEATsoRequestStruct_t));

    printf("CeaTsoRequest Eyecatcher      = ");

    ptr = ceatso_request.ceatso_eyecatcher;
    for ( i = 1; i <= 8; i++ )
        printf("%C", *ptr++);
    printf("\n");

    printf("CeaTsoRequest Version              = %d\n",
           ceatso_request.ceatso_version);

    printf("CeaTsoRequest Requesttype          = %d\n",
           ceatso_request.ceatso_requesttype);

    printf("CeaTsoRequest Userid                = ");

    ptr = ceatso_request.ceatso_userid;
    for ( i = 1; i <= 8; i++ )
        printf("%C", *ptr++);
    printf("\n");

    printf("CeaTsoRequest Asid                  = %X\n",
           ceatso_request.ceatso_asid);

    printf("CeaTsoRequest LogonProc             = ");

    ptr = ceatso_request.ceatso_logonproc;
    for ( i = 1; i <= 8; i++ )
        printf("%C", *ptr++);
    printf("\n");

    printf("CeaTsoRequest Command               = ");

    ptr = ceatso_request.ceatso_command;
    for ( i = 1; i <= 40; i++ )
        printf("%C", *ptr++);
    printf("\n");

    printf("CeaTsoRequest Numqueryreq          = %d\n",
           ceatso_request.ceatso_numqueryreq);

    printf("CeaTsoRequest Numqueryrslt         = %d\n",
           ceatso_request.ceatso_numqueryrslt);

    printf("CeaTsoRequest Duration              = %d\n",
           ceatso_request.ceatso_duration);

    printf("CeaTsoRequest Msgqueueid           = %d\n",

```



```

        ceatso_request.cea_tso_msgqueueid);

printf("CeaTsoRequest Charset      = %d\n",
       ceatso_request.cea_tso_charset);

printf("CeaTsoRequest Codepage     = %d\n",
       ceatso_request.cea_tso_codepage);

printf("CeaTsoRequest Screenrows   = %d\n",
       ceatso_request.cea_tso_screenrows);

printf("CeaTsoRequest Screencols    = %d\n",
       ceatso_request.cea_tso_screencols);

printf("CeaTsoRequest Account       = ");

ptr = ceatso_request.cea_tso_account + 32;
for ( i = 1; i < 8; i++)
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoRequest Group          = ");

ptr = ceatso_request.cea_tso_group;
for ( i = 1; i <= 8; i++)
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoRequest Region         = ");

ptr = ceatso_request.cea_tso_region;
for ( i = 1; i <= 7; i++)
    printf("%C", *ptr++);
printf("\n");

ptr = ceatso_request.cea_tso_instance;
printf("CeaTsoRequest Instance      = %C\n", *ptr);

printf("CeaTsoRequest Apptag         = ");

ptr = ceatso_request.cea_tso_apptag;
for ( i = 1; i <= 8; i++)
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoRequest Stoken          = ");

stoken_ptr = ceatso_request.cea_tso_stoken;
for ( i = 1; i <= 8; i++)
    printf("%X ", *stoken_ptr++);
printf("\n");

printf("CeaTsoRequest ASCBaddr        = %8X\n",
       ceatso_request.cea_tso_ascbaddr);

printf("CeaTsoRequest Flags           = %d\n",
       ceatso_request.cea_tso_flags);

printf("CeaTsoRequest Index            = %d\n",
       ceatso_request.cea_tso_index);

printf("\n");

return;
}

/*****/
/**
/** Routine to print out the CEATsoQuery structure
/** used by CEATsoRequest( ) API.
/**
/**
*****/
void print_query_struct( void ) {
    int i;

    printf("\n\nCEATsoQuery structure\n\n");

    printf("sizeof(CEATsoQueryStruct_t) = %d\n",
           sizeof(CEATsoQueryStruct_t));

```

```

printf("CeaTsoQuery    Eyecatcher    = ");

ptr = ceatso_query.ceatsoq_eyecatcher;
for ( i = 1; i <= 8; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery    Version        = %d\n",
       ceatso_query.ceatsoq_version);

printf("CeaTsoQuery    Requesttype    = %d\n",
       ceatso_query.ceatsoq_requesttype);

printf("CeaTsoQuery    Userid         = ");

ptr = ceatso_query.ceatsoq_userid;
for ( i = 1; i <= 8; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery    Asid           = %X\n",
       ceatso_query.ceatsoq_asid);

printf("CeaTsoQuery    LogonProc      = ");

ptr = ceatso_query.ceatsoq_logonproc;
for ( i = 1; i <= 8; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery    Command        = ");

ptr = ceatso_query.ceatsoq_command;
for ( i = 1; i <= 40; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery    Numqueryreq     = %d\n",
       ceatso_query.ceatsoq_numqueryreq);

printf("CeaTsoQuery    Numqueryrslt    = %d\n",
       ceatso_query.ceatsoq_numqueryrslt);

printf("CeaTsoQuery    Duration        = %d\n",
       ceatso_query.ceatsoq_duration);

printf("CeaTsoQuery    Msgqueueid      = %d\n",
       ceatso_query.ceatsoq_msgqueueid);

printf("CeaTsoQuery    Charset         = %d\n",
       ceatso_query.ceatsoq_charset);

printf("CeaTsoQuery    Codepage        = %d\n",
       ceatso_query.ceatsoq_codepage);

printf("CeaTsoQuery    Screenrows      = %d\n",
       ceatso_query.ceatsoq_screenrows);

printf("CeaTsoQuery    Screencols      = %d\n",
       ceatso_query.ceatsoq_screencols);

printf("CeaTsoQuery    Account          = ");

ptr = ceatso_query.ceatsoq_account + 32;
for ( i = 1; i < 8; i++)
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery    Group            = ");

ptr = ceatso_query.ceatsoq_group;
for ( i = 1; i <= 8; i++)
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery    Region           = ");

ptr = ceatso_query.ceatsoq_region;
for ( i = 1; i <= 7; i++)
    printf("%C", *ptr++);
printf("\n");

```

```

ptr = ceatso_query.ceatsoq_instance;
printf("CeaTsoQuery Instance = %C\n", *ptr);

printf("CeaTsoQuery Apptag = ");

ptr = ceatso_query.ceatsoq_apptag;
for ( i = 1; i <= 8; i++)
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery Stoken = ");

stoken_ptr = ceatso_query.ceatsoq_stoken;
for ( i = 1; i < 9; i++)
    printf("%X ", *stoken_ptr++);
printf("\n");

printf("CeaTsoQuery ASCBAddr = %8X\n",
       ceatso_query.ceatsoq_ascbaddr);

printf("CeaTsoQuery Flags = %d\n",
       ceatso_query.ceatsoq_flags);

printf("CeaTsoQuery Index = %d\n",
       ceatso_query.ceatsoq_index);

printf("\n");

return;
}

/*****
**/
/** Routine to print out the CEATsoError structure **/
/** used by CEATsoRequest( ) API. **/
**/
*****/
void print_error_struct( void ) {
    int i;

    printf("\n\nCEATsoError structure\n\n");

    printf("sizeof(CEATsoError_t) = %d\n\n",
           sizeof(CEATsoError_t));

    printf("CEAError Eyecatcher = ");

    ptr = ceatso_error.eyeCatcher;
    for ( i = 1; i <= 8; i++)
        printf("%C", *ptr++);
    printf("\n");

    printf("CEAError Version = %8d\n",
           ceatso_error.version);

    printf("CEAError ReturnCode(hex) = %8X\n",
           ceatso_error.returnValue);

    printf("CEAError ReasonCode(hex) = %8X\n",
           ceatso_error.reasonCode);

    printf("CEAError Diag.diag1(hex) = %8X\n",
           ceatso_error.diag.diag1);

    printf("CEAError Diag.diag2(hex) = %8X\n",
           ceatso_error.diag.diag2);

    printf("CEAError Diag.diag3(hex) = %8X\n",
           ceatso_error.diag.diag3);

    printf("CEAError Diag.diag4(hex) = %8X\n",
           ceatso_error.diag.diag4);

    printf("\n");

    return;
}

/*****/

```

```

/**                                                                    **/
/** Verify messages                                                    **/
/**                                                                    **/
/*****                                                                    */
int verify_messages(int message_id, int wait_seconds ) {
    int rc;
    char *string1;
    char *string2;
    char *string3;
    char *string4;
    char *string5;
    char *string6;

    if ( ceatso_request.ceatso_requesttype == CeaTsoStart ) {
        rc = check_message(message_id, wait_seconds);
        string1 = "LOGON IN PROGRESS";
        if ( rc != 0 || strstr(message_text, string1) == NULL ) {
            printf(" Failed to receive %s message.\n\n", string1);
            return 99;
        }

        rc = check_message(message_id, wait_seconds);
        string2 = "NO BROADCAST MESSAGES";
        if ( rc != 0 || strstr(message_text, string2) == NULL ) {
            printf(" Failed to receive %s.\n\n", string2);
            return 99;
        }

        rc = check_message(message_id, wait_seconds);
        string3 = "READY ";
        if ( rc != 0 || strstr(message_text, string3) == NULL ) {
            printf(" Failed to receive %s prompt.\n\n", string3);
            return 99;
        }

        rc = check_message(message_id, wait_seconds);
        string4 = "HIDDEN";
        string5 = "FALSE";
        if ( rc != 0 ||
            strstr(message_text, string4) == NULL ||
            strstr(message_text, string5) == NULL ) {
            printf(" Failed to receive %s : %s message.\n\n",
                string4, string5 );
            return 99;
        }
    }

    if ( ceatso_request.ceatso_requesttype == CeaTsoAttn ) {
        rc = check_message( message_id, wait_seconds );
        string6 = "ENTER DATA SET NAME OR * -";
        if ( rc != 0 ||
            strstr(message_text, string6) == NULL ) {
            printf(" Failed to receive %s message.\n\n", string6);
            return 99;
        }

        rc = check_message(message_id, wait_seconds);
        string4 = "HIDDEN";
        string5 = "FALSE";
        if ( rc != 0 ||
            strstr(message_text, string4) == NULL ||
            strstr(message_text, string5) == NULL ) {
            printf(" Failed to receive %s : %s message.\n\n",
                string4, string5 );
            return 99;
        }
    }

    return 0;
}

/*****                                                                    */
/**                                                                    **/
/** Verify messages after Attn                                         **/
/**                                                                    **/
/*****                                                                    */
int verify_attn_messages(int message_id, int wait_seconds ) {

```

```

int    rc;
char   *string1;
char   *string2;
char   *string3;

rc = check_message(message_id, wait_seconds);
string1 = "READY ";
if ( rc != 0 || strstr(message_text, string1) == NULL ) {
    printf("    Failed to receive %s prompt after Attn.\n\n",
           string1);
    return 99;
}

rc = check_message(message_id, wait_seconds);
string2 = "HIDDEN";
string3 = "FALSE";
if ( rc != 0
    || strstr(message_text, string2) == NULL
    || strstr(message_text, string3) == NULL ) {
    printf("    Failed to receive %s : %s message.\n\n",
           string2, string3);
    return 99;
}

return 0;
}

/*****
**/
/** Check message text                               **/
**/
*****/
int check_message( int message_id, int wait_seconds ) {
    int         rc;
    size_t      iconv_rc;
    ssize_t     msg_rc;
    iconv_t     cd;
    char        *input_ptr;
    char        *output_ptr;
    size_t      input_msgsize;
    size_t      output_msgsize;
    time_t      wait_time;
    time_t      start_time;
    time_t      receive_time;

    message_size = sizeof(message_queue_t) - sizeof(long int);

    memset(&message_text, '\0', message_size);

    time(&start_time);

    /* -6 should include 2 and 3                               */
    message_queue.message_type = (long int)-6;

    sleep_time = 2;
    msg_rc = 0;

    /* Must include IPC_NOWAIT flag, otherwise could hang      */
    /* the program execution when no msg sending back.         */
    do {
        msg_rc = msgrcv(message_id, &message_queue, message_size,
                        message_queue.message_type, MSG_NOERROR | IPC_NOWAIT);
        sleep( sleep_time);
        wait_time = time(&receive_time) - start_time;
    } while ( wait_time <= wait_seconds && msg_rc <= 0 );

    if ( msg_rc == -1 ) {
        printf("\n\nReceive message failed with\n");
        printf("    msg_rc = %d ", msg_rc);
        printf("    Wait time = %d seconds\n", wait_time);
        printf("    Errno = %X", errno);
        printf("    Errno_Jr = %X\n", __errno2());
        return 99;
    }
    else
        printf("    Received Message in %d seconds.\n",
               wait_time);

    if ( (rc = setenv("_ICONV_UCS2", "D", 1)) != 0 ) {

```

```

        printf("\n      setenv( )      failed with      ");
        printf("      rc = %d      ", rc);
        printf("      Errno = %X      ", errno);
        printf("      Errno_Jr = %X\n\n", __errno2());
        return rc;
    }

    if ( (cd = iconv_open("IBM-1047", "UTF-8")) == (iconv_t)-1 ) {
        printf("      iconv_open( )      failed with      ");
        printf("      Errno = %X      ", errno);
        printf("      Errno_Jr = %X\n\n", __errno2());
        return 99;
    }

    input_ptr    = message_queue.message_text;
    output_ptr   = message_text;

    input_msgsize = msg_rc;
    output_msgsize = msg_rc;

    if ((iconv_rc = iconv(cd, &input_ptr, &input_msgsize, &output_ptr,
        &output_msgsize)) == (size_t)-1 ) {
        printf("      iconv( )      failed with      ");
        printf("      rc = %d      ", iconv_rc);
        printf("      Errno = %X      ", errno);
        printf("      Errno_Jr = %X\n\n", __errno2());
        return 99;
    }

    if ( (rc = iconv_close( cd )) == -1 ) {
        printf("      iconv_close( )      failed with      ");
        printf("      rc = %d      ", rc);
        printf("      Errno = %X      ", errno);
        printf("      Errno_Jr = %X\n\n", __errno2());
        return rc;
    }

    printf("      Reveived Message Type:      %2d\n",
        message_queue.message_type);
    printf("      Reveived Message Length: %d\n", strlen(message_text));
    printf("      Received Message Text: \n");
    printf("      %s\n", message_text);
    printf("\n");

    return 0;
}

/*****/
/**
/** Send TSO command and check the proper message received
/**
/*****/
int send_message( void ) {
    int      rc;
    size_t    iconv_rc;
    iconv_t    cd;
    size_t    input_msgsize;
    size_t    output_msgsize;
    char      *input_ptr;
    char      *output_ptr;

    message_size = sizeof(message_queue_t) - sizeof(long int);
    memset(&message_queue.message_text, '\0', message_size);
    memset(&message_text, '\0', message_size);

    strcpy(message_text, tso_cmd);

    if ( (cd = iconv_open("UTF-8", "IBM-1047")) == (iconv_t)-1 ) {
        printf("      iconv_open( )      failed with      ");
        printf("      Errno = %X      ", errno);
        printf("      Errno_Jr = %X\n\n", __errno2());
        return 99;
    }

    input_ptr    = message_text;
    output_ptr   = message_queue.message_text;

    input_msgsize = strlen(message_text);
    output_msgsize = input_msgsize;

```

```

if ((iconv_rc = iconv(cd, &input_ptr, &input_msgsize, &output_ptr,
                    &output_msgsize)) == (size_t)-1 ) {
    printf("    iconv( ) failed with ");
    printf("    rc = %d ", iconv_rc);
    printf("    Errno = %X ", errno);
    printf("    Errno_Jr = %X\n\n", __errno2());
    return 99;
}

if ( (rc = iconv_close( cd )) == -1 ) {
    printf("    iconv_close( ) failed with ");
    printf("    rc = %d ", rc);
    printf("    Errno = %X ", errno);
    printf("    Errno_Jr = %X\n\n", __errno2());
    return rc;
}

message_queue.message_type = (long int)7;
message_size = strlen(message_queue.message_text);

rc = msgsnd(message_id, &message_queue, message_size, 0);

return rc;
}

/*****
**
** Save some required members of request structure
** for ATTN and END process
**
**
*****/
void save_required_members( void ) {
    int i;

    /* Not required input for End
    if ( ceatso_request.ceatso_requesttype == CeaTsoEnd ) {
        strcpy(userid, ceatso_request.ceatso_userid);
        strcpy(apptag, ceatso_request.ceatso_apptag);
    }

    if ( ceatso_request.ceatso_requesttype == CeaTsoAttn )
        asid = ceatso_request.ceatso_asid;
    */

    asid = ceatso_request.ceatso_asid;

    stoken_ptr = stoken;
    ptr = ceatso_request.ceatso_stoken;
    for ( i = 1; i < 9; i++)
        *stoken_ptr++ = *ptr++;

    ascbaddr = ceatso_request.ceatso_ascbaddr;

    index_value = ceatso_request.ceatso_index;

    /*
    printf("\nSave the following value:\n");
    */

    /* Not required input for End
    if ( ceatso_request.ceatso_requesttype == CeaTsoEnd ) {
        printf("    userid = ");

        ptr = userid;
        for ( i = 1; i <= 8; i++ )
            printf("%C", *ptr++);
        printf("\n");

        printf("    apptag = ");

        ptr = apptag;
        for ( i = 1; i <= 8; i++ )
            printf("%C", *ptr++);
        printf("\n");
    }
    */
}

```

```

printf("    asid          = %X\n", asid);

ptr = ceatso_request.ceatso_stoken;
printf("    stoken        = ");
for ( i = 1; i < 9; i++)
    printf("%X ", *ptr++);
printf("\n");

printf("    ascdaddr       = %X\n", ascbaddr);

printf("    index_value    = %X\n", index_value);

printf("\n");
*/

return;
}

/*****
**
** Initialize some required members of request structure
** for ATTN and END process
**
**
*****/
void init_required_members( void )
{
    int    i;

    memset(ceatso_request.ceatso_eyecatcher, 'F', 8);

    ceatso_request.ceatso_version = 0;

    if ( ceatso_request.ceatso_requesttype == CeaTsoAttn )
        ceatso_request.ceatso_asid = 0;

    /*
    if ( ceatso_request.ceatso_requesttype == CeaTsoEnd ) {
        memset(ceatso_request.ceatso_userid, 'F', 8);
        memset(ceatso_request.ceatso_apptag, 'F', 8);
    }
    */

    memset(ceatso_request.ceatso_stoken, 0xFF, 8);

    ceatso_request.ceatso_ascbaddr = 0;

    ceatso_request.ceatso_index = 0;

    /* Initialize the CEA TSO Error structure for CEATsoRequest() */
    memset(&ceatso_error, 0x00, sizeof(CEATsoError_t));

    return;
}

/*****
**
** Set some required members of request structure back
** to the original value for ATTN and END process
**
**
*****/
void set_required_members( void )
{
    int    i;

    strcpy(ceatso_request.ceatso_eyecatcher, CEATSOREQUEST_EYECATCHER);

    ceatso_request.ceatso_version = CEATSOREQUEST_CURRENTVERSION;

    /*
    if ( ceatso_request.ceatso_requesttype == CeaTsoEnd ) {
        strcpy(ceatso_request.ceatso_userid, userid);
        strcpy(ceatso_request.ceatso_apptag, apptag);
    }
    */

    if ( ceatso_request.ceatso_requesttype == CeaTsoAttn )
        ceatso_request.ceatso_asid = asid;

    stoken_ptr = stoken;

```



```

ptr = ceatso_request.ceatso_stoken;
for ( i = 1; i < 9; i++)
    *ptr++ = *stoken_ptr++;

ceatso_request.ceatso_ascbaddr = ascbaddr;

ceatso_request.ceatso_index = index_value;

/* Initialize the CEA TSO Error structure for CEATsoRequest() */
memset(&ceatso_error, 0x00, sizeof(CEATsoError_t));
strcpy(ceatso_error.eyeCatcher, CEAINCT_EYE_CEAIERRO);
ceatso_error.version = CEAIERRO_CURRENTVERSION;

return;
}

/*****
**/
/** CeaTsoSamp1: Sample code to invoke CEATsoRequest() to start **/
/** a CEA TSo Session send it an Attn interrupt the end the TSO **/
/** session. **/
/** Results are returned in the error structure **/
**/
*****/
int CeaTsoSamp1( ) {
    int i;
    int rc;

    printf("=====\n");
    printf("==          Start CeaTsoRequest( ) Example          ==\n");
    printf("=====\n");
    printf("\n");

    printf("CEATSORequest( ) Start session.\n\n");
    init_ceatso_struct( );
    init_expected_values( );
    ceatso_request.ceatso_requesttype = CeaTsoStart;

    CEATsoRequest(&ceatso_request, &ceatso_query, &ceatso_error);

    if ( ceatso_error.returnValue == expected_rc    &&
        ceatso_error.reasonCode == expected_rsn    &&
        ceatso_error.diag.diag1 == expected_diag1  &&
        ceatso_error.diag.diag2 == expected_diag2  &&
        ceatso_error.diag.diag3 == expected_diag3  &&
        ceatso_error.diag.diag4 == expected_diag4  )
        printf(" Verifying logon messages.\n\n");
    else {
        error_counter = error_counter + 1;
        printf("CEATsoRequest( ) Start session failed.\n\n\n");
        print_error_struct( );
        print_request_struct( );
        printf("\nVariation %d failed.\n\n\n", variation_id);
        printf("\n\n");
        return error_counter;
    }

    wait_seconds = 8;
    message_id = ceatso_request.ceatso_msgqueueid;
    rc = verify_messages( message_id, wait_seconds );

    if ( rc == 0)
        printf("\nCEATsoRequest( ) Start session successful.\n\n");
    else {
        error_counter = error_counter + 1;
        printf("CEATsoRequest( ) Start failed to receive the message ");
        printf("with rc = %d.\n\n\n", rc);
        printf("\nVariation %d failed.\n\n\n", variation_id);
        printf("\n\n");
        return error_counter;
    }

    save_required_members( );

```

```

ceatso_request.ceatso_requesttype = CeaTsoAttn;

rc = send_message( );

if ( rc == 0 ) {
    printf("\n\nSend TSO Command Successful.\n\n");
    printf("    Send      Message Type:    %2d\n",
           message_queue.message_type);
    printf("    Send      Message Length:  %d\n",
           strlen(message_queue.message_text));
    printf("\n");
}
else {
    printf("\nSend message failed with    ");
    printf("    rc = %d    ", rc);
    printf("    Errno = %X    ", errno);
    printf("    Errno_1r = %X\n\n", __errno2());
    error_counter = error_counter + 1;
    printf("\nVariation %d failed.\n\n\n", variation_id);
    printf("\n");
    return error_counter;
}

rc = verify_messages(message_id, wait_seconds);

if ( rc == 0 )
    printf("\n\nCEATsoRequest( ) Attn starts.\n\n");
else {
    error_counter = error_counter + 1;
    printf("\nVariation %d failed.\n\n\n", variation_id);
    printf("\n\n");
    return error_counter;
}

ceatso_request.ceatso_requesttype = CeaTsoAttn;
set_required_members( );
init_expected_values( );
strcpy(ceatso_request.ceatso_eyecatcher, CEATSOREQUEST_EYECATCHER);

CEATsoRequest(&ceatso_request, &ceatso_query, &ceatso_error);

if ( ceatso_error.returnCode == expected_rc    &&
      ceatso_error.reasonCode == expected_rsn    &&
      ceatso_error.diag.diag1 == expected_diag1    &&
      ceatso_error.diag.diag2 == expected_diag2    &&
      ceatso_error.diag.diag3 == expected_diag3    &&
      ceatso_error.diag.diag4 == expected_diag4    &&
      )
    printf("    Verifying messages after Attn.\n\n");
else {
    error_counter = error_counter + 1;
    printf("CEATsoRequest( ) Attn failed.\n\n");
    print_error_struct( );
    print_request_struct( );
    printf("\nVariation %d failed.\n\n\n", variation_id);
    return error_counter;
}

rc = verify_attn_messages(message_id, wait_seconds);

if ( rc == 0 )
    printf("\n\nCEATsoRequest( ) Attn successful.\n\n");
else {
    error_counter = error_counter + 1;
    printf("CEATsoRequest( ) Attn failed.\n\n");
    print_error_struct( );
    print_request_struct( );
    printf("\nVariation %d failed.\n\n\n", variation_id);
    return error_counter;
}

printf("\n\nCEATsoRequest( ) End    starts.\n\n");
set_required_members( );
init_expected_values( );
ceatso_request.ceatso_requesttype = CeaTsoEnd;

CEATsoRequest(&ceatso_request, &ceatso_query, &ceatso_error);

```

```

if ( ceatso_error.returnValue == expected_rc    &&
    ceatso_error.reasonCode == expected_rsn    &&
    ceatso_error.diag.diag1 == expected_diag1 &&
    ceatso_error.diag.diag2 == expected_diag2 &&
    ceatso_error.diag.diag3 == expected_diag3 &&
    ceatso_error.diag.diag4 == expected_diag4 )
    printf("\n\nCEATsoRequest( ) End session successful.\n");
else {
    error_counter = error_counter + 1;
    printf("\n\nCEATsoRequest( ) End session failed.\n\n");
    print_request_struct( );
    print_error_struct( );
    printf("\nVariation %d failed.\n\n\n", variation_id);
    return error_counter;
}

if ( ceatso_error.returnValue == CEASUCCESS )
    printf("\n\nVariation %d succeeded.\n\n\n\n", variation_id);
else {
    error_counter = error_counter + 1;
    printf("\n\nVariation %d failed.\n\n\n\n", variation_id);
}

printf("=====\n");
printf("==          Finished Start CeaTsoRequest( ) Example  \n");
printf("=====\n");
printf("\n\n\n\n");

return error_counter;
}

```

Sample compile job

For C programmers, you can use the following sample compile job to compile the sample program. For more details about the sample program, see [“Programming example” on page 169](#).

```

/* rexx */
/* c89/cc/c++ */
/* dbx needs -g or -Wc,debug */
/* list\(.\/) */
/* export _C89_STEPS='-1'      enable all steps, inc prelinker */
/* export _C89_TMPS='-3'      prelinker will write composite .p file*/

'c89 -oceasamt -v -g -Wc,LP64,SHOW,S0,AGGR,XREF,N00FF,N00PT,EXP,LIST\(.\/)
SSCOMM,DLL,STA,'LANGLVL(EXTENDED)',WARN64
-Wl,LP64,map,xref
ceasamt.c ceasapit.x

'ls -gatlre ceasamt.* ceasamt'

```

Part 6. zEnterprise Data Compression (zEDC)

Chapter 13. Overview and planning of zEnterprise Data Compression (zEDC)

In today's z/OS environment, many installations want to compress certain types of data to occupy less space while it is not in use, and then restore the data when necessary. Using zEnterprise® Data Compression (zEDC) to compress data might help to reduce CPU cost and elapsed time of data compression compared to traditional software-based compression services, such as CSRCESRV and CSRCMPSC. zEDC can also decrease the cost of applications that use host-based compression that are running on z/OS.

zEDC supports the DEFLATE compression data format, which compresses data by using the following algorithms, which are defined by RFC 1951:

- LZ77
 - Replaces repeated string with length, back pointer pairs.
 - Points back up to 32 K.
- Huffman coding
 - Variable length encoding of characters.
 - Minimize bit length of a stream of characters by assigning shorter codes to more frequent characters.
 - Data and length, back pointer pairs are Huffman encoded.

For more details, check the IETF standard RFC 1951 (tools.ietf.org/html/rfc1951).

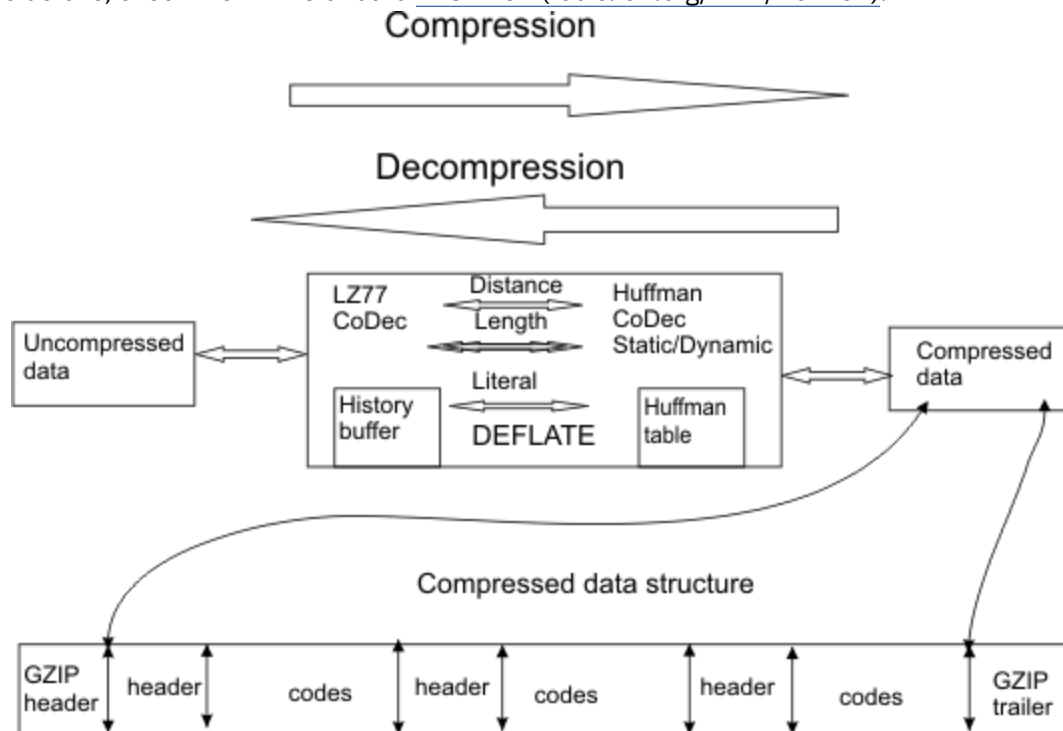


Figure 20. Compression and Decompression Workflow

For help with getting started, and access to various technical resources about Integrated Accelerator for zEDC, see [Integrated Accelerator for zEDC \(www.ibm.com/support/z-content-solutions/compression/\)](http://www.ibm.com/support/z-content-solutions/compression/).

Requirements for zEnterprise Data Compression

Hardware and software requirements.

zEDC requires the following:

- z/OS V2R1 (or later) operating system.
- One of the following:
 - IBM z15®, or later, with the Integrated Accelerator for zEDC
 - IBM zEnterprise EC12 CPC (with GA2 level microcode) or zBC12 CPC, or later, with the zEDC Express feature.
- zEDC software feature enabled in an IFAPRDxx parmlib member. For more information, see [“Product Enablement for zEnterprise Data Compression”](#) on page 188.
- Adequate 64-bit real storage that is configured to this z/OS image.

Product Enablement for zEnterprise Data Compression

Enabling the priced software feature.

The IFAPRDxx PARMLIB member contains definitions for products that require product enablement, this includes zEDC. For more information about IFAPRDxx PARMLIB member, see [IFAPRDxx \(product enablement policy\)](#) in the *z/OS MVS Initialization and Tuning Reference*.

After receiving access to this priced software feature, you are required to change the IFAPRDxx PARMLIB member to include the following statements:

```
PRODUCT OWNER('IBM CORP')
NAME('Z/OS')
ID(5655-ZOS)
FEATURENAME(ZEDC)
VERSION(*)
RELEASE(*)
MOD(*)
STATE(ENABLED)
```

Note: When the member is updated, an IPL is required for the zEDC Express device driver to recognize the enablement.

The **STATE** parameter value should be set to disabled if the zEDC feature is no longer required.

If you try to dynamically enable the feature by using SET PROD=03 IBM MVS System Command, you see the following output:

```
IFA100I IN PARMLIB MEMBER=IFAPRD03 ON LINE 44
PRODUCTS WITH OWNER=IBM CORP NAME=Z/OS
FEATURE=ZEDC VERSION=*.*. ID=5655-ZOS
HAVE BEEN ENABLED.
```

Otherwise, the status of the zEDC product feature remains Disabled. You can verify that the function is disabled with the DISPLAY IQP MVS system command, as shown in the following output:

```
D IQP
IQP066I 14.17.39 DISPLAY IQP
zEDC Information
DEFMINREQSIZE:          N/A
INFMINREQSIZE:          N/A
Feature Enablement:     Disabled
```

After an IPL, you can verify that the zEDC product feature is Enabled by issuing the DISPLAY IQP MVS system command.

```
D IQP
IQP066I 14.58.45 DISPLAY IQP
zEDC Information
DEFMINREQSIZE:          1K (STATIC)
```


INFMINREQSIZE: 1K (STATIC)
 Feature Enablement: Enabled

You can also use the D PROD,REG,FEATURENAME(ZEDC) MVS system command to verify the status of the priced feature. The following output from this command is shown.

```
IFA111I 16.25.49 PROD DISPLAY
S OWNER      NAME      FEATURE  VERSION  ID
E IBM CORP   z/OS      ZEDC     03.01.00 5655-Z0S
```

Planning for zEnterprise Data Compression

Planning for zEnterprise Data Compression use.

zEDC is established by starting either an unauthorized or authorized interface:

- Unauthorized interface for zEDC:
 - zlib for zEDC:
 - zlib is an OpenSource data compression library that supports the DEFLATE compressed data format.
 - The zlib compression library provides in-memory compression and decompression functions, including integrity checks of the decompressed data. For more information, see [zlib Compression Library \(zlib.net\)](#).
- zSystem authorized interfaces for zEDC:
 - Requires supervisor state and supports task and SRB mode.
 - Allows application buffers to be directly read by and written to by compression accelerator hardware, allowing the application to avoid a data move, but also adding complexity to managing I/O buffers.
 - Operates on independent requests:
 - A deflate request produces a full DEFLATE block.
 - An inflate request consumes a full DELFATE block.
 - Provides software inflate capability to maintain data access when z System compression accelerator hardware is not available.
- Additional method with the option to use zEDC:
 - SMF compression. Use the COMPRESS and PERMFIX keywords in the SMFPRMxx parmlib member to compress data before writing to a log stream. For more information, see [z/OS MVS System Management Facilities \(SMF\)](#) and [z/OS MVS Initialization and Tuning Reference](#).

Table 28. Comparison table between unauthorized and z System authorized interfaces for zEDC		
Options	Unauthorized interfaces for zEDC	zSystem authorized interfaces for zEDC
Language	C	Any language that can call OS callable services
Data streaming	zlib-style data streams supported. Data can be broken up across requests as needed, but must be within the minimum input buffer limit.	Each request is independent and handled as a single DEFLATE block. Inflate requests must receive single complete DEFLATE block.
Buffer management	Data move to device driver-managed buffer (IBM z14® and earlier) or data that is compressed directly by using the buffer in the application (IBM z15 and higher).	Application buffer directly used by z System hardware.

<i>Table 28. Comparison table between unauthorized and z System authorized interfaces for zEDC (continued)</i>		
Options	Unauthorized interfaces for zEDC	zSystem authorized interfaces for zEDC
Co-existence support	Both inflate and deflate are completed in software when hardware is not available.	Inflate completed in software when hardware is not available.
Authorization	Controlled by SAF-protected FACILITY class resource FPZ.ACCELERATOR.COMPRESSION (IBM z14 and earlier).	Supervisor state.

Chapter 14. Application interfaces for zEnterprise Data Compression

This topic describes the following interfaces, considerations, and samples for zEnterprise Data Compression (zEDC):

- Invoking unauthorized interface for zEDC:
 - [“zlib for zEnterprise Data Compression” on page 191](#)

Note: This section describes only the POSIX C® language zlib library and functions. For an example of using zlib compression from a non-POSIX C language, see [Using zlib compression from a COBOL program](#).
- Invoking z System authorized interfaces for zEDC:
 - [“z System authorized compression services” on page 197](#)
 - [“FPZ4RZV - Rendezvous compression service” on page 197](#)
 - [“FPZ4PRB — Probe device availability compression service” on page 200](#)
 - [“FPZ4RMR - Memory registration compression service” on page 202](#)
 - [“FPZ4DMR - Deregister memory compression service” on page 204](#)
 - [“FPZ4ABC — Submit compression request” on page 206](#)
 - [“FPZ4URZ - Unrendezvous compression request” on page 211](#)

Invoking unauthorized interfaces for zEnterprise Data Compression

zlib for zEnterprise Data Compression

The zlib data compression library provides in-memory compression and decompression functions, including integrity checks of the uncompressed data. A modified version of the zlib compression library is used by zEDC. The IBM-provided zlib compatible C library provides a set of wrapper functions that use zEDC compression when appropriate and when zEDC is not appropriate, software-based compression services are used.

The zlib wrapper functions use the following criteria to determine if zEDC can be used for compression:

- The system requirements for zEDC have been met. See [“Requirements for zEnterprise Data Compression” on page 188](#) for the details.
- For a deflate stream, the parameters specified on `deflateInit2()` are supported by zEDC. For an inflate stream, all the parameters specified on `inflateInit2()` are supported. See [“Standard zlib functions” on page 192](#) for the details.
- Because there are overhead costs when communicating with the hardware, on the first call to deflate or inflate a data stream, the provided input is checked to ensure that it is sufficiently large enough to make it worthwhile to use zEDC. If the data stream is large enough, zEDC is used. If the data stream is small, it might cost more to compress the data stream with zEDC so software-based compression services are used. This check is only performed on the first call to deflate or inflate a data stream.

If any of these criteria are not met, the zlib wrapper function calls the standard zlib functions to process the data stream in software.

Once zEDC is used as the compression mechanism (for example, after the first call to inflate or deflate the data stream is completed), you cannot change the compression method to software-based compression services. At the same time, if software-based compression services are used as the compression

mechanism (for example, after the first call to inflate or deflate the data stream is completed), you cannot change the compression method to zEDC.

Note: Once a data stream starts using zEDC for compression, if a function is called that cannot be supported by zEDC or the zEDC hardware becomes unavailable, the unsupported function returns an error return code.

Standard zlib functions

The following table contains the standard zlib functions and whether they are supported using zEDC:

Table 29. Standard zlib functions and whether they are supported using zEDC		
zlib function	zEDC-supported	Details
zlibVersion	Supported.	Returns '1.2.11-zEDC'
deflateInit	Supported.	Level is ignored if using zEDC.
deflate	All flush modes are supported.	If the input buffer size is smaller than the minimum threshold for zEDC on the first call to deflate (compress) a data stream, the data stream is compressed using traditional software-based compression.
deflateEnd	Supported.	
inflateInit	Supported.	
inflate	Supported if the flush mode is one of the following: <ul style="list-style-type: none"> • z_no_flush • z_sync_flush • z_finish 	If either the input buffer size is smaller than a minimum threshold for zEDC or the flush mode is z_block or z_trees on the first call to inflate (decompress) a data stream, the data stream is decompressed using traditional software-based decompression. On subsequent calls to inflate a data stream, if the flush mode is z_block or z_trees and the stream is using zEDC decompression, Z_STREAM_ERROR is returned
inflateEnd	Supported.	
deflateInit2	Support is based on the input parameters.	<p>Input parameters:</p> <p>level This option is ignored for zEDC and does not affect the software or zEDC compression decision. This option is supported for zlib software compression.</p> <p>method Must be Z_DEFLATED.</p> <p>windowBits Must be -15 for raw deflate, 15 for zlib header and trailer, or 31 for gzip header and trailer. For all other windowBits values, the data stream uses traditional software-based compression.</p> <p>memLevel This option is ignored for zEDC and does not affect the software or zEDC compression decision. This option is supported for zlib software compression.</p> <p>strategy Use Z_DEFAULT_STRATEGY or Z_FIXED for zEDC. All other options use traditional software-based compression.</p>

Table 29. Standard zlib functions and whether they are supported using zEDC (continued)		
zlib function	zEDC-supported	Details
deflateGetDictionary	Not supported for zEDC.	Returns Z_STREAM_ERROR if the stream is using zEDC.
deflateSetDictionary	Supported.	This option is supported for zEDC when called before the first deflate call for the data stream and is not supported after the first call to deflate.
deflateCopy	Supported.	
deflateReset	Supported.	
deflateResetKeep	Not supported for zEDC.	Returns Z_STREAM_ERROR if the stream is using zEDC.
deflateParams	Support is based on the input parameters.	<p>Input parameters:</p> <p>Level This option is ignored for zEDC.</p> <p>Strategy Use Z_DEFAULT_STRATEGY or Z_FIXED for zEDC. All other options use traditional software-based compression.</p>
deflateTune	Supported.	This option only applies to traditional software-based compression. zEDC accepts the call, but none of the parameters apply to zEDC.
deflateBound	Supported.	
deflatePending	Supported.	
deflatePrime	Not supported for zEDC.	Returns Z_STREAM_ERROR if the stream is using zEDC.
deflateSetHeader	Supported.	
inflateInit2	Supported.	
inflateGetDictionary	Not supported for zEDC.	Returns Z_STREAM_ERROR if the stream is using zEDC.
inflateSetDictionary	Supported if called immediately after a call to inflate the data stream that returns Z_NEED_DICT.	Otherwise, Z_STREAM_ERROR is returned if the data stream is attempting to use zEDC decompression.
InflateSync	Supported.	
inflateSyncPoint	Not supported for zEDC.	Returns Z_STREAM_ERROR if the stream is using zEDC.
inflateCodesUsed	Not supported for zEDC.	Returns Z_STREAM_ERROR if the stream is using zEDC.
inflateCopy	Supported.	
inflateReset	Supported.	
inflatateReset2	Supported.	
inflatePrime	Not supported for zEDC.	Returns Z_STREAM_ERROR if the stream is using zEDC decompression.

Table 29. Standard zlib functions and whether they are supported using zEDC (continued)		
zlib function	zEDC-supported	Details
inflateMark	Not supported for zEDC.	Returns Z_STREAM_ERROR if the stream is using zEDC decompression.
inflateGetHeader	Supported.	
inflateBackInit	Not supported for zEDC.	InflateBackInit forces stream to software-based compression.
inflateBack	Not supported for zEDC.	
inflateValidate	Not supported for zEDC.	Returns Z_STREAM_ERROR if the stream is using zEDC.
zlibCompileFlags	Supported.	
compress	Supported.	
compress2	Supported.	Level is ignored if using zEDC.
compressBound	Supported.	
uncompress	Supported.	
uncompress2	Supported.	
gz* routines	Not supported for zEDC.	Uses software-based compression for inflate and deflate functions.
checksum functions	Not supported for zEDC.	Checksum functions calculate the checksum values using software-based compression services.

IBM-provided zlib compatible C library

The IBM-provided zlib compatible C library provides the following query functions in addition to the standard zlib functions:

deflateHwAvail(*buflen*)

Determines if the compression accelerator is available for a deflate operation. The input parameter *buflen* is an integer that represents the input buffer size of the first deflate request. The function returns an integer with a value of 1 if the compression accelerator will be used for the deflate operation or a value of 0 if software will be used instead.

inflateHwAvail(*buflen*)

Determines if the compression accelerator is available for an inflate operation. The input parameter *buflen* is an integer that represents the input buffer size of the first inflate request. The function returns an integer with a value of 1 if the compression accelerator will be used for this inflate operation or a value of 0 if software will be used instead.

hwCheck(*strm*)

Determines if a zlib stream is using the compression accelerator or software compression. The input parameter *strm* is a pointer to a zlib *z_stream* structure to check. The function returns an integer with a value of 0 if the stream has gone to the compression accelerator, a value of 1 if the stream is pending to go to the compression accelerator, but still could fall back to software compression, a value of 2 if the stream has gone to software compression, or Z_STREAM_ERROR if the stream has not been initialized correctly.

Running zlib

To compress data with zEDC, your installation must meet the system requirements. See [“Requirements for zEnterprise Data Compression”](#) on page 188 for the system requirements for zEDC.

To use the IBM-provided zlib compatible C library for data compression or data expansion services, follow these steps:

1. Link or relink applications to use the IBM-provided zlib.

The IBM-provided zlib is an archive file in the z/OS UNIX System Services file system and can be statically or dynamically linked into your applications. The paths for the zlib archive file and the zlib header files are:

Path for the zlib archive file:

/lib/libz.a

Path for 31-bit non-xplink dynamic library files:

/lib/libz.so

/lib/libz.x

Path for 31-bit xplink dynamic library files:

/lib/libzX.so

/lib/libzX.x

Path for 64-bit dynamic library files:

/lib/libz64.so

/lib/libz64.x

Path for the zlib header files:

/usr/include/

When a new IBM service is provided for zlib, all applications that statically or dynamically link zlib must relink in order to use the updated IBM-provided zlib and take advantage of the new function.

2. Provide access to System Authorization Facility (SAF):

- Access to zEDC Express is protected by the SAF FACILITY resource class for installations that are running IBM zEnterprise z14 and earlier processors. IBM zEnterprise z15 and later processors no longer have this requirement.
- Give READ access to FPZ.ACCELERATOR.COMPRESSION to the identity of the address space that the zlib task will run in.

The access check is performed during the first call in a given task. The results of that first check are cached for the duration of the task.

3. Use the z/OS UNIX environmental variable `_HZC_COMPRESSION_METHOD` to control if zEDC is used for data compression. If the value of *software* is set, software-based compression services are used. All other values result in the default behavior of attempting to use zEDC for data compression.
4. Ensure that adequately sized input buffers are available. If the input buffer size falls below the minimum threshold, data compression occurs using zlib software compression and not zEDC.

Note: IBM zEnterprise z15 and above processor thresholds will no longer be tunable through parmlib. The IQPPRMxx will still be allowed in the configuration, but the values are not accepted.

The environment variables `_HZC_DEFLATE_THRESHOLD` and `_HZC_INFLATE_THRESHOLD` can also be used to control the threshold for going to zEDC. The valid values are in the range 1-9999999.

For example:

`_HZC_DEFLATE_THRESHOLD=1` would force all deflate requests with an initial input size of 1 byte or larger to use zEDC.

This threshold can be controlled at a system level by using the PARMLIB member IQPPRMxx.

5. Use the z/OS UNIX environmental variable, `_HZC_CHECKSUM_METHOD`, to control if SIMD acceleration is used in checksum verification. If the value of *software* is set, software-based checksum verification is used. All other values result in the default behavior, which means if hardware supported SIMD, then SIMD acceleration is used.

6. Allocate the correct amount of storage for I/O buffers. The zEDC requests generated by zlib use predefined I/O buffer pools. The size of these I/O buffer pools can be set using PARMLIB member IQPPRMxx.

For IBM z15 and later processors, these buffers are no longer applicable. The IQPPRMxx will still be allowed in the configuration, but the values will no longer be accepted.

7. Environmental variable operations are not thread-safe. For multi-thread programming, if some threads use zEDC while other threads do not, a thread-specific data key (hzc_compression_method) is provided to handle this scenario.
 - a. To use the new feature, the user code must include the following statement: `extern pthread_key_t hzc_compression_method;`
 - b. The key must be created in the user code and data must be specified to the key. The destructor routine is also defined by the user.
 - c. If any data is specified to the thread-specific data key hzc_compression_method, its value will override the environmental variable _HZN_COMPRESSION_METHOD. Otherwise, _HZN_COMPRESSION_METHOD will take effect.
 - d. If the specified data value is software, software-based data compression is used. All other values result in the default behavior of attempting to use zEDC for data compression.

For more information about thread-specific data keys, see [pthread_key_create\(\) – Create thread-specific data key in z/OS C/C++ Runtime Library Reference](#).

When zlib is statically linked into an application that runs on software or hardware that is not compatible with zEDC, zlib uses the following compression and decompression:

Table 30. Compression and decompression with zlib			
Hardware level	z/OS level	zEDC Express	Description
zEC12 (with GA2 level microcode)	z/OS V2R1	Active	zEDC is used for both data compression and decompression.
zEC12 (with GA2 level microcode)	z/OS V2R1	Not Active	Requirements are not met for zEDC. When zEDC Express is not available, traditional software zlib is used for compression and decompression.
Pre-zEC12 (with GA2 level microcode)	z/OS V2R1 or pre-z/OS V2R1	N/A	Requirements are not met for zEDC. When zEDC Express is not available, traditional software zlib is used for compression and decompression.

zEDC error handling:

- If a z System compression accelerator is unavailable, data compression requests transfer to another z System compression accelerator configured to the same partition. These request transfers are transparent to the application.
- If all z System compression accelerators are unavailable, an error message is sent to the application.

Invoking z System authorized interfaces for zEnterprise Data Compression

This topic describes how to invoke z System authorized interfaces for zEnterprise Data Compression by:

- [“z System authorized compression services” on page 197](#)
 - [“FPZ4RZV - Rendezvous compression service” on page 197](#)
 - [“FPZ4PRB – Probe device availability compression service” on page 200](#)
 - [“FPZ4RMR - Memory registration compression service” on page 202](#)

- [“FPZ4DMR - Deregister memory compression service” on page 204](#)
- [“FPZ4ABC – Submit compression request” on page 206](#)
- [“FPZ4URZ - Unrendezvous compression request” on page 211](#)

To compress data with zEDC, your installation must meet the system requirements. See [“Requirements for zEnterprise Data Compression” on page 188](#) for the system requirements for zEDC.

All z/OS exploitation of zEDC handles mixed hardware and software levels. Compatibility APAR OA41245 provides software decompression for installations running with z/OS V1R13 or V1R12. The same software decompression is also provided for installations running z/OS V2R1 on pre-IBM zEnterprise EC12 (with GA2 level microcode). This allows access to compressed data on all combinations of environments.

<i>Table 31. Compression and decompression with z System authorized interfaces for zEDC</i>			
Hardware level	z/OS level	zEDC Express	Description
zEC12 (with GA2 level microcode)	z/OS V2R1	Active	zEDC is used for both data compression and decompression.
zEC12 (with GA2 level microcode)	z/OS V2R1	Not Active	Requirements are not met for zEDC. Software-based decompression services for zEDC Express compressed data are used because zEDC Express compression is not available.
Pre-zEC12 (with GA2 level microcode)	z/OS V2R1	N/A	Requirements are not met for zEDC. Software-based decompression services for zEDC Express compressed data are used because zEDC Express compression is not available.
Pre-zEC12 (with GA2 level microcode)	Pre-z/OS V2R1	N/A	Requirements are not met for zEDC. Software-based decompression services for zEDC Express compressed data are used because zEDC Express compression is not available. APAR OA41245 is required to use the software-based decompression services.

z System authorized compression services

The following compression services are available when using z System authorized interfaces for zEDC:

- [“FPZ4RZV - Rendezvous compression service” on page 197](#)
- [“FPZ4PRB – Probe device availability compression service” on page 200](#)
- [“FPZ4RMR - Memory registration compression service” on page 202](#)
- [“FPZ4DMR - Deregister memory compression service” on page 204](#)
- [“FPZ4ABC – Submit compression request” on page 206](#)
- [“FPZ4URZ - Unrendezvous compression request” on page 211](#)

FPZ4RZV - Rendezvous compression service

Description

The FPZ4RZV service performs the required setup and initialization of the compression services for an exploiter. The scope is the address space of the application and it is valid for the life of the Cross Memory Resource Owner (CMRO) task.

Notes:

1. A maximum of 255 rendezvous tokens are supported per each address space. This allows multiple applications to exploit the compression driver so each can maintain their own rendezvous scope.
2. All 64-bit storage is obtained with the MEMLIMIT=NO option.

Table 32. Environment for the FPZ4RZV service

Environmental factor	Requirement
Minimum authorization:	Supervisor State with Key 0
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	64-bit
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Table 33. Parameters for the FPZ4RZV service

Name	Type	Input/ Output	Description
<i>ApplicationId</i>	Fixed(3 2)	Input	The application type to use. 0x01 is the application type for zEDC.
FPZ4RZV_options	Bit(64)	Input	Options for the FPZ4RZV service: SoftwareInflate (X'80000000 00000000') Allows compression requests to fall back to software inflation when no compression devices are available. EnableABCScatter (X'40000000 00000000') Allows compression requests to use the FPZ4ABC compression service to submit work with scatter/gather lists. FailOnNoDevices (X'20000000 00000000') If specified, compression requests fail when no compression devices are available. If FailOnNoDevices is not specified, a valid rendezvous token is returned even if no compression devices are currently available. This returned rendezvous token is used for all other services. PlusOne (X'08000000 00000000') If specified, compression requests will only use zEDC Express Adapters with the February 26, 2014 Firmware MCL release, or later. RmrEntriesExact (X'04000000 00000000') If specified, the <i>rmr_entries</i> parameter represents the maximum number of outstanding memory registrations for this rendezvous. If this limit is exceeded, the FPZ4RMR service may fail the request.
<i>userid</i>	Char(8)	Input	An eight character EBCDIC string identifying the user.

Table 33. Parameters for the FPZ4RZV service (continued)

Name	Type	Input/ Output	Description
<i>rmr_entries</i>	Fixed(32)	Input	The estimated number of FPZ4RMR compression service calls to be performed that helps to size the tables used until the maximum number of registrations is reached. This is an optional parameter. The value of the <i>rmr_entries</i> parameter can be anywhere between 1 and 64K. The default is 128. Define <i>rmr_entries</i> as integer data of length 32.
Rendezvous token	Char(16)	Output	This is the token that must be passed to all FPZ services.
Return code	Fixed(31)	Output	The return code for the service.
Reason code	Fixed(32)	Output	The reason code for the service.

Table 34. Return and reason codes for the FPZ4RZV service

Hexadecimal return code	Reason code	Meaning and action
00	0000	Meaning: The call completed successfully. Action: None.
04	0000	Meaning: No zEDC devices are available. zEDC support is active so it is possible that zEDC devices might become available in the future. Action: If zEDC devices are available to this system, perform diagnostics to determine the reason for the failure.
04	0102	Meaning: No zEDC devices are available because the system requirements for zEDC were not met. See “Requirements for zEnterprise Data Compression” on page 188 for the details. A 'thin' rendezvous was created. Action: None.
08	0000	Meaning: No zEDC devices are available because the system requirements for zEDC were not met. This is the result of RvzFailOnNoDev being ON or SoftwareInflate being OFF when on downlevel hardware or software. See “Requirements for zEnterprise Data Compression” on page 188 for the details. No rendezvous token is returned. Action: None.
0C	0201	Meaning: Invalid parameter combination. Action: Check the calling program for a probable coding error. Correct the program and rerun it.

Table 34. Return and reason codes for the FPZ4RZV service (continued)		
Hexadecimal return code	Reason code	Meaning and action
0C	0207	Meaning: The calling environment is invalid. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0210	Meaning: <i>rnr_entries</i> specified an invalid value. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0226	Meaning: Invalid application specified. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
10	0301	Meaning: An internal error caused recovery to be entered. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.
10	0303	Meaning: The maximum number of rendezvous tokens have been reached for the address space. Action: Determine if the calling program is at fault because of a coding error. If there is no coding error, another program might be consuming all the rendezvous tokens for the address space. Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.
10	030B	Meaning: The CMRO task is ending. Action: None, since the address space is ending. This reason code should be accounted for in your code scenarios.
10	030C	Meaning: Too many latch sets requested. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

FPZ4PRB – Probe device availability compression service

Description

The FPZ4PRB service checks for the required hardware and software needed for zEDC. This service returns successful if they are available to the system. See [“Requirements for zEnterprise Data Compression”](#) on page 188 for the details.

Table 35. Environment for the FPZ4PRB service	
Environmental factor	Requirement
Minimum authorization:	Supervisor State with Key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	64-bit
Interrupt status:	Enabled for I/O and external interrupts

Table 35. Environment for the FPZ4PRB service (continued)	
Environmental factor	Requirement
Locks:	No locks held

Table 36. Parameters for the FPZ4PRB service			
Name	Type	Input/Output	Description
<i>ApplicationId</i>	Fixed(32)	Input	The application type to use. 0x01 is the application type for zEDC.
FPZ4PRB_options	Bit(64)	Input	Options for the FPZ4PRB service: PlusOne (X'80000000 00000000') If specified, only zEDC Express Adapters with the March 31, 2014 Firmware MCL release, or later, will be honored. The value returned in <i>NumDevices</i> will only indicate this subset of devices. Note: This option is not applicable when running on z15 and above. PRBHasSync(X'40000000) If specified, the service will return with a return code of zero in the event that zEDC is running on a z15.
<i>NumDevices</i>	Fixed(32)	Output	The number of devices available for this application.
Return code	Fixed(31)	Output	The return code for the service.
Reason code	Fixed(32)	Output	The reason code for the service.

Table 37. Return and Reason Codes for the FPZ4PRB service		
Hexadecimal Return Code	Reason Code	Meaning and Action
00	0000	Meaning: Devices are available. Action: None.
00	00	Meaning: Compression is available. Action: None.
08	0900	Meaning: The z/OS software level is not correct for zEDC. See “Requirements for zEnterprise Data Compression” on page 188 for the details. Action: None.
08	0901	Meaning: The hardware level is not correct for zEDC. See “Requirements for zEnterprise Data Compression” on page 188 for the details. Action: None.

Table 37. Return and Reason Codes for the FPZ4PRB service (continued)

Hexadecimal Return Code	Reason Code	Meaning and Action
08	0902	Meaning: No zEDC devices are available. The hardware is at the correct level, but no zEDC devices were available. Action: If zEDC devices are available to this system, perform diagnostics to determine the reason for the failure.
08	0903	Meaning: zEDC devices were available during this IPL at some point, but there are no zEDC devices available now. Action: Perform diagnostics to determine the reason for the failure.

FPZ4RMR - Memory registration compression service

Description

The FPZ4RMR service registers a segment of memory for use by zEDC. The result is that this storage becomes fixed. The data area passed to FPZ4RMR must be page-aligned, and the size must be a multiple of a page boundary.

Note: This is not compatible with existing page fix services. This storage is eligible to be used for I/O as a result of this service.

Table 38. Environment for the FPZ4RMR service

Environmental factor	Requirement
Minimum authorization:	Supervisor State with Key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	64-bit
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Table 39. Parameters for the FPZ4RMR service

Name	Type	Input/Output	Description
<i>ApplicationId</i>	Fixed(32)	Input	The application type to use. 0x01 is the application type for zEDC.
FPZ4RMR_options	Bit(64)	Input	There are no supported options for the FPZ4RMR service.
Rendezvous token	Char(16)	Input	The rendezvous token.
Data@	Ptr(64)	Input	The address of the data area to register. Note: Large page frames must be in fixed storage.
DataLen	Fixed(64)	Input	The length of the data area to register.

Table 39. Parameters for the FPZ4RMR service (continued)			
Name	Type	Input/Output	Description
Reserved	Fixed(32)	Input	Reserved. Must be 0.
DataKey	Fixed(8)	Input	The key of the data area to register. The format of this parameter is 0xk0, where <i>k</i> represents the key of the data area.
RMR Token	Char(8)	Output	The region memory registration token associated with this data area. This token needs to be passed to the FPZ4ABC service when this data area is used as input or output.
Return code	Fixed(31)	Output	The return code for the service.
Reason code	Fixed(32)	Output	The reason code for the service.

Table 40. Return and Reason Codes for the FPZ4RMR service		
Hexadecimal Return Code	Reason Code	Meaning and Action
00	0000	Meaning: The call completed successfully. Action: None.
08	0000	Meaning: Memory can not be registered because of lack of hardware support. Action: None.
08	0900	Meaning: Incorrect software level for zEnterprise data compression accelerator support. Action: None.
0C	0207	Meaning: The calling environment is invalid. Action: Determine if the calling program is at fault because of a coding error.
0C	0208	Meaning: An invalid rendezvous token was passed. Action: Check that the application successfully called the FPZ4RZV service.
0C	021D	Meaning: The supplied region was not CONTROL(AUTH). Action: Determine if the calling program is at fault because of a coding error.
0C	021E	Meaning: The supplied region address is incorrect. It might not have been page-aligned. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Table 40. Return and Reason Codes for the FPZ4RMR service (continued)		
Hexadecimal Return Code	Reason Code	Meaning and Action
0C	021F	Meaning: The region length is invalid. It is possible that it is not a multiple of page size. Action: Determine if the calling program is at fault because of a coding error.
0C	0220	Meaning: There is a region key mismatch. Action: Determine if the calling program is at fault because of a coding error.
0C	0226	Meaning: An invalid application ID was encountered. Action: Determine if the calling program is at fault because of a coding error.
0C	0227	Meaning: Rendezvous was not created with data space support. Action: Determine if the calling program is at fault because of a coding error.
10	0301	Meaning: An internal error has occurred. Action: Determine if the calling program is at fault because of a coding error.
10	0304	Meaning: Compression services were not initialized. Rendezvous was not called. Action: Check that the application successfully called the FPZ4RZV service.
10	0305	Meaning: Capacity has been reached for memory registrations. Action: Determine if the calling program is at fault because of a coding error.
10	0306	Meaning: There is not enough DMA memory available. Action: Determine if the calling program is at fault because of a coding error.
10	030D	Meaning: Missing latch set token. Action: Determine if the calling program is at fault because of a coding error.

FPZ4DMR - Deregister memory compression service

Description

The FPZ4DMR service unregisters a segment of memory for use by zEDC Express. The result is that this storage becomes unfixed.

Table 41. Environment for the FPZ4DMR service	
Environmental factor	Requirement
Minimum authorization:	Supervisor State

<i>Table 41. Environment for the FPZ4DMR service (continued)</i>	
Environmental factor	Requirement
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	64-bit
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

<i>Table 42. Parameters for the FPZ4DMR service</i>			
Name	Type	Input/Output	Description
<i>ApplicationId</i>	Fixed(32)	Input	The application type to use. 0x01 is the application type for zEDC.
FPZ4DMR_options	Bit(64)	Input	There are no supported options for the FPZ4DMR service.
Rendezvous token	Char(16)	Input	The rendezvous token.
RMR token	Char(8)	Input	The region memory registration (RMR) token associated with this data area to be unregistered.
Return code	Fixed(31)	Output	The return code for the service.
Reason code	Fixed(32)	Output	The reason code for the service.

<i>Table 43. Return and Reason Codes for the FPZ4DMR service</i>		
Hexadecimal Return Code	Reason Code	Meaning and Action
00	0000	Meaning: The call completed successfully. Action: None.
08	0900	Meaning: Incorrect software level for zEnterprise data compression accelerator support. Action: None.
0C	0207	Meaning: The calling environment is invalid. Action: Determine if the calling program is at fault because of a coding error.
0C	0208	Meaning: An invalid rendezvous token was passed. Action: Check that the application successfully called the FPZ4RZV service.
0C	0209	Meaning: An invalid RMR token was provided. Action: Determine if the calling program is at fault because of a coding error.

Table 43. Return and Reason Codes for the FPZ4DMR service (continued)		
Hexadecimal Return Code	Reason Code	Meaning and Action
10	0301	Meaning: An internal error has caused recovery to be entered. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.
10	0304	Meaning: Compression services were not initialized. Rendezvous was not called. Action: Check that the application successfully called the FPZ4RZV service.
10	030D	Meaning: Missing latch set token. Action: Determine if the calling program is at fault because of a coding error.

FPZ4ABC – Submit compression request

Description

The FPZ4ABC service submits a single autonomous compression request for one or more DEFLATE blocks. The input and output buffers can be either direct buffers or scatter/gather lists. The maximum size of a request for FPZ4ABC is 1 MB.

Table 44. Environment for the FPZ4ABC service	
Environmental factor	Requirement
Minimum authorization:	Supervisor State with Key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	64-bit
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Table 45. Parameters for the FPZ4ABC service			
Name	Type	Input/Output	Description
<i>ApplicationId</i>	Fixed(3 2)	Input	The application type to use. 0x01 is the application type for zEDC.

Table 45. Parameters for the FPZ4ABC service (continued)			
Name	Type	Input/ Output	Description
FPZ4ABC_options	Bit(64)	Input	Options for the FPZ4ABC service: Inflate (X'80000000 00000000') When ON, specifies that this is an inflation request. Input Scatter List (X'40000000 00000000') When ON, the area pointed to by input@ is a scatter/gather list. Output Scatter List (X'20000000 00000000') When ON, the area pointed to by output@ is a scatter/gather list. AbcSyncRequest(X'10000000 ..) When ON, this is the synchronous request.
Rendezvous token	Char(16)	Input	The rendezvous token.
Input@	Ptr(64)	Input	The address of the input area or input scatter/gather list.
Output@	Ptr(64)	Input	The address of the output area or output scatter/gather list.
Input@RMR Token	Char(8)	Input	The region memory registration (RMR) token for the input area or area pointed to by the input scatter/gather list.
Output@RMR Token	Char(8)	Input	The region memory registration (RMR) token for the output area or area pointed to by the output scatter/gather list.
InputLen	Fixed(64)	Input	The length of the area pointed to by Input@. In the event that a scatter/gather list was provided using Input@, the total length of the areas provided by the scatter/gather areas must be provided.
OutputLen	Fixed(64)	Input	The length of the area pointed to by Output@. In the event that a scatter/gather list was provided using Output@, the total length of the areas provided by the scatter/gather areas must be provided.
GeneratedOutputLen	Fixed(64)	Output	This length describes how much output was generated and stored in either the Output@ or the scatter/gather list specified by Output@. This length spans across scatter/gather entries.
Return code	Fixed(31)	Output	The return code for the service.
Reason code	Fixed(32)	Output	The reason code for the service.

The FPZ4ABC service allows for the input and output areas to span several non-contiguous areas. The header of the FPZ4ABC list is immediately followed by the list entries. All entries in the scatter/gather list must be associated with the same RMR token.

Scatter/gather lists have alignment rules and every entry in the scatter/gather list is checked for the following conditions:

- The start of the first buffer in the list can be on any byte boundary.
- The end of the first buffer must be on the required byte boundary.
- The start / end of the intermediate buffers must be on the required byte boundary.
- The start of the last buffer must be on the required byte boundary.
- The end of the last buffer can be on any boundary.

All required boundaries are on 128-byte alignment. A maximum of 8 scatter/gather entries are allowed.

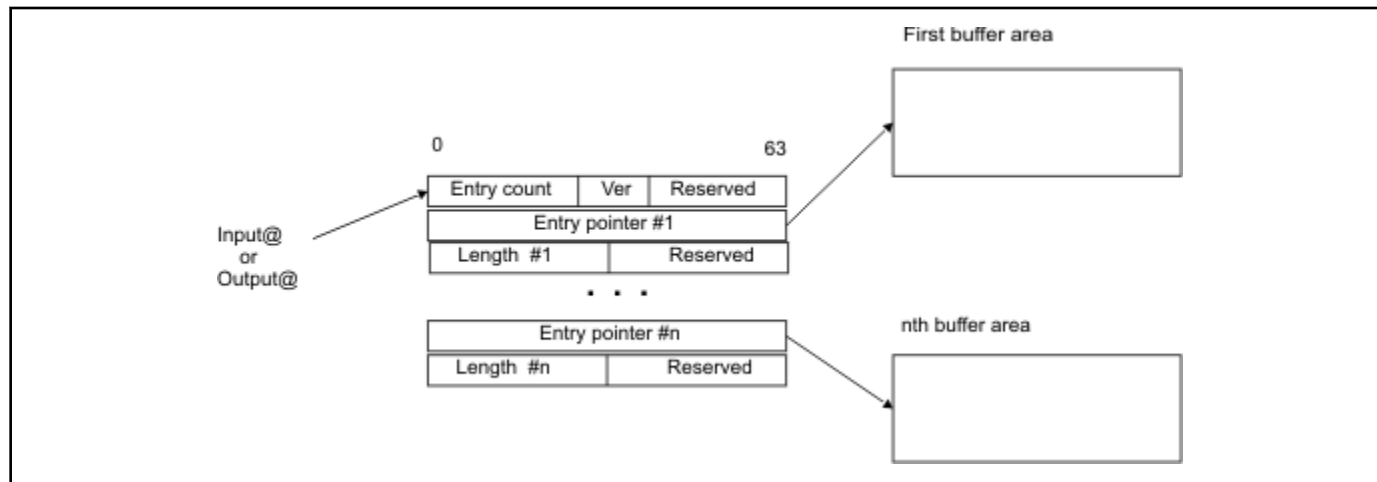


Table 46. Header elements in the FPZ4ABC-generated list

Name	Type	Description
# Of Entries	Fixed(32)	The number of entries in the list.
Version	Fixed(8)	The version associated with the list.
Reserved	Char(3)	Reserved space.

Table 47. Entries elements in the FPZ4ABC-generated list

Name	Type	Description
Address	Fixed(64)	The address into the area mapped by the region memory registration (RMR) token.
Length	Fixed(32)	The length of the area, starting at address, to use.
Reserved	Fixed(32)	Reserved space.

Table 48. Return and Reason Codes for the FPZ4ABC service

Hexadecimal Return Code	Reason Code	Meaning and Action
00	0000	Meaning: The call completed successfully. Action: None.
04	2000	Meaning: No zEDC devices are available. Inflate is completed in software when hardware is not available. Action: None.

Table 48. Return and Reason Codes for the FPZ4ABC service (continued)		
Hexadecimal Return Code	Reason Code	Meaning and Action
08	0000	<p>Meaning: No zEDC devices are available.</p> <p>Action: If zEDC devices are available to this system, perform diagnostics to determine the reason for the failure.</p>
0C	0202	<p>Meaning: One of the buffers had a length of 0, or the first word of a length was non-zero, or one of the buffers has a length greater than 1 MB.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	0203	<p>Meaning: A failure occurred while accessing one of the provided scatter/gather buffers.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	0206	<p>Meaning: The output area was not large enough to complete the request.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	0207	<p>Meaning: The calling environment is invalid. The caller is either Problem State, non-zero key, or in XMEM mode.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	0208	<p>Meaning: The rendezvous token is invalid.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	0209	<p>Meaning: The region memory registration (RMR) token is invalid.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	0221	<p>Meaning: The header of the FPZ4ABC-generated list was not formed correctly.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	0222	<p>Meaning: Either zero or a number greater than the maximum supported was specified for the number of entries in the FPZ4ABC-generated list.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	0223	<p>Meaning: A buffer in the scatter/gather list was not aligned properly.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>

Table 48. Return and Reason Codes for the FPZ4ABC service (continued)		
Hexadecimal Return Code	Reason Code	Meaning and Action
0C	0224	<p>Meaning: The total length of the buffers in the scatter/gather list does not match the length in the parmlist.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	0225	<p>Meaning: Scatter/gather was requested, but it was not enabled for this rendezvous token.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	1202	<p>Meaning: An address range is not contained in the region denoted by the region memory registration (RMR) token.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	1203	<p>Meaning: An unsupported operation was requested.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	1205	<p>Meaning: An inflate request failed because of malformed data.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	2101	<p>Meaning: An inflate request failed in software mode due to malformed input data.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	2102	<p>Meaning: Not enough space in the output buffer to process the request in software mode.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
10	0301	<p>Meaning: An internal component error occurred.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
10	0304	<p>Meaning: A rendezvous has not yet occurred for this address space.</p> <p>Action: Check that the application successfully called the FPZ4RZV service.</p>
10	1203	<p>Meaning: There are no zEDC devices available and either the request was a deflate request or software inflate was not enabled.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>

Table 48. Return and Reason Codes for the FPZ4ABC service (continued)		
Hexadecimal Return Code	Reason Code	Meaning and Action
10	1301	Meaning: The request failed unexpectedly for an unknown reason. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.
10	030D	Meaning: Missing latch set token. Action: Determine if the calling program is at fault because of a coding error.

FPZ4URZ - Unrendezvous compression request

Description

The FPZ4URZ service removes the address space level information related to zEDC Express compression services. Any outstanding memory registrations are unregistered.

Table 49. Environment for the FPZ4URZ service	
Environmental factor	Requirement
Minimum authorization:	Supervisor State
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	64-bit
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Table 50. Parameters for the FPZ4URZ service			
Name	Type	Input/Output	Description
<i>ApplicationId</i>	Fixed(3 2)	Input	The application type to use. 0x01 is the application type for zEDC.
FPZ4URZ_options	Bit(64)	Input	There are no supported options for the FPZ4URZ service.
Rendezvous token	Char(1 6)	Input	The rendezvous token.
Return code	Fixed(3 1)	Output	The return code for the service.
Reason code	Fixed(3 2)	Output	The reason code for the service.

Table 51. Return and Reason Codes for the FPZ4URZ service

Hexadecimal Return Code	Reason Code	Meaning and Action
00	0000	Meaning: The call completed successfully. Action: None.
0C	0207	Meaning: The calling environment is invalid. The caller is either Problem State, non-zero key, or in XMEM mode. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0208	Meaning: An invalid rendezvous token was passed. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
10	0301	Meaning: An internal error has caused recovery to be entered. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.
10	0304	Meaning: Compression services were not initialized. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
10	030D	Meaning: Missing latch set token. Action: Determine if the calling program is at fault because of a coding error.

Usage example of a z System authorized service

The following example uses the authorized services to perform compression using zEDC Express. If zEDC Express adapters are not available, data is written to the destination uncompressed.

The FPZ4PRB service is called intermittently after the FPZ4ABC service returns to the application with a return code that indicates that all zEDC devices have left the configuration.

```

Call FPZ4RZV(AppId, RzvOptions, RzvUserId, RzvToken, RetCode, RsnCode)          /* Rendezvous
with the compression                                                         device
driver (once per address)                                                    space) */

If RetCode = RcNoDevices Then                                                /* If no
devices available */                                                         /* Indicate no
  NoDevices = ON                                                             devices */

Call FPZ4RMR(AppId, RmrOptions, RzvToken, InBuffer@, InBufferLen, 0, InBufKey, InRmrToken,          /* Register the
RetCode, RsnCode)                                                           input buffer */
Call FPZ4RMR(AppId, RmrOptions, RzvToken, OutBuffer@, OutBufferLen, 0, OutBufKey, OutRmrToken,          /* Register the
RetCode, RsnCode)                                                           output buffer for
data */                                                                      compressed

Do Until End of Data
  Read next block of data into InBuffer@

  If NoDevices = ON Then                                                      /* If no
devices available */                                                         /* Probe for
  Call FPZ4PRB(AppId, Options, NumDevices, RetCode, RsnCode)                new devices */

  If RetCode = RcOk Then                                                      /* If devices
now available */

```



```

                                NoDevices = OFF                                /* Indicate we
have devices */                                /* Else no
                                Write InBuffer                                /* Processed
devices */                                /* If devices
                                If NoDevices = OFF Then                                /* If devices
available */                                /* If data was
                                Call FPZ4ABC(RzvToken,                                /* Perform
                                InBuffer@, InBufferLen, InRmrToken,
                                OutBuffer@, OutBufferLen, OutRmrToken,
                                RetCode, RsnCode)                                /* Perform
compression */                                /* If data was
                                If RetCode = RcOk Then                                /* If data was
compressed */                                /* Process
                                Write OutBuffer                                /* Process
compressed data */                                /* If no
                                Else If RetCode = RcNoDevices Then                                /* If no
devices available */                                /* Indicate no
                                NoDevices = ON                                /* Indicate no
devices */                                /* Process
                                Write InBuffer                                /* Process
uncompressed data */                                /* Process
End Loop

Call FPZ4DMR(AppId, DmrOptions, RzvToken, InRmrToken, RetCode, RsnCode)
Call FPZ4DMR(AppId, DmrOptions, RzvToken, OutRmrToken, RetCode, RsnCode)

```

Chapter 15. Troubleshooting for zEDC

This topic explains troubleshooting techniques for zEDC.

RMF provides the following data for the z System accelerator device:

- Load current partition is putting on device
- Compression and decompression request rate and throughput
- Achieved compression ratio

See *z/OS Resource Measurement Facility User's Guide* for the available options to specify on your Monitor I session for reporting on the z System compression accelerator.

Part 7. Other callable services

Chapter 16. 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.

CALL statement	Parameters
CALL IEAAFFN	(feature ,operation_type ,return_code)

The parameters are explained as follows:

feature

Specifies the feature required by your program. Specify CRYPTO to indicate an ICRF.

Define *feature* as character data of length 10. Pad the string on the right with 4 blanks.

,operation_type

Specifies the type of action you want to take. The types are:

GRANT

Establish affinity for the program to processors with an ICRF.

REMOVE

Remove affinity for the program to processors with an ICRF.

Note: After you issue a REMOVE request, the program has no processor affinity; it can run on any processor.

Define *operation_type* as character data of length 6. If you specify GRANT, pad the string on the right with 1 blank.

,return_code

When IEAAFFN completes, *return_code* contains the return code from the service. The return code value is also in register 15.

Define *return_code* as integer data of length 4. The return codes are explained under [“Return codes”](#) on page 220.

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

Requirement	Details
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 52. IEAAFFN Return Codes	
Return code	Meaning and Action
00000000 (0)	Meaning: The operation was successful. Action: None required.
00000004 (4)	Meaning: The program already had processor affinity assigned to it by the system programmer. The system replaces that affinity with the affinity you requested in this service. Action: None required.
0000000C (12)	Meaning: Your program was not running in task mode. Action: This service is not available to SRB mode programs. See the FEATURE= option on the SCHEDULE macro for the use of this function in SRB mode.
00000010 (16)	Meaning: The feature you specified was not a valid feature. Action: Specify a valid feature name.
00000014 (20)	Meaning: The operation type you specified was not valid. Action: Specify a valid operation type.
00000018 (24)	Meaning: The feature you specified is not installed on any of the processors in the system. Action: To the system programmer: See that the program runs on a system with the feature installed.
0000001C (28)	Meaning: A system error has occurred. Action: To the system programmer: The error is recorded in LOGREC. Look for a record with a subcomponent of "IEAAFFN CSS"; then call your IBM Support Center.

Chapter 17. CSRL16J/CSRLJ1 – Transfer control with all registers intact

The CSRL16J/CSRLJ1 service allows you to transfer control with all registers intact running under the same request block (RB) as the calling program. The CSRL16J/CSRLJ1 service will transfer control with the contents of all 16 registers intact. When you transfer control to the other routine, use the CSRL16J/CSRLJ1 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/CSRLJ1 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/L16J1, which forms the parameter list passed from the calling program to CSRL16J/CSRLJ1. Use the CSRYL16J mapping macro to see the format of the L16J/L16J1 parameter lists. To build the L16J/L16J1 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/L16J1 (defined in mapping macro CSRYL16J):

- The version of the parameter list, 0 when building L16J in field L16JVERSION, 1 when building L16J1 in field L16J1VERSION.
- Length of the L16J/L16J1 parameter list, L16JLENGTH/L16J1LENGTH field.
- For L16J, contents of the general purpose registers (GPRs) 0-15, L16JGRS field.
- For L16J1, contents of the 64-bit general registers 0-15, L16J1G64RS field.
- Contents of the access registers (ARs) 0-15, L16JARS/L16J1ARS field.
- PSW information for the target routine, for L16J the 8-byte ESA/390 PSW field L16JPSW, for L16J1 the 16-byte z/Architecture PSW field L16J1PSWE.
 - PSW address and AMODE (AMODE 64 can be identified only if using L16J1)
 - 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/L16J1PROCESSARS bit.

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/CSRLJ1 passes control to the target routine, the access registers (ARs) contain:

**Register
Contents**

0-15

Specified by the caller

If the bit is set off when CSRL16J/CSRLJ1 passes control to the target routine, the access registers (ARs) contain:

**Register
Contents**

0-1

Do not contain any information for use by the routine

2-13

The contents are the same as they were when the caller issued the CSRL16J service.

14-15

Do not contain any information for use by the routine

Freeing dynamic storage associated with the caller

If the calling program has a dynamic storage area associated with it, you can specify that some or all of this storage area be freed before CSRL16J transfers control to the target. In the L16J/L16J1 parameter list defined in mapping macro CSRYL16J, specify:

- The subpool of the area that you want the system to free. L16JSUBPOOL/L16J1SUBPOOL field.
- The length, in bytes, of the dynamic storage area you want the system to free. L16JLENGHTHOFREE/L16J1LENGHTHOFREE field.
- The address of the dynamic storage area you want the system to free. L16JAREATOFREE/L16J1AREATOFREE field.

Make sure that the address is on a double-word boundary. Otherwise the service ends with an abend code X'978'. See *z/OS MVS System Codes* for information on abend code X'978'.

The system frees the storage only when the CSRL16J/CSRLJ1 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. Before you use the CSRLJ1 service, you must build a parameter list, L16J1, 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/L16J1 parameter area and return codes.

If you are coding in PL/I, you can include the CSRLJPLI macro to provide declarations for the return codes only. See Figure 21 on page 225 for the CSRLJPLI macro. Use the data area, mapped by the CSRYL16J mapping macro, as a model for the structure of your parameter list when coding in PL/I.

CSRLJC provides the following declarations for use in your C/370 program:

```

/*****
 *      Type Definitions for User Specified Parameters      *
 *****/

/* Type for user supplied L16J                                */
typedef struct ??<                                           */
    int Version;      /* Must be 0 */

```

```

int Length;          /* Initialize to CSRL16J_LENGTH          */
int SubPool;         /* Subpool of storage to be freed          */
union ??<
    char GRs??(64??); /* General registers          */
    int  GR??(16??);  /* General register 0-15      */
??> u1;
union ??<
    char ARs??(64??); /* Access registers          */
    int  AR??(16??);  /* Access register 0-15      */
??> u2;
union ??<
    char PSW??(8??);  /* PSW: the processing will use the address,
                        AMODE, ASC mode, CC, and program mask. For a
                        supervisor state or PKM 0-7 or key 0-7
                        caller, it will use the state and key from
                        the PSW. Otherwise, it will set to caller
                        key and state.          */
struct ??<
    int PSWByte0to3 : 32; /* First 4 bytes          */
    union ??<
        void *PSWAddr; /* Address and AMODE      */
        struct ??<
            int PSWAmode : 1; /* AMODE          */
            int Rsvd0 : 31;
            ??> s2;
        ??> u4;
    ??> s1;
??> u3;
union ??<
    struct ??<
        int Flags : 8; /* Flags          */
        int Rsvd0 : 24; /* Reserved      */
    ??> s3;
    struct ??<
        int ProcessARs : 1; /* If on, ARs will be processed. Otherwise
                            not. If not processed, ARs 0, 1, 14, and 15 are
                            unpredictable. ARs 2-13 are taken from the values
                            present when the service is entered.          */
        int Rsvd0 : 31; /* Reserved      */
    ??> s4;
??> u5;

void *AreaToFree; /* Address of area to free. If this is non-0
                  then the area will be freed using the subpool
                  specified in L16J.Subpool. This can be used
                  to free the caller's entire dynamic area if
                  so desired. When this option is specified, it
                  is necessary that the area begin on a
                  doubleword boundary.          */
int LengthToFree; /* Length of area to free, in bytes.          */
char Rsvd??(8??); /* Reserved          */
??> L16J;

/* Type for user supplied L16J1          */
typedef struct ??<
    int Version; /* Must be 1          */
    int Length; /* Initialize to CSRL16J1_LENGTH          */
    int SubPool; /* Subpool of storage to be freed          */
    char Rsvd1??(64??); /* Reserved. Must be zeroes.          @l11C*/
    union ??<
        char ARs??(64??); /* Access registers          */
        int  AR??(16??);  /* Access register 0-15      */
    ??> u2;
    char Rsvd2??(8??); /* Reserved. Must be zeroes.          @l11C*/
    union ??<
        struct ??<
            int Flags : 8; /* Flags          */
            int Rsvd0 : 24; /* Reserved. Must be zeroes.          */
        ??> 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;
    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.                                */
union ??<
    char G64Rs??(128??); /* 64-bit GPRs 0-15                                @L1A*/
    double G64R??(16??); /* General register 0-15                                @L1A*/
??> u1;
union ??<
    char PSWE??(16??); /* z/Architecture 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.                                @L1A*/

    struct ??<
        int PSWEByte0to3 : 32; /* First 4 bytes                                */
        union ??<
            int PSWEByte4to7 : 32; /* Second 4 bytes                                */
            struct ??<
                int PSWEAMode : 1; /* AMODE                                */
                int Rsvd0 : 31;
                ??> s2;
            ??> u4;
            char PSWEADDR??(8??); /* 8-byte instruction address                                */
            ??> s1;
            ??> u3;
        ??> L16J1;
    /*****
    *      Fixed Service Parameter and Return Code Defines
    *      *****/

#define CSRL16J_LENGTH          168      /* Length of L16J                                */
#define CSRL16J1_LENGTH        304      /* Length of L16J1                                @L1A*/

/* 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

/* Service Return Codes                                */
#define CSRLJ1_OK              0      /*                                @L1A*/
#define CSRLJ1_BAD_VERSION    4      /*                                @L1A*/
#define CSRLJ1_BAD_AMODE      8      /*                                @L1A*/
#define CSRLJ1_BAD_RESERVED   12     /*                                @L1A*/
#define CSRLJ1_BAD_LENGTH     16     /*                                @L1A*/
#define CSRLJ1_BAD_PSW        24     /*                                @L1A*/
#define CSRLJ1_NOT_ZARCHITECTURE 28   /*                                @L1A*/
#define CSRLJ1_NOT_ESAME      28     /*                                @L1A*/

/*****
*      Function Prototypes for Service Routines
*      *****/

extern void csrl16j(
    L16J *__L16J, /* Input - User supplied L16J block                                */
    int *__RC); /* Output - Return code                                */

extern void csrlj1(
    L16J1 *__L16J1, /* Input - User supplied L16J1 block                                */
    int *__RC); /* Output - Return code                                @L1A*/

/*****
#endif

```

CSRLJPLI provides the following declarations for use in your PL/I program:

```

/*****
 *      Constants for Fixed Return Codes      *
 *****/

/* Load 16 and Jump Service Return Codes */

%DCL CSRL16J_OK FIXED;
%CSRL16J_OK          = 0;

%DCL CSRL16J_BAD_VERSION FIXED;
%CSRL16J_BAD_VERSION = 4;

%DCL CSRL16J_BAD_AMODE FIXED;
%CSRL16J_BAD_AMODE   = 8;

%DCL CSRL16J_BAD_RESERVED FIXED;
%CSRL16J_BAD_RESERVED = 12;

%DCL CSRL16J_BAD_LENGTH FIXED;
%CSRL16J_BAD_LENGTH   = 16;

%DCL CSRL16J_BAD_PSW FIXED;
%CSRL16J_BAD_PSW      = 24;

%DCL CSRLJ1_OK FIXED;          /* @L1A*/
%CSRLJ1_OK                  /* @L1A*/

%DCL CSRLJ1_BAD_VERSION FIXED; /* @L1A*/
%CSRLJ1_BAD_VERSION         /* @L1A*/

%DCL CSRLJ1_BAD_AMODE FIXED;   /* @L1A*/
%CSRLJ1_BAD_AMODE           /* @L1A*/

%DCL CSRLJ1_BAD_RESERVED FIXED; /* @L1A*/
%CSRLJ1_BAD_RESERVED        /* @L1A*/

%DCL CSRLJ1_BAD_LENGTH FIXED;  /* @L1A*/
%CSRLJ1_BAD_LENGTH          /* @L1A*/

%DCL CSRLJ1_BAD_PSW FIXED;     /* @L1A*/
%CSRLJ1_BAD_PSW             /* @L1A*/

%DCL CSRLJ1_NOT_ZARCHITECTURE FIXED; /* @L1A*/
%CSRLJ1_NOT_ZARCHITECTURE = 28;      /* @L1A*/
%DCL CSRLJ1_NOT_ESAME FIXED; /* @L1A*/
%CSRLJ1_NOT_ESAME        = 28;      /* @L1A*/

/*****
 *      Service Entry Declarations      *
 *****/

DCL CSRL16J ENTRY
  (CHAR(168), /* Input - L16J */
   FIXED BIN(31)) /* Output - Return code */
  OPTIONS(INTER ASSEMBLER);

DCL CSRLJ1 ENTRY
  (CHAR(304), /* Input - L16J1 */
   FIXED BIN(31)) /* Output - Return code */
  OPTIONS(INTER ASSEMBLER); /* @L1A*/

/* End of Load 16 and Jump Service Declares */

```

Figure 21. CSRLJPLI declarations for return codes for PL/I

Restrictions

None.

Performance implications

None.

Syntax diagram

Code the invocation following the syntax of the language you are using. Specify parameters in the order shown.

C/370 syntax

Code	Parameters
csrl16j	(&L16J ,&return_code)
csrl1j1	(&L16J1 ,&return_code)

PL/I syntax

Code	Parameters
CALL CSRL16J	(L16J ,return_code)
CALL CSRLJ1	(L16J1 ,return_code)

Parameters

The parameters are explained as follows:

L16J/L16J1

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/CSRLJ1 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/CSRLJ1 service are as follows:

Table 53. CSRL16J/CSRLJ1 Return Codes	
Return Code (hexadecimal)	Meaning and Action
00	<p>Meaning: Successful completion. The calling program will never see this return code because it indicates that the target routine received control.</p> <p>Action: None.</p>

Table 53. CSRL16J/CSRLJ1 Return Codes (continued)

Return Code (hexadecimal)	Meaning and Action
04	<p>Meaning: The value specified in the L16JVERSION/L16J1VERSION field of the L16J/L16J1 data area was not a zero or one. The L16JVERSION/L16J1VERSION field must contain a value of zero or one. One indicates that CSRLJ1 is being called even if CSRL16J is identified.</p> <p>Action: When you build the L16J/L16J1 data area, first zero the entire L16J/L16J1 data area and then fill in the required fields. This process ensures that all fields that must contain zeroes are correct. Fill in L16J1VERSION as needed.</p>
08	<p>Meaning: The calling program was not in 31-bit addressing mode, which is required.</p> <p>Action: Make sure the calling program is in 31-bit addressing mode.</p>
0C	<p>Meaning: One of the fields in the L16J/L16J1 data area that is reserved for IBM use contained a nonzero value. Any field reserved for IBM use must contain a value of zero.</p> <p>Action: When you build the L16J/L16J1 data area, first zero the entire L16J/L16J1 data area and then fill in the required fields. This process ensures that all fields that must contain zeroes are correct</p>
10	<p>Meaning: The value specified in field L16JLENGTH/L16J1LENGTH in the L16J/L16J1 data area was less than the actual length of the L16J.</p> <p>Action: Make sure that the value in the L16JLENGTH/L16J1LENGTH field reflects the actual length of the L16J/L16J1 data area.</p>
18	<p>Meaning: The PSW provided in field L16JPSW/L16J1PSWE of the L16J/L16J1 data area specified an incorrect ASC mode.</p> <p>Action: In the L16JPSW/L16J1PSWE field, specify either primary or AR ASC mode.</p>

Example

The following example, coded in C/370 uses CSRL16J to transfer control to a C/370 program. The target routine executes in the mode and with the register contents specified by the calling program in the L16J parameter list.

This example performs the following operations:

- Fills in L16J parameter list with PSW and execution mode data.
- Calls an assembler routine to obtain the current register contents of registers 0 through 13 and copies them to the L16J parameter list.
- Defines the contents of registers 14 and 15 for the target routine.
- Issues setjmp to allow return from the target routine.
- Invokes the C/370 function L16JPrg through CSRL16J.
- CSRL16J issues longjmp to return to caller and complete processing.

To use this example, you must also use the assembler program following the C/370 example.

C/370 example program

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <setjmp.h>
#include "CSRLJC.H"

#define FALSE 0
#define TRUE 1

/* REG0T013 is the assembler assist routine (below) to extract
   registers 0 through 13, for C/370 addressability */
#pragma linkage(REG0T013,05)

int      rcode;
int      i;
unsigned int regs??(14??); /* Register save area */
```

```

jmp_buf      JumpBuffer;    /* Buffer for setjmp/longjmp */
L16J         L16JParmArea;  /* L16J parameter list structure */

/* Function prototype for function to be called via L16J */
void L16JPrg();

/* Invoke a C/370 function via L16J Callable Services */
main()
{
    /* Start by initializing the entire L16J parameter list */
    memset(&L16JParmArea,'\0',sizeof(L16J));

    /* The following fields were implicitly initialized to zero
       by the preceding statement:
       L16JParmArea.Version
       L16JParmArea.SubPool
       L16JParmArea.AreaToFree
       L16JParmArea.LengthToFree
       These field do not need to be explicitly set unless a value
       other than zero is required */

    /* Place parameter list length size into parameter list */
    L16JParmArea.Length = sizeof(L16J);

    /* Create a Problem State/Key 8 PSW */
    L16JParmArea.u3.s1.PSWByte0to3 = 0x078D1000;
    L16JParmArea.u3.s1.u4.PSWAddr = (void *) &L16JPrg;

    /* Mode data */
    L16JParmArea.u3.s1.u4.s2.PSWAmode = 1;
    L16JParmArea.u5.s4.ProcessARs = 1;

    /* Call assembler assist routine to obtain current register
       values */
    REG0T013(&regs);

    /* Place register values into parameter list */
    for (i=0;i<14;i++)
        L16JParmArea.u1.GR??(i??)= regs??(i??);

    /* Register 14 is not being used in this linkage, but we
       have set it to zero for this example */
    L16JParmArea.u1.GRAddr??(14??) = 0;

    /* Set register 15 for entry to routine */
    L16JParmArea.u1.GRAddr??(15??) = (void *) &L16JPrg;

    printf("L16JC - Call L16J to invoke L16JPrg\n");

    /* Use setjmp to allow return to this point in program. If
       setjmp is being called for the first time, invoke L16JPrg
       via L16J Callable Services. If returning from longjmp,
       skip call to L16J services and complete processing. */
    if (!setjmp(JumpBuffer))
    {
        csrl16j (&L16JParmArea,&rcode);

        /* Demonstrate use of L16J C/370 declares */
        switch (rcode)
        {
            /* Select on a particular return code value */
            case CSRL16J_BAD_PSW:
                printf("L16JC - L16J unsuccessful, bad PSW\n");
                break;
            /* Default error processing */
            default:
                printf("L16JC - L16J unsuccessful, RC = %d\n",rcode);
                break;
        }
    }
    printf("L16JC - Returned from L16JPrg\n");
}

/* The routine below receives control via L16J Callable Services.
   control is passed back to main via longjmp. */
void L16JPrg(void)
{
    printf("L16JC - L16JPrg got control\n");
    longjmp(JumpBuffer,1);
}

```


Assembler program for use with the C/370 example

To use this example you must assemble the following program and linkedit it with the C/370 program.

```
SR0T013 CSECT
SR0T013 AMODE 31
SR0T013 RMODE ANY
*
* Assembler assist routine to save contents of registers 0 through 13
* to the area pointed to by register 1.
*
REG0T013 DS 0H
          ENTRY REG0T013
* 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    SR0T013
```


Chapter 18. 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_ST SINOTAVAILABLE), 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:

Requirement	Details
Minimum authorization:	Problem state, key 8–15
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit when using the CALL CSRSI form (or csrsi in C), 31-bit when using an alternate form
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold a LOCAL lock, the CMS lock, or the CPU lock but is not required to hold any locks.

Programming requirements

The caller should include the CSRSIIDF macro to map the returned information and to provide equates for the service.

Restrictions

None.

Input register information

The caller is not required by the system to set up any registers.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

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

Syntax

CALL statement	Parameters
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* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).
- If you are coding in C, use include file CSRSIC.

The following table describes the return codes, shown in decimal.

Return Code and Equate Symbol	Meaning and Action
00 (0) CSRSI_SUCCESS	Meaning: The CSRSI service completed successfully. All information requested was returned. Action: Check the si00validityflags field to determine the validity of each returned area.
04 (4) CSRSI_STSinOTAVAILABLE	Meaning: The CSRSI service completed successfully, but since the Store System Information (STSI) instruction was not available, only the SI00PCCACPID, SI00PCCACPUA, and SI00PCCACAFM fields are valid. Action: None required.
08 (8) CSRSI_SERVICenOTAVAILABLE	Meaning: Environmental error: The CSRSI service is not available on this system. Action: Avoid calling the CSRSI service unless running on a system on which it is available.
12 (C) CSRSI_BADREQUEST	Meaning: User error: The request parameter did not specify a word formed from any combination of CSRSI_Request_V1CPC_Machine, CSRSI_Request_V2CPC_LPAR, and CSRSI_Request_V3CPC_VM. Action: Correct the parameter.
16 (10) CSRSI_BADINFOAREALEN	Meaning: User error: The Infoarealen parameter did not match the length of the area required to return the requested information. Action: Correct the parameter.
20 (14) CSRSI_BADLOCK	Meaning: User error: The service was called while holding a system lock other than CPU, LOCAL/CML, or CMS. Action: Avoid calling in this environment.

CSRSIC C/370 header file

For the C programmer, include file CSRSIC provides equates for return codes and data constants, such as Register service request types. To use CSRSIC, copy the file from SYS1.SAMPLIB to the appropriate local C library. Here are the contents of the file:

```
#ifndef __CSRSI
#define __CSRSI

/*****
 *      Type Definitions for User Specified Parameters
 *****/

/* Type for Request operand of CSRSI */
typedef int  CSRSIRequest;

/* Type for InfoAreaLen operand of CSRSI */
typedef int  CSRSIInfoAreaLen;

/* Type for Return Code */
typedef int  CSRSIReturnCode;

/*****
```

```

*          Function Prototypes for Service Routines          *
*****/

#ifdef __cplusplus
extern "OS" ??<
#else
#pragma linkage(CSRSI_calltype,OS)
#endif
typedef void CSRSI_calltype(
    CSRSIRequest      __REQUEST,    /* Input - request type          */
    CSRSIInfoAreaLen  __INFOAREALEN, /* Input - length of infoarea    */
    void              *__INFOAREA,  /* Input - info area            */
    CSRSIReturnCode   *__RC);       /* Output - return code         */

extern CSRSI_calltype csrsi;

#ifdef __cplusplus
??>
#endif

#ifdef __cplusplus
#define csrsi_byaddr(Request, Flen, Fptr, Rcptr) \
??< \
    struct CSRSI_PSA* CSRSI_pagezero = 0; \
    CSRSI_pagezero->CSRSI_cvt->CSRSI_cvtsrtr->CSRSI_addr \
    (Request,Flen,Fptr,Rcptr); \
??>;
#endif
??
>;

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

    struct CSRSI_CVT ??<
        unsigned char CSRSI_cvt_filler1 ??(116??);
        struct ??<
            int CSRSI_cvtdcb_rsvd1 : 4;    /* Not needed          */
            int CSRSI_cvtosex : 1;    /* If on, indicates that the \
                                     CVTOSLVL fields are valid */
            int CSRSI_cvtdcb_rsvd2 : 3;    /* Not needed          */
            ??> CSRSI_cvtdcb;
            unsigned char CSRSI_cvt_filler2 ??(427??);
            struct CSRSI_CSRT * CSRSI_cvtsrtr;
            unsigned char CSRSI_cvt_filler3 ??(716??);
            unsigned char CSRSI_cvtoislvl0;
            unsigned char CSRSI_cvtoislvl1;
            unsigned char CSRSI_cvtoislvl2;
            unsigned char CSRSI_cvtoislvl3;
            struct ??<
                int CSRSI_cvtsrtr : 1;    /* If on, indicates that the \
                                         CSRSI service is available */
                int CSRSI_cvtoislvl_rsvd1 : 7;    /* Not needed          */
                ??> CSRSI_cvtoislvl4;
            unsigned char CSRSI_cvt_filler4 ??(11??);    /* \
??>;

struct CSRSI_PSA ??<
    char CSRSI_psa_filler??(16??);
    struct CSRSI_CVT* CSRSI_cvt;
??>;

/* End of CSRSI Header */

#endif

/*****/
/* s11lv1 represents the output for a V1 CPC when general CPC \
/* information is requested \
/*****/

typedef struct ??<
    unsigned char _filler1??(32??); /* Reserved */
    unsigned char s11lv1cpcmanufacturer??(16??); /* \
The 16-character (0-9 \
or uppercase A-Z) EBCDIC name \
of the manufacturer of the V1 \
CPC. The name is

```

```

left-justified with trailing
blank characters if necessary.
unsigned char  si11v1cpctype??(4??); /* The 4-character (0-9) EBCDIC
                                     type identifier of the V1 CPC. */
unsigned char  _filler2??(12??); /* Reserved */
unsigned char  si11v1cpcmodel??(16??); /* The 16-character (0-9 or
                                     uppercase A-Z) EBCDIC model
                                     identifier of the V1 CPC. The
                                     identifier is left-justified
                                     with trailing blank characters
                                     if necessary. */
unsigned char  si11v1cpcsequencecode??(16??); /*
                                     The 16-character (0-9
                                     or uppercase A-Z) EBCDIC
                                     sequence code of the V1 CPC.
                                     The sequence code is
                                     right-justified with leading
                                     EBCDIC zeroes if necessary. */
unsigned char  si11v1cpcplantofmanufacture??(4??); /* The 4-character
                                     (0-9 or uppercase A-Z) EBCDIC
                                     plant code that identifies the
                                     plant of manufacture for the
                                     V1 CPC. The plant code is
                                     left-justified with trailing
                                     blank characters if necessary. */
unsigned char  _filler3??(3996??); /* Reserved */
??> si11v1;

/*****
/* si22v1 represents the output for a V1 CPC when information
/* is requested about the set of CPUs
/* *****/

typedef struct ??<
unsigned char  _filler1??(32??); /* Reserved */
unsigned char  si22v1cpucapability??(4??); /*
                                     An unsigned binary integer
                                     that specifies the capability
                                     of one of the CPUs contained
                                     in the V1 CPC. It is used as
                                     an indication of the
                                     capability of the CPU relative
                                     to the capability of other CPU
                                     models. */
unsigned int    si22v1totalcpucount      : 16; /* A 2-byte
                                     unsigned integer
                                     that specifies the
                                     total number of CPUs contained
                                     in the V1 CPC. This number
                                     includes all CPUs in the
                                     configured state, the standby
                                     state, and the reserved state. */
unsigned int    si22v1configuredcpucount : 16; /* A 2-byte
                                     unsigned binary
                                     integer that specifies
                                     the total number of CPUs that
                                     are in the configured state. A
                                     CPU is in the configured state
                                     when it is described in the
                                     V1-CPC configuration
                                     definition and is available to
                                     be used to execute programs. */
unsigned int    si22v1standbycpucount    : 16; /* A 2-byte
                                     unsigned integer
                                     that specifies the
                                     total number of CPUs that are
                                     in the standby state. A CPU is
                                     in the standby state when it
                                     is described in the V1-CPC
                                     configuration definition, is
                                     not available to be used to
                                     execute programs, but can be
                                     used to execute programs by
                                     issuing instructions to place

```



```

it in the configured state.
                                                                    */
unsigned int    si22v1reservedcpucount    : 16; /* A 2-byte
unsigned binary
integer that specifies
the total number of CPUs that
are in the reserved state. A
CPU is in the reserved state
when it is described in the
V1-CPC configuration
definition, is not available
to be used to execute
programs, and cannot be made
available to be used to
execute programs by issuing
instructions to place it in
the configured state, but it
may be possible to place it in
the standby or configured
state through manually
initiated actions
                                                                    */
struct ??<
    unsigned char    _si22v1mpcpucapaf??(2??); /* Each individual
adjustment factor.
                                                                    */
    unsigned char    _filler2??(4050??);
??> si22v1mpcpucapafs;
??> si22v1;

#define si22v1mpcpucapaf    si22v1mpcpucapafs._si22v1mpcpucapaf

/*****
/* si22v2 represents the output for a V2 CPC when information
/* is requested about the set of CPUs
/*
/*****
typedef struct ??<
    unsigned char    _filler1??(32??); /* Reserved
                                                                    */
    unsigned int    si22v2cpcnumber    : 16; /* A 2-byte
unsigned integer
which is the number of
this V2 CPC. This number
distinguishes this V2 CPC from
all other V2 CPCs provided by
the same logical-partition
hypervisor
                                                                    */
    unsigned char    _filler2; /* Reserved
                                                                    */
struct ??<
    unsigned int    _si22v2lcpudedicated    : 1; /*
When one, indicates that
one or more of the logical
CPUs for this V2 CPC are
provided using V1 CPUs that
are dedicated to this V2 CPC
and are not used to provide
logical CPUs for any other V2
CPCs. The number of logical
CPUs that are provided using
dedicated V1 CPUs is specified
by the dedicated-LCPU-count
value. When zero, bit 0
indicates that none of the
logical CPUs for this V2 CPC
are provided using V1 CPUs
that are dedicated to this V2
CPC.
                                                                    */
    unsigned int    _si22v2lcpushared    : 1; /*
When one, indicates that
or more of the logical CPUs
for this V2 CPC are provided
using V1 CPUs that can be used
to provide logical CPUs for
other V2 CPCs. The number of
logical CPUs that are provided
using shared V1 CPUs is
specified by the
shared-LCPU-count value. When
zero, it indicates that none
of the logical CPUs for this
V2 CPC are provided using
shared V1 CPUs.
                                                                    */

```

```

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    si22v2totalcpucount        : 16; /*
A 2-byte unsigned
integer that specifies the
total number of logical CPUs
that are provided for this V2
CPC. This number includes all
of the logical CPUs that are
in the configured state, the
standby state, and the
reserved state. */

unsigned int    si22v2configuredlcpucount  : 16; /*
A 2-byte unsigned
binary integer that specifies
the total number of logical
CPUs for this V2 CPC that are
in the configured state. A
logical CPU is in the
configured state when it is
described in the V2-CPC
configuration definition and
is available to be used to
execute programs. */

unsigned int    si22v2standbylcpucount     : 16; /*
A 2-byte unsigned
binary integer that specifies
the total number of logical
CPUs that are in the standby
state. A logical CPU is in the
standby state when it is
described in the V2-CPC
configuration definition, is
not available to be used to
execute programs, but can be
used to execute programs by
issuing instructions to place
it in the configured state. */

unsigned int    si22v2reservedlcpucount    : 16; /*
A 2-byte unsigned
binary integer that specifies
the total number of logical
CPUs that are in the reserved
state. A logical CPU is in the
reserved state when it is
described in the V2-CPC
configuration definition, is
not available to be used to
execute programs, and cannot
be made available to be used
to execute programs by issuing
instructions to place it in
the configured state, but it
may be possible to place it in
the standby or configured
state through manually
initiated actions */

unsigned char    si22v2cpcname??(16??); /*
The 8-character EBCDIC name of
this V2 CPC. The name is
left-justified with trailing
blank characters if necessary. */

unsigned char    si22v2cpccapabilityaf??(4??); /* Capability Adjustment
Factor (CAF). An unsigned
binary integer of 1000 or
less. The adjustment factor

```

```

        specifies the amount of the
        V1-CPC capability that is
        allowed to be used for this V2
        CPC by the logical-partition
        hypervisor. The fraction of
        V1-CPC capability is
        determined by dividing the CAF
        value by 1000. */
unsigned char _filler4??(16??); /* Reserved */
unsigned int si22v2dedicatedlcpucount : 16; /*
        A 2-byte unsigned
        binary integer that specifies
        the number of configured-state
        logical CPUs for this V2 CPC
        that are provided using
        dedicated V1 CPUs. (See the
        description of bit
        si22v2lcpudedicated.) */

unsigned int si22v2sharedlcpucount : 16; /*
        A 2-byte unsigned
        integer that specifies the
        number of configured-state
        logical CPUs for this V2 CPC
        that are provided using shared
        V1 CPUs. (See the description
        of bit si22v2lcpushared.)

        */
unsigned char _filler5??(4012??); /* Reserved */
??> si22v2;

#define si22v2lcpudedicated si22v2lcpuc._si22v2lcpudedicated
#define si22v2lcpushared si22v2lcpuc._si22v2lcpushared
#define si22v2lcpuulimit si22v2lcpuc._si22v2lcpuulimit

/*****
/* si22v3db is a description block that comprises part of the */
/* si22v3 data. */
/*****/

typedef struct ??<
    unsigned char _filler1??(4??); /* Reserved */
    unsigned int si22v3dbtotalcpucount : 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.
        */
    unsigned int si22v3dbstandbylcpucount : 16; /*
        A 2-byte unsigned
        binary integer that specifies
        the number of logical CPUs for
        this V3 CPC that are in the
        standby state. A logical CPU
        is in the standby state when
        it is described in the V3-CPC
        configuration definition, is
        not available to be used to
        execute programs, but can be
        used to execute programs by
        issuing instructions to place
        it in the configured state.
        */
    unsigned int si22v3dbreservedlcpucount : 16; /*

```

```

A 2-byte unsigned
binary integer that specifies
the number of logical CPUs for
this V3 CPC that are in the
reserved state. A logical CPU
is in the reserved state when
it is described in the V2-CPC
configuration definition, is
not available to be used to
execute programs, and cannot
be made available to be used
to execute programs by issuing
instructions to place it in
the configured state, but it
may be possible to place it in
the standby or configured
state through manually
initiated actions */
unsigned char  si22v3dbcpcname??(8??); /* The 8-character EBCDIC name
of this V3 CPC. The name is
left-justified with trailing
blank characters if necessary. */

unsigned char  si22v3dbcpccaf??(4??); /* A 4-byte unsigned binary
integer that specifies an
adjustment factor. The
adjustment factor specifies
the amount of the V1-CPC or
V2-CPC capability that is
allowed to be used for this V3
CPC by the
virtual-machine-hypervisor
program. */

unsigned char  si22v3dbvmhpidentifier??(16??); /* The 16-character
EBCDIC identifier of the
virtual-machine-hypervisor
program that provides this V3
CPC. (This identifier may
include qualifiers such as
version number and release
level). The identifier is
left-justified with trailing
blank characters if necessary. */

unsigned char  _filler2??(24??); /* Reserved */
??> si22v3db;
/*****
/* si22v3 represents the output for a V3 CPC when information
/* is requested about the set of CPUs
*****/

typedef struct ??<
    unsigned char  _filler1??(28??); /* Reserved */
    unsigned char  _filler2??(3??); /* Reserved */
    struct ??<
        unsigned int  _filler3          : 4; /* Reserved */
        unsigned int  _si22v3dbcount    : 4; /*
Description Block Count. A
4-bit unsigned binary integer
that indicates the number (up
to 8) of V3-CPC description
blocks that are stored in the
si22v3dbe array. */
??> si22v3dbcountfield; /*
si22v3db  si22v3dbe??(8??); /* Array of entries. Only the number
indicated by si22v3dbcount
are valid */
    unsigned char  _filler5??(3552??); /* Reserved */
??> si22v3;

#define si22v3dbcount    si22v3dbcountfield._si22v3dbcount

/*****
/* SI00 represents the "starter" information. This structure is
/* part of the information returned on every CSRSI request.
*****/

typedef struct ??<
    char            si00cpcvariety; /* SI00CPCVariety_V1CPC_MACHINE,

```

```

SI00CPCVariety_V2CPC_LPAR, or
SI00CPCVariety_V3CPC_VM */
struct ??<
    int     _si00validsi11v1 : 1; /* si11v1 was requested and
                                   the information returned is valid */
    int     _si00validsi22v1 : 1; /* si22v2 was requested and
                                   the information returned is valid */
    int     _si00validsi22v2 : 1; /* si22v2 was requested and
                                   the information returned is valid */
    int     _si00validsi22v3 : 1; /* si22v3 was requested and
                                   the information returned is valid */
    int     _filler1         : 4; /* Reserved */
    ??> si00validityflags;
    unsigned char _filler2??(2??); /* Reserved */
    unsigned char si00pccacpid??(12??); /* PCCACPID value for this CPU */
    unsigned char si00pccacpua??(2??); /* PCCACPUA value for this CPU */
    unsigned char si00pccacafm??(2??); /* PCCACAFM value for this CPU */
    unsigned char _filler3??(4??); /* Reserved */
    unsigned char si00lastupdatetimestamp??(8??); /* Time of last STSI
                                                    update, via STCK */
    unsigned char _filler4??(32??); /* Reserved */
    ??> si00;

#define si00validsi11v1      si00validityflags._si00validsi11v1
#define si00validsi22v1      si00validityflags._si00validsi22v1
#define si00validsi22v2      si00validityflags._si00validsi22v2
#define si00validsi22v3      si00validityflags._si00validsi22v3

/*****
/* siv1 represents the information returned when V1CPC_MACHINE
/* data is requested
*****/

typedef struct ??<
    si00 siv1si00;                                /* Area mapped by
                                                    struct si00 */
    si11v1 siv1si11v1;                            /* Area
                                                    mapped by struct si11v1 */
    si22v1 siv1si22v1;                            /* Area
                                                    mapped by struct si22v1 */
    ??> siv1;

/*****
/* siv1v2 represents the information returned when V1CPC_MACHINE
/* data and V2CPC_LPAR data is requested
*****/

typedef struct ??<
    si00 siv1v2si00;                                /* Area mapped by
                                                    by struct si00 */
    si11v1 siv1v2si11v1;                            /* Area
                                                    mapped by struct si11v1 */
    si22v1 siv1v2si22v1;                            /* Area
                                                    mapped by struct si22v2 */
    si22v2 siv1v2si22v2;                            /* Area
                                                    mapped by struct si22v2 */
    ??> siv1v2;

/*****
/* siv1v2v3 represents the information returned when V1CPC_MACHINE
/* data, V2CPC_LPAR data and V3CPC_VM data is requested
*****/

typedef struct ??<
    si00 siv1v2v3si00;                                /* Area
                                                    mapped by struct si00 */
    si11v1 siv1v2v3si11v1;                            /* Area
                                                    mapped by struct si11v1 */
    si22v1 siv1v2v3si22v1;                            /* Area
                                                    mapped by struct si22v1 */
    si22v2 siv1v2v3si22v2;                            /* Area
                                                    mapped by struct si22v2 */
    si22v3 siv1v2v3si22v3;                            /* Area
                                                    mapped by struct si22v3 */

```

```

??> siv1v2v3;

/*****
/* siv1v3 represents the information returned when V1CPC_MACHINE
/* data and V3CPC_VM data is requested
*****/

typedef struct ??<
    si00 siv1v3si00;                                /* Area mapped
                                                    by struct si00 */
    si11v1 siv1v3si11v1;                            /* Area
                                                    mapped by struct si11v1 */
    si22v1 siv1v3si22v1;                            /* Area
                                                    mapped by struct si22v1 */
    si22v3 siv1v3si22v3;                            /* Area
                                                    mapped by struct si22v3 */
??> siv1v3;

/*****
/* 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;

/*****
*      Fixed Service Parameter and Return Code Defines
*****/

/* SI00 Constants */

#define SI00CPCVARIETY_V1CPC_MACHINE 1
#define SI00CPCVARIETY_V2CPC_LPAR 2
#define SI00CPCVARIETY_V3CPC_VM 3

/* CSRSI Constants */

#define CSRSI_REQUEST_V1CPC_MACHINE 1
#define CSRSI_REQUEST_V2CPC_LPAR 2
#define CSRSI_REQUEST_V3CPC_VM 4

/* CSRSI Return codes */

#define CSRSI_SUCCESS 0
#define CSRSI_STSinOTAVAILABLE 4
#define CSRSI_SERVICENOTAVAILABLE 8
#define CSRSI_BADREQUEST 12
#define CSRSI_BADINFOAREALEN 16
#define CSRSI_BADLOCK 20

```

Part 8. Base Control Program internal interface (BCPii) services

Chapter 19. 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 z System hardware base through a set of application program interfaces. These applications can access the z System hardware that the application is running on and extend their reach to other z System 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 z System topology of the current interconnected Central Processor Complexes (CPCs) as well as the images, capacity records, activation profiles, and user-defined image group, group profile and LPAR Capacity group defined on a particular CPC.
- Query CPC, image (LPAR), capacity record, activation profile, user-defined image group, group profile and LPAR Capacity group information.
- Set various configuration values related to CPC, image and activation profiles.
- Issue commands against CPCs, images (LPARs), and user-defined image groups 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 z System hardware communication from any other network traffic within the intranet/internet.

Calls using the BCPii Application Programming Interfaces (APIs) can be made from the C, the REXX, or the assembler programming languages. See [“Syntax, linkage and programming considerations” on page 263](#) 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 an installation begins to issue BCPii APIs, a series of setup and installation steps must be performed. A summary of these steps follows. For additional details on each of these steps, see the supporting documentation that explains how each of these steps is accomplished:

1. Configure the local Support Element (SE) to support BCPii:
 - a. Check the levels of hardware that BCPii supports
 - b. Set up BCPii firmware security
 - c. Grant permission to BCPii requests
 - i) For z13 and lower CPCs, enable cross-partition authority for each image (LPAR) that you want to grant BCPii access
 - ii) For z14 and higher CPCs, set the BCPii security settings using the System Details task (CPC permission) and/or Change LPAR Security task (Image/LPAR permission).
 - d. Define an uppercase BCPii SNMP community name on the SE.

See [“Setting up connectivity to the support element” on page 246](#) for details.
2. Authorize an application to use BCPii, including authority to specific resources (such as CPCs, images and capacity records):

- a. Check that the BCPii application is program-authorized.
- b. Check that the BCPii application has general authority to use BCPii.
- c. Authorize the BCPii application to access the particular resource that requires BCPii service.
- d. Define an uppercase BCPii SNMP community name in the security product for each CPC as it was defined on the SE. Use the APPLDATA field with the CPC profile definition to associate a BCPii SNMP community name with a particular CPC.

These steps enable communication to the local CPC and allows the BCPii address space to initialize. See [“Setting up authority to use BCPii” on page 251](#) for details.

3. Configure the BCPii address space. See [“BCPii configuration considerations” on page 256](#) for details.
4. If the caller is running in a z/OS UNIX System Services environment, set up the notification mechanism to allow hardware and software events to be propagated to the z/OS UNIX application. See [“Setting up event notification for BCPii z/OS UNIX applications” on page 258](#) for details.
5. If the installation allows TSO/E users to have access to the BCPii APIs using REXX, see [“Setting up an environment to run BCPii TSO/E REXX execs” on page 259](#).

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 260](#) for details.

Figure 22 on page 246 shows the steps needed to setup and install BCPii.

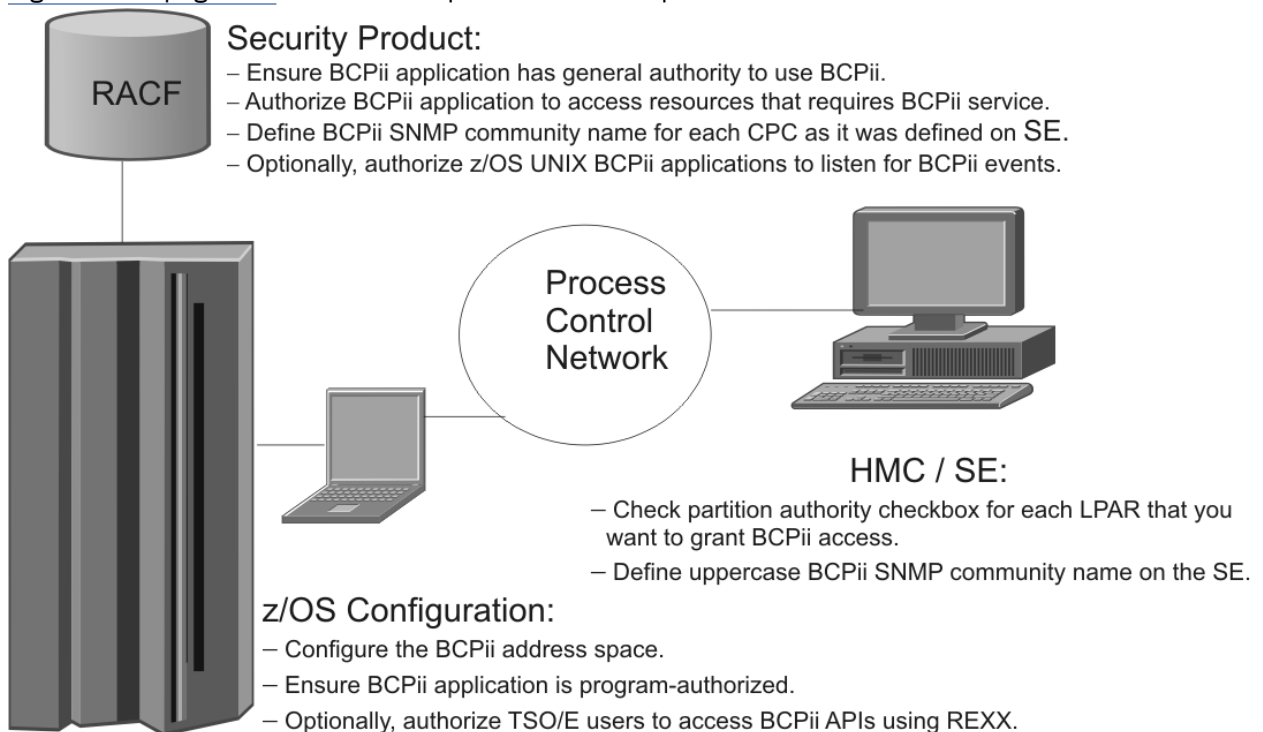


Figure 22. BCPii setup and installation steps

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.

Note: In order to customize the API settings controls on the SE, your userid must have administrator rights to access these panels.

Levels of hardware that BCPii supports

The HWIBCPii address space, which supports the issuing of BCPii APIs from a z/OS image, will run on any hardware that supports a level of the z/OS operating system in which BCPii is included. However, there will be some reduced BCPii functionality when a BCPii request targets a system that is not running on a zEnterprise machine. The BCPii restrictions increase the further downlevel the hardware is from a zEnterprise machine. To run with the fewest functionality restrictions possible, make sure the recommended microcode levels are installed for that SE, HMC and LPAR hardware.

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.

Note:

- IBM z15 (HMC Version 2.15.0) supports n-2 system levels only (IBM z13® and IBM z14). IBM z14 (HMC Version 2.14.1) is the last level to support four generations of systems (n through n-4).
- For optional use of JSON Web Tokens (JWTs) for JWT based authorization to resources, a minimum hardware level of an IBM z17® or higher is required.

The HWICMD / HWICMD2 services are 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 earlier than System z9. See [“HWICMD / HWICMD2 — Issue a BCPii hardware management command”](#) on page 274 for further information.

The HWIREST service is only allowed to be targeted to at least an IBM z15. See [“HWIREST — Issue RESTlike requests to the SE”](#) on page 378 for details regarding the specific minimum microcode level requirements.

Consult Table 54 on page 247 to determine the minimum level of microcode required to run BCPii on a specific hardware level.

Table 54. Minimum BCPii microcode levels by SE hardware level

SE hardware level	Minimum microcode level
IBM System z9 Driver 67	MCL 258 in the G40965 (SE-SYSTEM) EC stream
IBM System z10® Driver 79	MCL 163 in the N24409 (SE-SYSTEM) EC stream
IBM zEnterprise 196	MCL 220 in the N29802 (SE-SYSTEM) EC stream
IBM zEnterprise EC12	Any level
IBM z13 GA2	MCL 315 in the P00339 (SE-SYSTEM) EC stream
IBM z14	MCL 059 in the P42601 (SE-SYSTEM) EC stream
IBM z14 GA2	MCL 219 in the P41414 (SE-SYSTEM) EC stream
IBM z15 and higher	Any level

Consult Table 55 on page 247 to determine the minimum level of microcode required to run BCPii on a specific HMC level.

Table 55. Minimum BCPii microcode levels by HMC level

HMC level	Minimum microcode level
IBM System z9 Driver 67	MCL 158 in the G40969 (HMC-SYSTEM) EC stream
IBM System z10 Driver 79	MCL 034 in the N24415 (HMC-SYSTEM) EC stream
IBM zEnterprise 196	Any level
IBM zEnterprise EC12	Any level

Consult [Table 56 on page 248](#) to determine the minimum level of microcode required to run BCPii on a specific LPAR level.

Table 56. Minimum BCPii microcode levels by LPAR level

LPAR level	Minimum microcode level
IBM System z9 Driver 67	MCL 008 in the G40954 (LPAR) EC stream
IBM System z10 Driver 79	MCL 002 in the N24404 (LPAR) EC stream
IBM zEnterprise 196	Any level
IBM zEnterprise EC12	Any level

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

It is necessary to grant authority on the support element (SE) to allow the support element to accept BCPii APIs flowing from the user application through the HWIBCPii address space.

The methodology to enable the firmware to accept and target BCPii API requests varies, depending on whether the CPC is lower than a z14 or if it is z14 or higher.

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.

Firmware security settings on CPCs lower than z14

For all CPC levels lower than a z14, it is necessary to enable cross partition authority for all images (LPARs) that want to use BCPii or want to be the target of BCPii requests. To change this setting, perform the following steps on the HMC:

1. Select the CPC that is required.
2. Open Single Object Operations.
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.

Firmware security settings on z14 and higher CPCs

Starting with the z14 machine, BCPii firmware security has been greatly enhanced to allow much more robust and granular controls of BCPii authority to both the CPC and LPARs. It is now possible to control which LPARs can access CPC attributes and commands through BCPii as well as which LPARs can access other LPARs through BCPii. **The cross partition authority checkbox no longer provides the security control to allow BCPii requests.** On the HMC or SE, use the instructions below to configure the desired security controls for the target CPC or LPARs.

Migrating from a pre-z14 to a z14 and higher machine

If you are migrating from a pre-z14 machine to a z14 or higher machine, the firmware will auto-migrate your settings so that the same security capabilities are in place on the z14 in the new BCPii security settings as you had using the cross partition authority checkbox on the pre-z14 machine. Once the migration is complete, it is important to review these migrated security settings, and to make changes as

necessary to refine the authority to be as granular as your installation needs. See [“Manual BCPii firmware security configuration”](#) on page 249.

Manual BCPii firmware security configuration

If you are manually updating your BCPii firmware security settings, there are a number of permissions that can be granted or denied. These include:

- which LPARs can issue BCPii requests
- which LPARs are allowed to access CPC attributes and commands, and
- which LPARs are allowed to access other LPAR's attributes and commands.

Setting BCPii firmware security access to the CPC

1. From the HMC or SE, select System(s) Management, select the CPC and then the System Details task.
2. Select the Security tab.
3. Check the Enable the system to receive commands from partitions checkbox and select the All partitions radio button to allow the CPC to receive commands (BCPii requests) from any BCPii enabled partition or limit which partitions can issue BCPii requests by clicking the Selected partitions radio button and specifying the specific list of LPARs to be granted this authority.
4. Click OK.

Setting BCPii firmware security access for each LPAR

1. Open Change LPAR Security from the CPC Operational Customization task.
2. For each image (LPAR) that you want to grant BCPii access:
 - a. Click the current BCPii enablement level in the BCPii Permissions column. The Configure BCPii Permission panel will be displayed.
 - b. Check the checkbox for Enable the partition to send commands to grant authority for this LPAR to send BCPii requests to other CPCs and LPARs.

Note: This checkbox must be enabled for BCPii address space to be active on this LPAR.
 - c. Check the Enable the partition to receive commands from other partitions check box and select the All partitions radio button to grant permission to all partitions on all CPCs to target this LPAR with BCPii requests or limit which partitions can issue BCPii requests by clicking the Selected Partitions radio button and specifying the specific list of LPARs to be granted this authority. Click OK.
 - d. When Change LPAR Security displays, update the BCPii permission by choosing Save and Change, Change Running System or Save to Profiles.

BCPii permissions can also be set via the Image Activation Profile when multiple LPARs need to be configured. See *zSystem Hardware Management Console Security* and *zEnterprise System Processor Resource/Systems Manager Planning Guide* as well as the Help panels from the HMC or SE for more information.

See the HMC book and *Support Element Operations Guide* for more information regarding changing the support element settings. For z14 and higher machines, see [IBM Documentation \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos) for more details.

Failure to set the security control access 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.
2. Open Single Object Operations.
3. Select Tasks Index in the navigation panel.
4. Select Customize API Settings task from the tasks list.

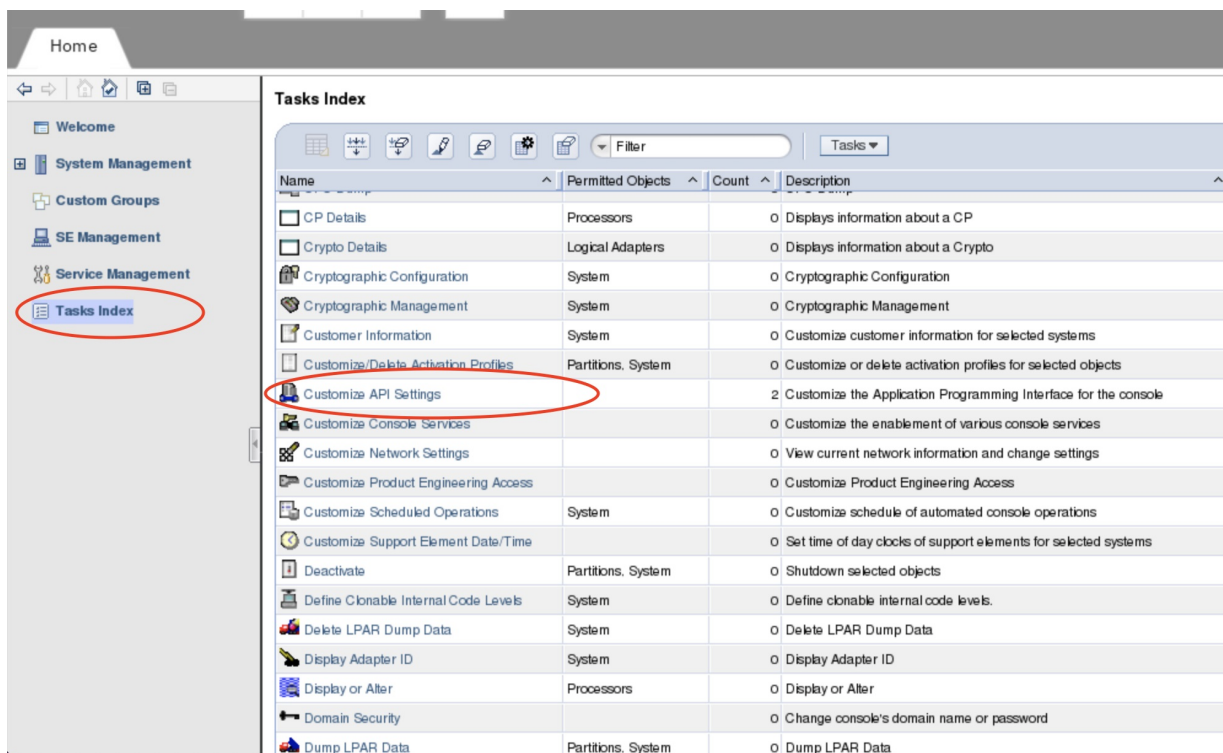


Figure 23. Tasks index

5. Select Enable.
6. 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.
7. Make sure that the SNMP agent parameters are blank.
8. 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. Only uppercase letters and numbers are allowed. 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 254](#) 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

9. Save the changes.

IBM Support Element

Home Customize API Settings

TSNCAR21: Customize API Settings

SNMP

☒ Enable

SNMP agent parameters:

☒ Allow capacity change API requests

Community Names

Select	Name	Address	Network Mask / Prefix	Access Type
<input checked="" type="radio"/>	SCOUT	127.0.0.1	255.255.255.255	write

Add... Change... Delete

SNMPv3 Users

Select	User Name	Access Type
<input type="button" value="Add..."/>	<input type="button" value="Change..."/>	<input type="button" value="Delete"/>

Event Notification Information

Specify any additional locations where SNMP trap messages will be sent.

Select	TCP/IP Address	Port number
<input type="button" value="Add..."/>	<input type="button" value="Change..."/>	<input type="button" value="Delete"/>

OK Cancel Help

Figure 24. Customize API settings

See *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 failure and you cannot start the address space. Message HWI022I 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 254 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.

Setting up authority to use the BCPII DISPLAY command

See [authority to use the BCPII DISPLAY system command](#).

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. The RACF syntax is as follows:

```
RDEFINE FACILITY HWI.APPLNAME.HWISERV UACC(NONE)
PERMIT HWI.APPLNAME.HWISERV CLASS(FACILITY) ID(userid) ACCESS(READ)
SETOPTS RACLIST(FACILITY) REFRESH
```

This RACF example allows 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)
SETOPTS RACLIST(FACILITY) REFRESH
```

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

This RACF example defines user IDs BCPii and HWISTART to the security product:

```
ADDUSER BCPii DFLTGRP(SYS1)
RDEFINE STARTED BCPii.** STDATA(USER(BCPii) GROUP(SYS1))
ADDUSER HWISTART DFLTGRP(SYS1)
RDEFINE STARTED HWISTART.** STDATA(USER(BCPii) GROUP(SYS1))
SETOPTS RACLIST(STARTED) REFRESH
```

Authority to the particular resource

There are two mechanisms for checking authority to a particular resource in BCPii.

- The legacy method utilizes FACILITY class profiles for all non-HWIREST / non-HWIREST2 callable services. This method may be used with certain HWIREST calls unless configured otherwise.
- The alternative mechanism for checking authority to a particular resource in BCPii utilizes JSON Web Tokens (JWTs) enabling the use of HMC authority controls. This method of authority checking is always used for HWIREST2 and certain HWIREST requests, and may be configured for all HWIREST requests.

Authority checking using FACILITY class profiles

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

Request Type	FACILITY Class Profile Required
CPC	HWI.TARGET. <i>netid.nau</i> where <i>netid.nau</i> represents the 3- to 17-character SNA name of the particular CPC.
Image	HWI.TARGET. <i>netid.nau.imagename</i> where <i>netid.nau</i> represents the 3- to 17-character SNA name of the particular CPC and <i>imagename</i> represents the 1- to 8-character LPAR name.

Request Type	FACILITY Class Profile Required
Capacity record	HWI.CAPREC. <i>netid.nau.caprec</i> where <i>netid.nau</i> represents the 3– to 17–character SNA name of the particular CPC and <i>caprec</i> represents an 8–character capacity record name.
Activation profiles	HWI.TARGET. <i>netid.nau</i> where <i>netid.nau</i> represents the 3– to 17–character SNA name of the particular CPC the activation profile is defined.
User-defined image groups	HWI.TARGET. <i>netid.nau</i> where <i>netid.nau</i> represents the 3– to 17–character SNA name of the particular CPC the user-defined image group is defined.
Group profile	HWI.TARGET. <i>netid.nau</i> where <i>netid.nau</i> represents the 3– to 17–character SNA name of the particular CPC the group profile is defined.
LPAR Capacity Group	HWI.TARGET. <i>netid.nau</i> where <i>netid.nau</i> represents the 3– to 17–character SNA name of the particular CPC the LPAR Capacity group is defined.

Note: For compatibility with security products, BCPII automatically transforms the following names to all uppercase characters: CPC names (including the local CPC name represented by '*'), image names, and capacity record names specified on the HWICONN service.

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. The RACF syntax is as follows:

```
RDEFINE FACILITY HWI.TARGET.netid.nau UACC(NONE) APPLDATA('uppercasecommunityname')
PERMIT HWI.TARGET.netid.nau CLASS(FACILITY) ID(userid) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

where *netid.nau* represents the 3 to 17 character SNA name of the CPC.

This RACF example allows 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” on page 254](#) 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 RACLIST(FACILITY) REFRESH
```

This RACF example grants user JOE with Command, Connect, Event, List, Query, and Set 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 RACLIST(FACILITY) REFRESH
```

Authority checking using JWTs

When using JSON Web Tokens (JWTs) based authorization, BCPII provides a JWT along with the user's request which the Hardware Management Console (HMC) uses with a defined mapping to correlate z/OS users with HMC Users or Templates. The permissions of this HMC User or Template govern the authorization permitted to the z/OS user issuing the request.

For more information regarding JWTs, see [“Authorization” on page 381](#). See HMC online help for User Management for additional information on creating HMC Roles, Users, and Templates.

Note: HWIREST has the ability to use either FACILITY class profiles or JWTs for authority checking. See [“Authorization” on page 381](#) for more information about AUTHMODE configurations for HWIREST and when the AUTHMODE configurations apply.

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 RACF syntax is as follows:

```
RALTER FACILITY HWI.TARGET.netid.nau APPLDATA('uppercasecommunityname')
SETROPTS RACLIST(FACILITY) REFRESH
```

where *netid.nau* represents the 3 to 17 character SNA name of the CPC.

The APPLDATA field for the BCPii community name contains a 1– to 16–character alphanumeric field. Only uppercase letters and numbers are allowed. Because of restrictions with the security products on z/OS, the BCPii SNMP community name must not contain any lowercase characters.

This RACF example assigns 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 HWIO221.

Setting up JSON Web Tokens (JWTs) for authorization

BCPii supports the generation of signed JWTs on behalf of the user to be used by the Hardware Management Console (HMC) to map z/OS users to HMC Users or Templates. This capability is enabled only for use with HWIREST or HWIREST2 callable services.

JWTs are only available for use by BCPii on IBM z17 and above systems, and are necessary for HMC targeting, asynchronous notifications, and new URIs introduced by IBM z17 Web Services and above. See the "Hardware Management Console Web Services API" publication for more information.

Configuring signed JWTs for use by BCPii

1. Install Crypto Express card and assign to BCPii images:
 - a. Each IBM z17 CPC on which your BCPii images reside requires one Crypto Express card with a CCA coprocessor. It can be shared by the BCPii images that require JWT creation (does not need to be dedicated). See online help on the Support Element (SE) for information on how to install and configure the Crypto Express Card.
 - b. Assign the Crypto Express card to the image activation profile for each BCPii image in the 'Customize and Delete Activation Profile' panel on the SE. Shutdown each of the images, deactivate and re-activate them, then IPL each image.
 - c. ICSF is a software element of z/OS that works with hardware cryptographic features and Security Server (RACF) to provide secure, high-speed cryptographic services in the z/OS environment. ICSF provides application programming interfaces by which applications request the cryptographic services. As part of the ICSF installation, you will need to create a public key data set (PKDS) to store the RSA public and private keys required for BCPii JWT support. See z/OS Cryptographic

Services ICSF System Programmer's Guide for more information on how to install, initialize, and customize ICSF to use your Crypto Express card.

2. Create a BCPii authorization certificate and export:

- a. In order to generate a JWT (JSON Web Token) for BCPii to use, create a BCPii authorization certificate, using the RACDCERT GENCERT command. Key fields of the RACDCERT GENCERT command for this support are:

- SUBJECTSDN, WITHLABEL: name of the certificate which must be the same for both fields.
- RSA [(PKDS [(pkds-label | *)])] : must be RSA and PKDS. pkds-label is the key label under which RACF stores the signing key in the PKDS.
- SIZE(2048): must be 2048

If all systems that share the same RACF database are also in the same sysplex, they can share the same certificate. In this case, asterisk (*) should be used as the value for the pkds-label. When an asterisk is used, the certificate label from the WITHLABEL keyword will be used as the pkds-label.

If the systems are in different sysplexes but share the same RACF data through synchronization, a separate certificate per sysplex must be generated. The same pkds-label (not asterisk) must be used for all certificates generated.

For information on generating a BCPii authorization certificate, see the RACDCERT GENCERT section in the z/OS Security Server RACF Command Language Reference .

- b. After the certificate is created, it must be exported into a CERTB64 format so that it can be imported to the HMC. This can be accomplished with the RACDCERT EXPORT command.

Key fields of the RACDCERT EXPORT command for this support are:

- LABEL: must match value used in SUBJECTSDN and WITHLABEL
- FORMAT: CERTB64

See the RACDCERT EXPORT section in the z/OS Security Server RACF Command Language Reference for information on exporting a certificate to a dataset.

3. Create HMC User or Template on the HMC to which z/OS userids will be mapped:

- With this support, each z/OS user ID using BCPii will be mapped to an HMC User or Template that will define the permissions the z/OS user ID will have on the HMC/SE. Before the mapping can be done, the HMC Users or Templates must be created on the HMC. An HMC Role can be created to further customize access privileges.

See online help on the HMC for the 'User Management' panels for further information on creating HMC Roles, Users, and Templates.

4. Create a BCPii authorization record on the HMC and import the BCPii authorization certificate:

- a. After the BCPii authorization certificate is exported from the RACF database, the certificate will be imported to the HMC using FTP or file upload. If you choose FTP, prepare for the import by copying your certificate to a z/OS UNIX directory using the TSO OPUT command.
- b. Create a BCPii authorization record on HMC through the 'BCPii authorizations' section of the 'Customize Console Services' panel. The BCPii authorization record allows you to create one or more user associations. A user association represents a mapping between a z/OS user ID using BCPii, and a HMC User or Template. During this process, you will also be prompted to import the BCPii authorization certificate and link it to your user associations.

See online help on the HMC for information on how to create a BCPii authorization record and import the BCPii authorization certificate.

5. Enable WEB Services:

- a. On the HMC, in the 'Customize API Settings' panel, go to the WEB Services tab and select the **Enable** checkbox.
- b. Select the HMC Users and HMC User Templates that the z/OS user IDs are mapped to in the BCPii authorization record.

- c. Click **OK**.

See online help panels on the HMC for further information.

6. Configure IDTDATA profile in RACF:

- a. In RACF security, IDTDATA is a special class used to define profiles that control the generation and validation of Identity Tokens (IDTs), or JWTs. In order to define an IDTDATA profile for BCPii to use, you must first ensure the IDTDATA RACF class is activated. You may also wish to enable RACF to allow generic profiles for the IDTDATA class.

See the SETROPTS command in z/OS Security Server RACF Command Language Reference for further information.

- b. Using the RDEFINE command, create the IDTDATA class profile(s) in RACF. If you wish to create a generic profile, the profile name must be JWT.HWIBCPii.*.SAF . If you prefer, you may also create discrete profiles for specific z/OS user IDs (zosuserid) with the format JWT.HWIBCPii.<zosuserid>.SAF.

Note: If you do not create a generic IDTDATA profile, you must create a discrete profile for the user ID associated with the BCPii started task.

Key fields of the RDEFINE command for this support are:

- IDTPARMS:
 - SIGALG(RS512): must be RS512
 - SIGLABELPRIMARY: must match the pkds-label used to create the BCPii Authorization Certificate.
 - IDTIMEOUT: specifies the number of minutes that the Identity Token (IDT) associated with the profile is active.

See the RDEFINE command in the z/OS Security Server RACF Command Language Reference for information on creating the IDTDATA class profile(s).

- c. Activate your changes for the IDTDATA profile.

SETROPTS RACLIST(IDTDATA) REFRESH

BCPii configuration considerations

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*.SCEERUN2 and *high-level-qualifier*.SCEERUN 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. BCPii also requires the *high-level-qualifier*.SCEERUN2 and *high-level-qualifier*.SCEERUN data sets to be APF authorized. Failure to have these two data sets in the link list or APF authorized 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 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](#) for further information regarding CTRACE settings in BCPii.

BCPii writes SMF record 106 (X'6A') for certain API invocations. An SMFPRMxx parmlib member must be configured and activated in order to capture these records. See [“SMF recording in BCPii” on page 262](#) or [z/OS MVS System Management Facilities \(SMF\)](#) for more information about how BCPii uses SMF.

Considerations for Language Environment runtime options

z/OS BCPii uses z/OS Language Environment® to fulfill an API request from the caller. It creates an environment that usually does not conflict with an installation's default runtime option settings. However, specifying the NONOVR attribute to customize the Language Environment runtime options, either via the CEEPRMxx parmlib member or via the SETCEE command, can result in incompatible settings that can lead to incorrect behavior by BCPii, including an abend when BCPii initializes and attempts to open the SYSOUT data set.

When you specify the NONOVR attribute for a runtime option, that runtime option cannot be overridden later, including by a later specification in the same parmlib member or by a subsequent SETCEE command.

BCPii requires the following Language Environment runtime options and will attempt to override them:

```
TRAP(ON,NOSPIE)
POSIX(OFF)
ALL31(ON)
STACK(, ,ANY)
DYNDUMP(*USERID,NODYNAMIC,TDUMP)
MSGFILE(SYSOUT, , , ,ENQ)
TERMTHDACT(UADUMP, ,256)
```

Specifying the NONOVR attribute for any of these options can result in the following error message when BCPii starts and attempts to override the options that can no longer be overridden:

```
CEE3768I The system default for the run-time option option could not be overridden.
```

IBM recommends that you not specify the NONOVR attribute for these options in order to allow BCPii's SYSOUT initialization to complete successfully and, in general, to allow BCPii to behave properly.

Dynamic modification of CPC names

An installation that implements a dynamic CPC name change on a CPC, which either has BCPii active on one or more of its z/OS images or is a CPC that is targeted by other images on remote CPCs in the HMC network must review the following considerations before performing the name change.

BCPii provides support for changing the name of a CPC with ACTIVE images. When a CPC name change is detected for the local CPC, BCPii takes the following actions:

- Invalidates outstanding connections to the affected CPC.
- Issues an ENF 68 event to inform interested parties of the name change (hardware event HWIENF68_HWEVENT_NAMECHG).
- Reconnects to the local CPC (if the local CPC name is changed).

Applications that target the CPC using the old name receive a return code indicating that the connection is no longer valid (for instance, HWI_CONNECT_TOKEN_INV or HWI_TARGET_CPC_CHANGED). Applications that take advantage of the HWIREST interface will receive an HTTP Status 504 with Reason Code 1, implying the SE was unable to connect to the specified target name.

BCPii applications that have registered for communication errors associated with the local CPC may also receive a permanent communication error after the name change event (HWIENF68_HWCOMMERROR_PERM).

Users of BCPii should also be aware that the duration of time for a Support Element(SE) to complete processing a CPC name change can be very long. After a name change, applications might want to invoke the HWICONN service to connect to the CPC that has the new name. An application that attempts to communicate with a CPC that has just changed its name needs to handle this delay by not issuing any

further requests to that CPC and allowing the SE to complete initialization. Attempting to communicate prior to the SE completing its reboot can result in various BCPii communication errors.

It is necessary to review the security definitions for `HWI.TARGET.netid.nau`, `HWI.TARGET.netid.nau.image`, and `HWI.CAPREC.netid.nau.caprecname` profiles whenever a CPC name change is implemented to ensure that the proper security configuration is in effect. Review any profiles that might need to be modified before the name change takes place. Ensure any additions and changes to security profiles are made before the CPC name actually being changed, or security failures could immediately occur. See [“Setting up authority to use BCPii” on page 251](#) for general information on configuring the appropriate security definitions.

Setting up event notification for BCPii z/OS UNIX applications

Applications running in a started procedure, batch, TSO or other non z/OS UNIX 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 z/OS UNIX 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 z/OS UNIX applications to be able to receive such event notifications. BCPii provides several services that use the CEA functionality to deliver these same events to z/OS UNIX callers. See the documentation for the z/OS UNIX-only services of BCPii ([“HWIBeginEventDelivery — Begin delivery of BCPii event notifications” on page 446](#), [“HWIEndEventDelivery — End delivery of BCPii event notifications” on page 450](#), [“HWIManageEvents — Manage the list of BCPii events” on page 453](#), and [“HWIGetEvent — Retrieve outstanding BCPii event notifications” on page 458](#)) for details about the services a z/OS UNIX application can use to receive event notification.

The use of the CEA address space by BCPii requires some minor CEA setup before z/OS UNIX-only services of BCPii can work properly.

CEA address space setup

The Common Event Adapter (CEA) address space must be active to allow the z/OS UNIX-only services of BCPii to operate. CEA has two modes of operation: minimum or full-function mode. If the z/OS UNIX-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 z/OS UNIX 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 RACF example gives generic authority to the user id associated with a z/OS UNIX application authority to listen to any BCPii event:

```
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 RACF example allows user JOE authority to only receive events related to CPC command responses (CmdResp = '01'x):

```
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
```

Setting up access to BCPii REXX execs

This topic describes how to set up access to run BCPii REXX execs in the System REXX environment and in the TSO/E REXX environment.

Setting up access to the HWIREXX helper program

BCPii provides the HWIREXX helper application to allow easier execution of a BCPii REXX exec in the System REXX environment. (See [“Executing a BCPii REXX exec in the System REXX environment”](#) on page 265 for more information.)

In order to run a BCPii System REXX exec using the HWIREXX helper program, you must have at least READ authority to the HWI.HWIREXX.*execname* resource in the FACILITY class, where *execname* specifies a 1- to 8-character name of the System REXX exec that you want to be executed by the HWIREXX helper program. BCPii also requires that the FACILITY class be RACLIST-specified. The following example shows the RACF commands to accomplish this:

```
RDEFINE FACILITY HWI.HWIREXX.execname UACC(NONE)
PERMIT HWI.HWIREXX.execname CLASS(FACILITY) ID(userid) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Setting up an environment to run BCPii TSO/E REXX execs

In order to allow REXX execs running under TSO/E to use BCPii, BCPii must specifically be granted the authority to run under TSO/E and the proper host command environment must exist.

Setting up access for BCPii TSO/E REXX Execs

The TSO/E environment is an unauthorized program environment. BCPii normally requires its APIs to be invoked from a program-authorized application. An installation may choose to allow BCPii APIs to be run under TSO/E REXX by making a configuration update to the "TSO/E Commands and Programs" parmlib member (IKJTSOxx). The program HWIC1TRX must be added to the list of APF-authorized programs that may be called through the TSO Service Facility (AUTHTSF).

The following example shows the syntax required to add BCPii to this list:

```
AUTHTSF NAMES(HWIC1TRX)
```

To activate this change on a live system, issue the SET command: SET IKJTSO=*xx*; where *xx* is the two-character suffix of the IKJTSOxx parmlib member where the update was made.

Once this change is activated, the TSO/E user still requires SAF authorization to the correct BCPii profiles in order to successfully perform the desired BCPii operations.

Host command environment considerations

When setting up the environment for REXX execs using BCPii running under TSO/E or ISPF, be sure to check that you are not using a customized copy of the IBM-supplied host command environment modules, IRXTSPRM or IRXISPRM. If your installation is using its own copies of these modules, review

and update as appropriate to add the following statements to ensure that the BCPii host command environment can be found.

```
SUBCOMTB_NEXT_SYSCALL    DS 0C
SUBCOMTB_ENTRY_BCPPII    EQU *
SUBCOMTB_NAME_BCPPII     DC CL8'BCPII ' /* Name is BCPPII */
SUBCOMTB_ROUTINE_BCPPII   DC CL8'HWIM1RTI' /* Routine is HWIM1RTI */
SUBCOMTB_TOKEN_BCPPII     DC CL16' '
SUBCOMTB_NEXT_BCPPII      DS 0C
```

When adding these statements to the SUBCOMTB structure, increment the SUBCOMTB_TOTAL and SUBCOMTB_USED values in the SUBCOMTB_HEADER structure.

Note: Modules can be in LPALIB or any data set loaded as part of the LPA extension (LPALSTxx, fixed LPA (IEAFIXxx), MLPA (IEALPAXx), dynamic LPA via the SETPROG LPA,ADD command.)

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 246.
2. The community name of the local CPC is either not defined in the security product or contains an incorrect value. This is accompanied by message HWI022I (when the value defined in the security product is incorrect). See [“Community name defined in the security product for each CPC”](#) on page 254 for detailed information.

When these problems have been corrected, restart the BCPii address space. See [“Restarting the HWIBCPii address space”](#) on page 261 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 System Commands](#) 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, BCPii is now active and all BCPii requests may resume. However, all prior connections are no longer valid, and applications will need to re-establish these connections in order to resume their current BCPii activity. See [z/OS MVS System Commands](#) 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.

BCPii communication monitoring

BCPii monitors communication with the local CPC and any CPC that an application requests explicit interaction with, for example, an application issued an HWICONN or invoked HWIREST against a specific CPC. BCPii accomplishes this by keeping a heartbeat between itself and the targeted CPC's associated Support Element. While not common, BCPii may occasionally experience communication delays or interruptions of service. This can be due to a loss of a heartbeat or reception of a Console application ended event indicating the Support Element's console application is being shut down or restarted.

Starting with z16, the Support Element may precede the console application ended event with a shutdown event to prepare the application for the upcoming console shutdown or restart. BCPii classifies an interruption of services which was preceded by a shutdown event as an "expected" loss of communication versus "unexpected". In both scenarios, expected and unexpected loss of communication, BCPii will attempt to re-establish communication with the targeted CPC's associated Support Element.

In the case of an expected communication error, BCPii may wait up to 32 minutes, versus 2 minutes for an unexpected communication error, before attempting to re-establish communication. In between the communication attempts, BCPii continues to listen for a possible started event that indicates the Support Element has completed initialization. This process continues for 24 hours.

If 24 hours is exceeded and communication has not been re-established, BCPii considers this a permanent communication loss and will no longer attempt to re-establish communication with the CPC. Communication can be manually requested at any time by issuing an HWICONN or invoking HWIREST against a specific CPC.

As BCPii detects communication loss and attempts to re-establish communication, it will issue one or more of the following messages:

```
HWI024I - BCPii IS NO LONGER ATTEMPTING COMMUNICATION WITH A CPC
HWI025I - BCPii HAS ACTIVE COMMUNICATION WITH A CPC
HWI026I - BCPii IS EXPECTING LOSS OF COMMUNICATION WITH A CPC
HWI027I - BCPii HAS EXPERIENCED EXPECTED COMMUNICATION LOSS WITH A CPC
HWI028I - BCPii HAS EXPERIENCED UNEXPECTED COMMUNICATION LOSS WITH A CPC
HWI029I - BCPii IS ATTEMPTING TO REGAIN COMMUNICATION WITH A CPC
```

Detailed message descriptions can be found in [z/OS System Messages Volume 6](#) in the section listing [HWI messages](#).

SMF recording in BCPii

BCPii automatically writes SMF records for all HWISET / HWISET2, and HWICMD / HWICMD2 requests that complete with a return code of zero and HWIREST requests that modify a resource or perform a command like operation that return with successful HTTP status.

In order for SMF to keep these records, it is necessary to activate recording of SMF record type 106 (X'6A') on the system. You can do this in either of the following ways:

- Create or modify an SMFPRMxx parmlib member and specify SYS(TYPE(106)) to capture all type 106 records, or SYS(TYPE(106(1))) to only capture those records written by HWISET / HWISET2 / HWIREST that modify or SYS(TYPE(106(2))) to only capture those records written by HWICMD / HWICMD2 / HWIREST command like operations. Then, activate the parmlib member by issuing the **SET SMF=xxx** command.
- Dynamically add recording of type 106 records by issuing the **SETSMF** command.

Once activated, any successful, accepted by the SE, operation that was specified in the SMFPRMxx parmlib member will be recorded in SMF.

When you want to review the records, you must dump them using either the SMF data set dump program (IFASMFDP) or the SMF logstream dump program (IFASMF DL), depending on whether your installation uses data set recording or logstream recording for SMF records. You can find a sample invocation of each of these programs in the first step of the IBM-supplied HWI6AFMT job in SYS1.SAMPLIB.

After dumping the records, you can analyze them using either of the following methods:

- **Use the IBM-supplied HWI6AFMT sample reporting job**

IBM provides a sample report using the DFSORT ICETOOL. By using simple JCL, you can create reports of the BCPii SMF data. The HWI6AFMT job requires the companion HWIRPTMP sample job to make the ICETOOL program aware of structure and formatting for the BCPii SMF data. You can copy and customize the HWI6AFMT job to report on the desired data.

- **Sort and format the records manually**

You might choose to write your own program to report on the type 106 records. The records are mapped by the HWISMF6A mapping macro found in SYS1.MACLIB.

For more information, see [Record type 106 \(X'6A'\) – BCPii activity in z/OS MVS System Management Facilities \(SMF\)](#).

BCPii callable services

You can use base control program internal interface (BCPii) services to connect an authorized z/OS application to z System 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 (BCPii) 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 topic 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 topic contains the following subtopics:

- [“Syntax, linkage and programming considerations” on page 263](#)
- [“HWICMD / HWICMD2 – Issue a BCPii hardware management command” on page 274](#)
- [“HWICONN – Establish a BCPii connection” on page 298](#)
- [“HWIDISC – Release a BCPii connection” on page 311](#)
- [“HWIEVENT – Register or unregister for BCPii events” on page 318](#)
- [“HWILIST – Retrieve HMC and BCPii configuration-related information” on page 332](#)

- [“HWIQUERY — BCPii retrieval of SE/HMC-managed attributes” on page 347](#)
- [“HWIREST — Issue RESTlike requests to the SE” on page 378](#)
- [“HWISET/HWISSET2 — BCPii set single or multiple SE/HMC-managed attributes” on page 399](#)
- [“HWIBeginEventDelivery — Begin delivery of BCPii event notifications” on page 446](#)
- [“HWIEndEventDelivery — End delivery of BCPii event notifications” on page 450](#)
- [“HWIManageEvents — Manage the list of BCPii events” on page 453](#)
- [“HWIGetEvent — Retrieve outstanding BCPii event notifications” on page 458](#)

Syntax, linkage and programming considerations

Programming language 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 REXX (HWICIREX) in data set SYS1.MACLIB. Miscellaneous REXX constants are defined in HWIC2REX in the same data set.

Note:

1. If the REXX exec is running under System REXX using the TSO=YES environment, these include files may be read in at the time of execution by the REXX exec. A simple programming example that reads the values into the REXX exec through the use of the EXECIO function is provided in the IBM-supplied REXX samples. See [“Programming Examples” on page 274](#) for further information.
 2. If the REXX exec is running under System REXX using the TSO=NO environment, the definitions in these include files may be copied into the REXX exec.
- In assembler (HWICIASM) in data set SYS1.MACLIB. Miscellaneous assembler 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

```
BCPii_service_name (return_code,param1,param2, ...)
HWIREST(param1,param2)
```

REXX

```
ADDRESS BCPii “BCPii_service_name return_code parm1 parm2 ...”
ADDRESS BCPii “HWIREST parm1 parm2”
```

Assembler Call macro

```
CALL BCPii_service_name,(return_code,param1,param2, ...),VLIST
CALL HWIREST,(param1,param2),VLIST
```

BCPii connection scope

With the exception of HWIREST, all BCPii services have a concept of a connection. BCPii limits access to active BCPii connections. BCPii will not allow a program to use a previously established BCPii connection unless it is running in the proper environment. BCPii associates a connection with either an address space or a task, depending on the execution environment of the connector. It then uses this association (affinity) to determine if the connection is allowed to be used on subsequent requests.

Connections with address space affinity

The BCPii connections created by a C program, an assembler program, or a System REXX exec are associated with an address space.

- For C and assembler programs, BCPii creates an affinity between the connection and the address space that initiated the connection (via the HWICONN service).
- For a System REXX exec, BCPii creates an affinity between the connection and the address space that initiated the execution of the REXX exec (via the AXREXX authorized service call).

BCPii allows any task running in the same address space to use these connections on subsequent BCPii API calls. In addition, the connection remains active until the address space terminates.

Connections with task affinity

The BCPii connections created by a REXX exec running in either a TSO/E or ISV-provided REXX environment are associated with the task that initiated the execution of the REXX exec.

BCPii only allows the task that initiated the connection (via the HWICONN service) to access this connection on subsequent BCPii API calls. In addition, the connection only remains active until the task terminates.

Linkage considerations

There are two ways for a compiled BCPii application (non-REXX) to find BCPii callable services:

- Use the linkable stub routine HWICSS from SYS1.CSSLIB to link-edit your object code.
- Use the LOAD macro to find the address of the BCPii callable service at run time and then CALL the service.

REXX programming considerations

BCPii supports REXX execs being executed from the System REXX, TSO/E REXX, and independent software vendor (ISV) REXX programming environments. Each REXX environment is unique:

- System REXX supports all BCPii APIs and provides the capability to write sophisticated BCPii applications by utilizing REXX and other programming languages as part of a single application.

Note:

- To use the HWIEVENT and HWICMD / HWICMD2 services, a non-REXX adjunct helper program is needed to call z/OS system services to prepare for events and to coordinate with an event exit. See [“Programming Examples” on page 274](#) for detailed information.
- The System REXX "MODIFY AXR" command is supported only for HWIREST. For all other APIs, see [“Executing a BCPii REXX exec in the System REXX environment” on page 265](#).
- TSO/E REXX execs are easy to execute from a TSO user. This environment supports all the BCPii APIs, except HWIEVENT and HWICMD / HWICMD2.
- ISV-provided REXX environments provide different features, depending on which ISV product is being used. These environments support all the BCPii APIs, except HWIEVENT and HWICMD / HWICMD2.

The following table identifies the z/OS BCPii APIs supported in the three REXX environments:

<i>Table 57. BCPii APIs supported in the REXX environment</i>			
BCPii APIs	System REXX environment	TSO/E REXX environment	ISV-provided REXX environment
HWICONN	X	X	X
HWIDISC	X	X	X
HWILIST	X	X	X
HWIQUERY	X	X	X
HWISET / HWISET2	X	X	X
HWIEVENT	X		

<i>Table 57. BCPII APIs supported in the REXX environment (continued)</i>			
BCPII APIs	System REXX environment	TSO/E REXX environment	ISV-provided REXX environment
HWICMD / HWICMD2	X		
HWIREST	X	X	X

The syntax of the BCPII REXX execs are identical in all three REXX environments. Therefore, a BCPII REXX exec written to be used in one REXX environment can be run in another REXX environment without change.

Executing a BCPII REXX exec in the System REXX environment

BCPII supports the invocation of its APIs from the System REXX programming environment. Execs running in this environment are APF-authorized. A user may choose either of the following methods to have their exec run under System REXX:

- Invoke the authorized HWIREXX helper program for basic requests.
- Use the AXREXX macro from an authorized program for more customized requests.

The dataset where the REXX exec is to be run must be specified using the REXXLIB keyword in the AXRxx parmlib member, and users of this program must have the proper authority to run programs residing in LINKLIB.

BCPII REXX programming restrictions for the System REXX environment

Only the HWIREST API is supported for invocation from a REXX exec which has been started via the MODIFY AXR command. Any attempt to run any of the other BCPII API's from this environment results in a return code of HWI_REXXInvalidExecutionEnv.

Using the HWIREXX interface

For basic REXX execs, BCPII API calls can be run easily from the System REXX programming environment using the supplied HWIREXX helper program, without the need to code an assembler program with an AXREXX macro invocation. IBM provides sample invocation JCL for HWIREXX in SAMPLIB member HWIXMRJL.

The HWIREXX interface provides some of the most common AXREXX macro keywords as input parameters. [Table 58 on page 265](#) lists the supported keywords.

<i>Table 58. HWIREXX keywords</i>			
HWIREXX keyword	Required/Optional	Default value	AXREXX macro parameter equivalent
NAME=xxx; where xxx is a 1-8 character exec name to be executed.	Required	N/A	NAME
DSN=xxx.xxx.xxx; where xxx.xxx.xxx is a 1-44 character PDS data set name where the REXX exec output is directed. Note: The data set may be pre-allocated prior to execution of the exec. If the data set is not pre-allocated, the data set is allocated by System REXX. In either case, the output from the REXX exec is contained in a member name within the data set that matches the specified HWIREXX NAME.	Optional	NO_REXXOUTDSN	REXXOUTDSN

Table 58. HWIREXX keywords (continued)			
HWIREXX keyword	Required/ Optional	Default value	AXREXX macro parameter equivalent
TSO=<Y/N>; where 'Y' means to run in the TSO host command environment, and 'N' means to run in the standard MVS host environment.	Optional	N	TSO
SYNC=<Y/N>; where 'Y' means the request is synchronous, and 'N' means the request is asynchronous.	Optional	Y	SYNC
TIMELIM=<Y/N>; where 'Y' means that a time limit is applied, and 'N' means that no time limit is applied.	Optional	Y	TIMELIMIT
TIME=xxx; where xxx is a number value between 1 and 21474536 that represents the number of seconds to allow the exec to run.	Optional	System default value	TIMEINT

See the JCL example HWIXMRJL shipped in SAMPLIB for more information on the invocation of the HWIREXX helper program.

If additional AXREXX macro parameters are required (other than the AXREXX macro parameters listed in Table 58 on page 265) to properly establish the System REXX environment, an explicit invocation of the AXREXX macro is required. See [“Using the AXREXX macro” on page 268](#) for detailed information.

Return codes from the HWIREXX service

Table 59. Return codes from the HWIREXX service	
HWIREXX return code (in decimal)	Meaning and action
0	<p>Meaning: BCPii processed the REXX host command successfully.</p> <p>Action: Consult the BCPii return code on the BCPii service call to determine the final result of the request.</p>
100	<p>Meaning: Program error. Caller's JCL string has a syntax error.</p> <p>Action: Check for a probable coding error and correct the problem. See “Using the HWIREXX interface” on page 265 for detailed information.</p>
101	<p>Meaning: Program error. A required parameter is not found.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
102	<p>Meaning: Program error. No input parameters were specified.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
103	<p>Meaning: Program error. A parameter keyword was provided that is not supported by HWIREXX.</p> <p>Action: Check for a probable coding error and correct the problem. HWIREXX supports these keywords only: NAME, DSN, TSO, SYNC, TIMELIM, and TIME (which correspond to the AXREXX macro parameters: NAME, REXXOUTDSN, TSO, SYNC, TIMELIMIT, and TIMEINT, respectively.)</p>

Table 59. Return codes from the HWIREXX service (continued)

HWIREXX return code (in decimal)	Meaning and action
104	<p>Meaning: Program error. Duplicate parameter keys are specified.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
105	<p>Meaning: Program error. A keyword may only consist of alphanumeric characters.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
106	<p>Meaning: Program error. Parameter values may only consist of alphanumeric characters and periods (.).</p> <p>Action: Check for a probable coding error and correct the problem.</p>
107	<p>Meaning: Program error. The TSO parameter must be Y or N.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
108	<p>Meaning: Program error. The SYNC parameter must be Y or N.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
109	<p>Meaning: Program error. The TIMELIM parameter must be Y or N.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
110	<p>Meaning: Program error. A parameter value is too long. Name values are limited to 8 characters; data set names are limited to forty-four (44) characters; the TSO value is one character; the SYNC value is one character; the TIMELIM value is one character; and the TIME value is limited to 8 characters.</p> <p>Action: Check for a probable coding error. Reduce the length to the appropriate size based on the specified parameter.</p>
111	<p>Meaning: Program error. Blank character is not allowed in the JCL string.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
112	<p>Meaning: Setup error. The caller does not have the correct SAF authorization to run the HWIREXX program.</p> <p>Action: The security administrator needs to give the user at least READ authority to the HWI.HWIREXX.execname resource in the FACILITY class. See “Setting up access to the HWIREXX helper program” on page 259 for more information.</p>
2049 - 4111	<p>Meaning: Reason code returned from AXREXX.</p> <p>Action: See the AXREXX macro in <i>z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN</i>.</p>
4095	<p>Meaning: System error. An unexpected error is detected. The system rejects the service call.</p> <p>Action: Search the problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Using the AXREXX macro

If HWIREXX does not provide the options for your REXX exec requires, you can run your REXX exec using the AXREXX macro from the System REXX programming environment.

For example, an assembler program running in supervisor state, PKM 0-7, or APF-authorized can invoke the AXREXX macro to execute a REXX exec as follows:

```
AXREXX      REQUEST=EXECUTE,
            NAME=execname,      <--- 8-character name of REXX exec
            TSO=NO,             <--- Runs in a standard MVS host command environment
            REXXARGS=rexxargs,  <--- Input/output parameters mapped by AXRARGLST
            REXXOUTDSN=outdsn,  <--- Specify output data set
            REXXOUTMEMNAME=memname, <--- Specify output member name
            RETCODE=retcode,     <--- R15 as a result of REXX exec
            RSNCODE=rsncode,     <--- R10 as a result of REXX exec
            TIMELIMIT=[YES,NO],  <--- Do you want the REXX exec to timeout?
            TIMEINT=numofsecs    <--- If TIMELIMIT=YES, how much time to wait?
```

After the invocation of the AXREXX macro, the REXX exec gets control and the input parameters are passed to the REXX exec. If any output is generated from the exec, it is directed to the specified output data set and member name. Lastly, the return code and reason code are returned.

For a complete description of the AXREXX macro and its usage, see *z/OS MVS Programming: Authorized Assembler Services Guide* and *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*. For a BCPII example showing the invocation of the AXREXX macro, see SAMPLIB member HWIXMRA1.

Executing a BCPII REXX exec in the TSO/E REXX environment

BCPII supports the invocation of its APIs from the TSO/E REXX programming environment, as long as the installation has allowed BCPII to be available from the TSO/E environment. See [“Setting up an environment to run BCPII TSO/E REXX execs”](#) on page 259 for information on setting up BCPII to run in a TSO/E REXX environment.

BCPII APIs can be run from REXX execs under TSO/E in the following ways:

- TSO/E foreground:
 - Issue the exec from the TSO/E READY mode, or
 - ISPF by using the TSO EXECUTE command.
- See [z/OS TSO/E REXX User's Guide](#) for the syntax of the EXECUTE command.
- TSO/E background:
 - Issue the exec from JCL, specifying IKJEFT01 as the program name on the JCL EXEC statement. See [z/OS TSO/E REXX Reference](#) for more information about running REXX execs using IKJEFT01.

BCPII REXX programming restrictions for the TSO/E environment

The following are not supported in BCPII REXX execs running in the TSO/E environment:

- HWICMD / HWICMD2
- HWIEVENT
- HWI_LIST_EVENTS for the BCPII HWILIST service

Executing a BCPII REXX exec in an ISV-provided REXX environment

BCPII supports the invocation of its APIs from ISV-provided REXX programming environments, provided that the REXX execs running in this environment are program-authorized.

Because BCPII support is not native to ISV-provided REXX environments, the BCPII host command environment must first be enabled. To accomplish this, the BCPII REXX exec must first invoke the BCPII-provided *hwi/host* function to enable the BCPII host command environment prior to any BCPII API invocation using "address bcpii".

Note: It is also recommended (but not required) that you invoke the *hwihost* function to disable the BCPii host environment when it is no longer needed by the BCPii REXX exec.

To enable the BCPii host command environment, add the following statement to your BCPii REXX exec:

```
RC = hwihost("ON")
```

To disable the BCPii host command environment, add the following statement to your BCPii REXX exec:

```
RC = hwihost("OFF")
```

Invocations of the *hwihost* function in an exec running in either the System REXX or TSO/E REXX programming environments are ignored, and the resulting return code is always zero. This ensures compatibility of REXX execs running in any REXX programming environment on z/OS.

BCPii REXX programming restrictions for an ISV-provided REXX environment

The following are not supported in BCPii REXX execs running in an ISV-provided REXX environment:

- HWICMD / HWICMD2
- HWIEVENT
- HWI_LIST_EVENTS for the BCPii HWILIST service

REXX Programming tips

When programming a BCPii application using REXX, see the specific REXX programming considerations for each individual BCPii callable service for all necessary interface distinctions. Users of the BCPii REXX interface should be aware of the following:

- All parameters passed on BCPii REXX service calls must be REXX variables. Literals are not supported (for example, a variable name which has been assigned the value of a ListType should be specified on the call instead of the value itself).
- Variable names specified on BCPii REXX service calls are limited to 40 characters in length.
- Output variables specified on BCPii REXX service calls may be initialized or un-initialized. On input, the value of output variables are not verified. Output variables are initialized and set by BCPii.
- If the value of an input variable is incompatible with the parameter type required on a particular BCPii REXX service call, an error is flagged. See the REXX programming considerations for each BCPii callable service for the specific interface distinctions.
- The DiagArea call for each BCPii REXX service, excluding HWIREST, is returned using stem variables in the form: x.Diag_Index, x.Diag_Key, x.Diag_Actual, x.Diag_Expected, x.Diag_CommErr and x.Diag_Text (where x is the name of the stem variable specified on the parameter list). If no DiagArea information is filled in by BCPii, the value of the DiagArea stem-variable on return is all blanks.
- Stem variables utilized by BCPii have hard-coded stem variable tail values which usually correspond to the documented parameter name. For example, the QueryParm. stem must be prepared in REXX with the exact stem variable "ATTRIBUTEIDENTIFIER".
- The stem and compound variables utilized by BCPii have hard-coded variable tail names. Per REXX rules, compound symbols permit the substitution of variables within its name when they are referenced. To prevent unexpected changes to the stem variables passed into BCPii APIs, IBM does not recommend REXX applications use variable names that are the same as stem tail names documented by BCPii. The re-use of the variable names may result in BCPii service failures caused by missing or incorrect parameters.

See TSO/E REXX Reference for more information about how Stem and Compound variables are used:

- [Compound Symbols](#) in *z/OS TSO/E REXX Reference*
- [Stems](#) in *z/OS TSO/E REXX Reference*

- The ConnectToken parameter returned on the HWICONN call and passed as input on all subsequent services, excluding HWIREST, contains non-displayable characters. Ensure that this ConnectToken is untouched by the REXX exec, thereby allowing subsequent BCPii services to read the value correctly.
- For System REXX execs only: Consider the length of time necessary to run your BCPii REXX exec. BCPii applications are interacting with the CPC's support element. Therefore, BCPii REXX execs may take longer to run than other REXX execs. To avoid having your BCPii REXX application end prematurely, even when the amount of time calculated is reasonable to complete your BCPii REXX exec, consider using the TIMELIMIT and TIMEINT keywords on the AXREXX service call. The default TIMELIMIT=YES, TIMEINT=SYSTEM causes the REXX exec to stop running after a predetermined amount of time. The TIMEINT value may be increased to give the REXX exec additional time to complete its execution before being timed out by the system. In certain circumstances, it may be necessary to specify TIMELIMIT=NO to prevent the REXX exec from timing out. This option should be used with caution as System REXX has a finite number of system-wide regions where the System REXX execs are executed. If TIMELIMIT=NO is specified unnecessarily, this could eventually lead to a constrained System REXX environment.
- BCPii connections created under System REXX can be used by any program running in the address space of the connector (Address space affinity). BCPii connections created under the TSO/E or ISV-provided REXX environments can only be used by the same task as the connector (Task affinity). See “BCPii connection scope” on page 263 for detailed information.
- BCPii requires all callers to be program-authorized. REXX execs in the zFS cannot run as APF-authorized when invoked from the shell. Therefore, any calls to BCPii services from REXX execs in this environment will result in a HWI_AUTH_FAILURE return code or in the case of HWIREST, the services will return HTTP Status 403 and the appropriate reason code.
- The built-in REXX RC variable contains the return code from the REXX BCPii host command. This return code indicates BCPii's acceptance of the supplied REXX BCPii host command. The return codes returned in the RC variable are generally unique to the REXX environment. In contrast, the BCPii service return code, the variable supplied on the service call itself, excluding HWIREST, is only filled in if the RC variable has a value of HWI_OK (0) or HWI_REXXParmSyntaxError (1). Possible return codes returned by BCPii in the RC variable are:

Return codes from a REXX BCPii host command

Table 60. Return codes from a REXX BCPii host command	
REXX RC returned from a BCPii host command (in decimal)	Meaning and action
0 HWI_OK	<p>Meaning: BCPii processed the REXX host command successfully.</p> <p>Action: Consult the BCPii return code on the BCPii service call to determine the final result of the request.</p>
1 HWI_REXXParmSyntaxError	<p>Meaning: Program error. The REXX BCPii host command has detected that the format of the parameters is not in the proper form to be accepted by BCPii.</p> <p>Action: Check for a probable coding error. See the BCPii return code on the BCPii service call to determine the reason for the syntax error. See the REXX programming considerations of the BCPii service to see the exact calling specifications. Compare the BCPii REXX service call attempted with service call examples in the supplied BCPii REXX programming sample found in SYS1.SAMPLIB. See the DiagArea for further diagnostic information.</p>
2 HWI_REXXUnsupportedService	<p>Meaning: Program error. An unknown BCPii service name was specified on the BCPii REXX host command.</p> <p>Action: Check for a probable coding error. Specify a valid BCPii service name (for example, HWICONN, HWILIST, and so on).</p>

Table 60. Return codes from a REXX BCPii host command (continued)	
REXX RC returned from a BCPii host command (in decimal)	Meaning and action
3 HWI_REXXInvalidNumofParms	<p>Meaning: Program error. The number of parameters specified on the BCPii REXX host command for the service name specified does not match the number of parameters expected.</p> <p>Action: Check for a probable coding error. See the REXX programming considerations of the BCPii service to see the exact calling specifications. Compare the BCPii REXX service call attempted with service call examples found in the supplied BCPii REXX programming sample found in SYS1.SAMPLIB.</p>
4 HWI_REXXStemVarRequired	<p>Meaning: Program error. The BCPii REXX service specified on the BCPii REXX host command is missing one or more required stem variables in the positional parameter list.</p> <p>Action: Check for a probable coding error. See the REXX programming considerations of the BCPii service to see the exact calling specifications. A stem variable parameter must specify a “.” following the variable name (for example, “var.”). Also, compare the BCPii REXX service call attempted with service call examples found in the supplied BCPii REXX programming sample found in SYS1.SAMPLIB.</p>
5 HWI_REXXParmNameTooLong	<p>Meaning: Program error. One or more variables specified on the BCPii REXX service call on the BCPii REXX host command is greater than the BCPii maximum REXX variable length (40).</p> <p>Action: Check for a probable coding error. Reduce the variable name lengths on the BCPii REXX service call to be 40 characters or less in length.</p>
6 HWI_REXXInvalidHostEnv	<p>Meaning: System error. BCPii detected an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
7 HWI_REXXInvokerNotFound	<p>Meaning: Program error. The address space issuing the AXREXX invocation is no longer running. No new BCPii connections are allowed.</p> <p>Action: Determine the reason that the AXREXX-invoking address space terminated prior to the termination of the REXX exec. Correct the situation and start again.</p>
8 HWI_REXXInvalidExecutionEnv	<p>Meaning: Program error. BCPii does not support the BCPii host command running in the current execution environment.</p> <p>If the current execution environment is System REXX, it may mean that an attempt was made to issue a BCPii host command from an exec that was started using the MODIFY AXR command.</p> <p>If the current execution environment is either TSO/E or ISV-provided REXX, it may mean that the requested service was not supported in this environment.</p> <p>Action: Run the BCPii host command from a supported environment.</p>

Table 60. Return codes from a REXX BCPii host command (continued)

REXX RC returned from a BCPii host command (in decimal)	Meaning and action
9 HWI_REXXUnsupportedListType	<p>Meaning: Program error. BCPii does not support the specified ListType on the BCPii HWILIST service in the current execution environment.</p> <p>Action: Correct the specified ListType value or try this request again in a valid execution environment (for example, the System REXX environment).</p>
10 HWI_hwihost_MissingRequiredParm	<p>Meaning: Program Error. The HWIREST request parameter is missing one or both of the required stem variables: HTTPMethod or URI</p> <p>Action: Ensure the HWIREST request parameter includes HTTP Method and URI stem variables. For more information, please refer to BCPii's guides on submitting requests through the HWIREST service.</p>
11 HWI_hwihost_InvalidParmValue	<p>Meaning: Program Error. The value of a request parameter is not in a valid format. For example, the variable value is a string instead of an integer.</p> <p>Action: Ensure the values passed in are of the appropriate type.</p>
12 HWI_hwihost_ParmValTooLong	<p>Meaning: Program Error. The length of a provided value exceeds the maximum length supported for that request parameter stem variable. For example, the supplied URI had a length greater than the supported 2048 characters.</p> <p>Action: Ensure the values passed do not exceed the maximum length. For more information, please refer to BCPii's guides on submitting requests through the HWIREST service.</p>
32 HWI_REXXInternalSystemError	<p>Meaning: System error. BCPii detected an unexpected error while invoking REXX services. The system rejects the service call.</p> <p>Action: A symptom record has been written to LOGREC to record the problem. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
4095 HWI_Unexpected_Error	<p>Meaning: System error. BCPii detected an unexpected error. The system rejects the service call.</p> <p>Action: A symptom record has been written to LOGREC to record the problem. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

REXX return codes from the BCPii *hwihost* function

Table 61. REXX return codes from the BCPii *hwihost* function. The following return codes apply only to callers running their BCPii REXX execs in an ISV-provided REXX environment.

REXX RC returned by the BCPii <i>hwihost</i> function	Meaning and action
1 HWI_hwihost_ParmSyntaxError	Meaning: Program Error. The specified argument is not "ON" or "OFF". Action: Check for a probable coding error. Try this request again with an argument of "ON" or "OFF".
2 HWI_hwihost_InternalSystemError	Meaning: System error. BCPii detected an unexpected error while invoking TSO/E REXX services. The system rejects the service call. Action: A symptom record has been written to LOGREC to record the problem. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Sample REXX exec

Here is a sample REXX exec using BCPii calls that lists the names of all of the interconnected CPCs and then attempts to connect to each one of them:

```
/* REXX */
ListType = HWI_LIST_CPCS;
Address BCPii "HWIList Retcode ConnectToken ListType AnswerArea.
             DiagArea."

If RC = 0 & retcode = 0 Then
  Do
    ConnectType = HWI_CPC
    Do i = 1 To AnswerArea.0
      Say "CPC" i ":" AnswerArea.i

      InConnectToken = 0
      Address BCPii "HWICONN Retcode InConnectToken OutConnectToken
                  ConnectType AnswerArea.i DiagArea."
      If RC = 0 & retcode = 0 Then
        Say "Connected to CPC "AnswerArea.i"."
    End
  End
```

For REXX execs running in an ISV-provided environment, make sure to add the following line prior to the first address BCPii statement:

```
RC = hwihost("ON")
```

Assembler programming 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).

Programming Examples

BCPii provides sample programs to aid in the creation of BCPii applications in both C and REXX programming languages. The samples are shipped in SYS1.SAMPLIB. For HWIREST samples, see [IBM / zOS-BCPii \(github.com/IBM/zOS-BCPii\)](https://github.com/IBM/zOS-BCPii).

HWIXMCS1 (Metal C programming language) provides an example of how to use all of the BCPii APIs and how to construct a simple BCPii application. HWIXMCX1 (Metal C programming language) provides a simple example of how a BCPii Event Notification Facility (ENF) exit could be coded to field various BCPii-registered events.

HWIXMRS1 (REXX programming language) provides an example of how to use the most common BCPii APIs. It can easily be invoked in the System REXX environment by utilizing the IBM-provided HWIREXX program using the provided sample JCL HWIXMRJL.

Another REXX sample (HWIXMRS2) is provided to show how a REXX application can utilize the HWIEVENT and HWICMD APIs. It is invoked using an AXREXX macro invocation in the sample assembler "helper" program (HWIXMRA1). This second sample can utilize the Metal C ENF exit HWIXMCX1.

HWICMD / HWICMD2 — Issue a BCPii hardware management command

Call the HWICMD / HWICMD2 service to perform a command against an HMC-managed object that is associated with central processor complexes (CPCs) and CPC images (LPARs). User-defined image groups can also be used to target multiple images with a single command.

BCPii commands, because of the very nature of what they are attempting to do, can take a significant amount of time to complete. To prevent applications from being tied up for an excessive amount of time while waiting for the command to complete, HWICMD / HWICMD2 returns to the caller either when the command has been *accepted* by the target support element (SE) or when the command was found to contain errors. The actual completion of the command can be determined by consulting the final return code returned in the BCPii command response event.

To receive this BCPii command response event, an application must have registered for the Hwi_Event_CmdResp event before the HWICMD / HWICMD2 invocation. Registration for this or any event is accomplished by calling the HWIEVENT service, or for z/OS UNIX callers, by calling HwiManageEvents. The HWIEVENT service requires a user-supplied Event Notification Facility (ENF) exit.

When the command completes, BCPii signals the ENF to notify registered applications that a command response has been received. For non-z/OS UNIX callers, the ENF exit specified receives control and the command response event returned data contains the final return code of the request. For z/OS UNIX callers, the HwiGetEvent service can be used to receive the event notification and to determine the final return code of the HWICMD / HWICMD2 service.

BCPii provides two command services: HWICMD and HWICMD2. These services provide identical functions, but differ in the handling of the input command parameters. HWICMD2 allows for the specification of different versions of parameter lists to the service while HWICMD does not. See the rest of this section for more specifics on how the input parameter lists differ.

Description

Environment

The requirements for the callers are:

Requirement	Details
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
Console setup:	The main console on the HMC must be activated in order for the operating system commands to be sent successfully. To activate the main console, use the vary command: v cn(*),activate.

Programming requirements

See [“Syntax, linkage and programming considerations”](#) on page 263 for details about how to call BCPii services in the various programming languages.

The microcode level that supports the command service call 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 298 for more information.

See [“HWICMD / HWICMD2”](#) on page 821 for the summary table of the BCPii command types and the objects that can be targeted for each command.

REXX programming considerations for the HWICMD / HWICMD2 service

All information for the HWICMD / HWICMD2 service applies for REXX requests except:

- A stem variable (for example, CmdParm.) replaces CmdParm_ptr.
- The CmdParm structure names in [Table 64 on page 280](#) are used as the dot-qualified names in the CmdParm stem variable. The following are exceptions:
 - On the HWI_CMD_POWER_CONTROL, HWI_CMD_TEMPCAP, and HWI_CMD_SYSPLEX_TIME_SET_STP_CONFIG commands, XML replaces XML_ptr and XML_Size is ignored.
 - On the HWI_CMD_SYSRESET_IPLT command, IPL-Token replaces IPL-Token_Ptr and IPL-Token_Len is ignored.
- REXX allows HWICMD or HWICMD2 to be invoked. However the CmdParmVersion cannot be specified if HWICMD2 is used.
- REXX allows 4 or 5 digit load addresses to be specified.

Restrictions

- BCPii does not allow any command to be targeted to a CPC that is earlier than a z9 platform.
- BCPii does not allow command to be issued from within a BCPii ENF exit routine.
- BCPii does not allow any command to be issued from a REXX exec running in a TSO/E or ISV-provided REXX environment.

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 a ConnectToken that represents a CPC connection or an image group connection.
- HWI.TARGET.*netid.nau.image*name for a ConnectToken that represents an image connection.
- HWI.TARGET.*netid.nau.image*name for all individual images within the image group for a ConnectToken that represents a user-defined image group.

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

SMF recording

Requests that complete with a return code of zero will have SMF type 106 (X'6A') records written if the installation has activated recording of this record type in its active configuration.

Syntax

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

Table 62. HWICMD syntax	
Non-REXX parameters	REXX parameters
<pre>CALL HWICMD(ReturnCode, ConnectToken, CmdType, CmdParm_Ptr, DiagArea);</pre>	<pre>address bcpii "hwicmd ReturnCode ConnectToken CmdType CmdParm. DiagArea."</pre>

Table 63. HWICMD2 syntax	
Non-REXX parameters	REXX parameters
<pre>CALL HWICMD2(ReturnCode, ConnectToken, CmdType, CmdParm_Ptr, CmdParmVersion, DiagArea);</pre>	<pre>address bcpii "hwicmd2 ReturnCode ConnectToken CmdType CmdParm. DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

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.

A ConnectToken representing a user-defined image group may also be specified. In this case, the command will be executed on all members in the group, and not just on a single image.

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 (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)


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:

- *IBM z SNMP Application Programming Interfaces* (SB10-7171-06)
- *System z10 and eServer zSeries Application Programming Interfaces* (SB10-7030-09)
- *System z9 and eServer zSeries Application Programming Interfaces* (SB10-7030-08)
- *zEnterprise System Support Element Operations Guide* (SC28-6896-02)
- *System z10 Support Element Operations Guide* (SC28-6858-02)
- *System z9 Support Element Operations Guide* (SC28-6858-01)

Constant in Hexadecimal (Decimal) Equate Symbol	Description
1 (1) HWI_CMD_ACTIVATE	Activate request to start target systems with the default activation profile name (HWI_APROF) associated with a CPC or an image. Note: The input connection token represents a <i>CPC connection</i> , an <i>image connection</i> , or an <i>image group connection</i> . 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 connection token represents a <i>CPC connection</i> , an <i>image connection</i> , or an <i>image group connection</i> . 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 connection token must only represent a <i>CPC connection</i> .
4 (4) HWI_CMD_CBU	Capacity backup CPC feature operation. Note: The input connection token must only represent a <i>CPC connection</i> .
5 (5) HWI_CMD_OOCOD	On/Off capacity on demand request. Note: The input connection token must only represent a <i>CPC connection</i> .
6 (6) HWI_CMD_PROFILE	Access CPC activation profiles. Note: The input connection token must only represent a <i>CPC connection</i> .

Constant in Hexadecimal (Decimal) Equate Symbol	Description
7 (7) HWI_CMD_RESERVE	Set exclusive CPC control. Note: The input connection token must only represent a <i>CPC connection</i> .
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 connection token must only represent an <i>image connection</i> or an <i>image group connection</i> . 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 connection token must only represent an <i>image connection</i> or an <i>image group connection</i> . 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 connection token must only represent an <i>image connection</i> or an <i>image group connection</i> . This command cannot be issued specifying a connect token that represents the local image.
B (11) HWI_CMD_PSWRESTART	Restart request for one CP on target system. The first CP that is found to be in the correct state is reset. Note: The input connection token must only represent an <i>image connection</i> or an <i>image group connection</i> . 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 connection token must only represent an <i>image connection</i> .
D (13) HWI_CMD_LOAD	Load request to IPL target operating systems. Note: <ul style="list-style-type: none"> The input connection token must only represent an <i>image connection</i> or an <i>image group connection</i>. This command cannot be issued specifying a connect token that represents the local image. This CmdType can be invoked using two different input structures pointed by the specified CmdParm_Ptr. <ul style="list-style-type: none"> To specify a 4-digit load address, use HWICMD or HWICMD2 with CmdParmVersion = Hwi_ParmListVers_1. The HWI_CMD_LOAD_PARM mapping must be used. To specify a 5-digit load address, use HWICMD2 with CmdParmVersion = Hwi_ParmListVers_2. The HWI_CMD2_LOAD_PARM mapping must be used. Consult the IBM-supplied included files for further details.
E (14) HWI_CMD_TEMPCAP	Addition or removal of temporary capacity. Note: The input connection token must only represent a <i>CPC connection</i> . For more information see Writing XML for use with the temporary capacity SNMP APIs (www-01.ibm.com/servers/resourceLink/lib03011.nsf/pages/zCoDXMLforCoDCommands?OpenDocument) .
F (15) HWI_CMD_SYSRESET_IPLT	System reset request for target systems with IPL token correlation. This is an enhanced version of HWI_CMD_SYSRESET. Note: The input connection token must only represent an <i>image connection</i> .
10 (16) HWI_CMD_ACTIVATE_WITH_ACTPROF	Activate request to start target systems using a supplied activation profile name. This is an enhanced version of the HWI_CMD_ACTIVATE command. Note: The input connection token must only represent a <i>CPC connection</i> or an <i>image connection</i> .

Constant in Hexadecimal (Decimal) Equate Symbol	Description
11 (17) HWI_CMD_POWER_CONTROL	Control the power usage characteristics. Note: The input connection token must only represent a <i>CPC connection</i> .
12 (18) HWI_CMD_SCSI_LOAD	SCSI Load from FCP (Fibre Channel Protocol for SCSI) attached SCSI (Small Computer System Interface) disks. Note: <ul style="list-style-type: none"> The input connection token must only represent an <i>image connection</i> or an <i>image group connection</i>. This CmdType can be invoked using two different input structures pointed by the specified CmdParm_Ptr. <ul style="list-style-type: none"> To specify a 4-digit load address, use HWICMD or HWICMD2 with CmdParmVersion = Hwi_ParmListVers_1. The HWI_CMD_SCSICMD_PARM mapping must be used. To specify a 5-digit load address, use HWICMD2 with CmdParmVersion = Hwi_ParmListVers_2. The HWI_CMD_SCSICMD_PARM mapping must be used. Consult the IBM-supplied included files for further details.
13 (19) HWI_CMD_SCSI_DUMP	SCSI Dump to FCP (Fibre Channel Protocol for SCSI) attached SCSI (Small Computer System Interface) disks. Note: <ul style="list-style-type: none"> The input connection token must only represent an <i>image connection</i>. This CmdType can be invoked using two different input structures pointed by the specified CmdParm_Ptr. <ul style="list-style-type: none"> To specify a 4-digit load address, use HWICMD or HWICMD2 with CmdParmVersion = Hwi_ParmListVers_1. The HWI_CMD_SCSICMD_PARM mapping must be used. To specify a 5-digit load address, use HWICMD2 with CmdParmVersion = Hwi_ParmListVers_2. The HWI_CMD_SCSICMD_PARM mapping must be used. Consult the IBM-supplied included files for further details.
14 (20) HWI_CMD_SYSPLEX_TIME_SWAP_CTS	In a configured STP-only coordinated timing network (CTN), one CPC has the role of current time server (CTS). If the CTN has both a preferred time server and a backup time server configured, either one can be the CTS. This command swaps the role of CTS from preferred time server to backup time server or vice versa. The target system must be the system that will become the CTS. Note: The input connection token must only represent a <i>CPC connection</i> .
15 (21) HWI_CMD_SYSPLEX_TIME_SET_STP_CONFIG	This command sets the configuration for an STP-only coordinated timing network (CTN). The target system must be the system that will become the current time server (CTS). Note: The input connection token must only represent a <i>CPC connection</i> .
16 (22) HWI_CMD_SYSPLEX_TIME_CHANGE_STP_ONLY_CTN	This command, sent to the defined CPC with the role of current time server (CTS) in an STP-only coordinated timing network (CTN), changes the STP_ID portion of the CTN ID for the entire STP-only CTN. Note: The input connection token must only represent a <i>CPC connection</i> .
17 (23) HWI_CMD_SYSPLEX_TIME_JOIN_STP_ONLY_CTN	This command allows a CPC to join an STP-only coordinated timing network (CTN). The target system cannot be the current time server. If the CPC is already participating in an STP-only CTN, it will be removed from that CTN and joined to the specified one. If the CPC has an ETR ID, it will be removed. Note: The input connection token must only represent a <i>CPC connection</i> .  Attention: Use extreme caution when issuing this command. Joining the STP-only CTN may result in a disabled wait state for all images that are in a parallel sysplex on the target CPC.

Constant in Hexadecimal (Decimal) Equate Symbol	Description
18 (24) HWI_CMD_SYSPLEX_TIME _LEAVE_STP_ONLY_CTN	<p>This command removes a CPC from an STP-only coordinated timing network (CTN). The target system cannot be the current time server.</p> <p>Note: The input connection token must only represent a <i>CPC connection</i>.</p> <p> Attention: Use extreme caution when issuing this command. Leaving the STP-only CTN may result in a disabled wait state for all images that are in a parallel sysplex on the target CPC.</p>

CmdParm_Ptr (non-REXX)**CmdParm. (REXX)**

Supplied parameter

- Type: Pointer (non-REXX), stem variable (REXX)
- Length: 4 bytes (non-REXX)

Non-REXX:

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.
- For commands that target image groups, the parameters specified in the CmdParm must be appropriate for all the images in the image group.

REXX:

CmdParm stem contains compound (stem) variables which represent input parameters for the requested command. The tail names of the stem variable are constants which must match the parameter names in [Table 64 on page 280](#).

For optional parameters that are not initialized, BCPii interprets them as null parameters.

Table 64. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX)			
CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
ACTIVATE	HWI_CMD_ACT_PARM	ForceType	<p>A 4-byte integer (optional, the default is FORCE):</p> <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) <p>Note: Only a ForceType of HWI_CMD_FORCE will result in a successful activation of the target CPC or image if the target CPC or image is already active.</p>

Table 64. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
DEACTIVATE	HWI_CMD_DEACT_PARM	ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE)
HWMSG	HWI_CMD_HWMSG_PARM	HWMSGType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means REFRESH (HWI_CMD_HWMSG_REFRESH) • 2 – means DELETE (HWI_CMD_HWMSG_DELETE)
		HWMSGTimestamp	A null-terminated character string, up to 32 characters long. Required only for HWMSGType = HWI_CMD_HWMSG_DELETE. The timestamp specified must be an exact match of a timestamp returned on a HWMSGType = HWI_CMD_HWMSG_REFRESH request. An example of a timestamp: '08-20-2010 11:01: 23:145'. To delete a message, first run an HWI_CMD_HWMSG_REFRESH request to obtain the full timestamp and then issue the HWI_CMD_HWMSG_DELETE request, specifying the timestamp.
CBU	HWI_CMD_CBU_PARM	CBUType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means ACTIVATE (HWI_CMD_ACT) • 2 – means UNDO (HWI_CMD_UNDO)
		ActivateType	A 4-byte integer (required only for CBUType = HWI_CMD_ACT): <ul style="list-style-type: none"> • 1 – means REAL CBU (HWI_CMD_REAL) • 2 – means TEST CBU (HWI_CMD_TEST)
OOCOD	HWI_CMD_OOCOD_PARM	OOCODType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means ACTIVATE (HWI_CMD_ACT) • 2 – means UNDO (HWI_CMD_UNDO)
		OrderNumber	Required for OOCODType = HWI_CMD_ACT An 8-character string representing the order number of the On/Off Capacity on Demand (On/Off CoD) record to be activated. Note: The order number can be retrieved using the Hwi_RecID attribute via the HWIQUERY service.
PROFILE	HWI_CMD_PROFILE_PARM	ProfileType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means IMPORT (HWI_CMD_PROFILE_IMPORT) • 2 – means EXPORT (HWI_CMD_PROFILE_EXPORT)
		AreaNumber	A 2-byte integer area number is required and must be in the range of 1 to 4.

Table 64. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
RESERVE	HWI_CMD_RESERVE_PARM	ReserveType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means ADD (HWI_CMD_RESERVE_ADD) • 2 – means DELETE (HWI_CMD_RESERVE_DELETE)
		ApplName	An 8-character application name (required) padded with trailing blanks.
SYSRESET	HWI_CMD_SYSRESET_PARM	ResetType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means NORMAL (HWI_CMD_RESET_NORMAL) • 2 – means CLEAR (HWI_CMD_RESET_CLEAR)
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) Note: Only a ForceType of HWI_CMD_FORCE will result in a successful sysreset of the target CPC or image if the target CPC or image is already active.
START	0	N/A	N/A
STOP	0	N/A	N/A
PSWRESTART	0	N/A	N/A
OSCMD	HWI_CMD_OSCMD_PARM	PriorityType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means Priority (HWI_CMD_PRIORITY) • 2 – means Non-Priority (HWI_CMD_NONPRIORITY) Note: For WTOR replies targeting a z/OS image, a PriorityType of Non-Priority may need to be specified to allow z/OS to receive the reply command.
		OSCMDString	A 126-null-terminated character operating system command string (required).
LOAD	HWI_CMD_LOAD_PARM	LoadAddr	A 4-character string consisting only of hexadecimal characters identifying the device address to be used when performing the load (optional).
		LoadParm	An 8-character string as determined by the operating system being loaded (optional).
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) Note: Only a ForceType of HWI_CMD_FORCE will result in a successful load of the target CPC or image if the target CPC or image is already active.

Table 64. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)			
CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
	HWI_CMD2_LOAD_PARM	LoadAddr	A 5-character string consisting only of hexadecimal characters identifying the device address to be used when performing the load (optional).
		LoadParm	An 8-character string as determined by the operating system being loaded (optional).
		*	A 3-byte character string of any value to force proper boundary alignment.
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) Note: Only a ForceType of HWI_CMD_FORCE will result in a successful load of the target CPC or image if the target CPC or image is already active.
TEMPCAP	HWI_CMD_TEMPCAP_Parm	TEMPCAPType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means Add (HWI_CMD_TEMPCAP_ADD) • 2 – means Remove (HWI_CMD_TEMPCAP_REMOVE) For more information see Writing XML for use with the temporary capacity SNMP APIs (www-01.ibm.com/servers/resourcelink/lib03011.nsf/pages/zCoDXMLforCoDCommands?OpenDocument) .
		XML_Ptr (non-REXX)	A character string pointer that points to the address of the XML information that illustrates the markup used to perform activation of the temporary capacity (required).
		XML (REXX)	XML information that illustrates the markup used to perform activation of the temporary capacity (required).
		XML_Size (non-REXX)	A 4-byte integer (required). Length in bytes of the XML that the XML_Ptr points to.
SYSRESET _IPLT	HWI_CMD_SYSRESET _IPLT_PARM	ResetType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means NORMAL (HWI_CMD_RESET_NORMAL) • 2 – means CLEAR (HWI_CMD_RESET_CLEAR)
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) Note: Only a ForceType of HWI_CMD_FORCE will result in a successful sysreset of the target CPC or image if the target CPC or image is already active.

Table 64. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
		IPL-Token_Ptr (non-REXX)	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).
		IPL-Token (REXX)	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).
		IPL-Token_Len (non-REXX)	A 4-byte integer (required). Length in bytes of the IPL token to which the IPL-Token_Ptr points.
ACTIVATE_ WITH _ACTPROF	HWI_CMD_ACT_WITH_ ACTPROF_PARM	ActProfName	A 16-character activation profile name padded with trailing blanks (required).
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) Note: Only a ForceType of HWI_CMD_FORCE will result in a successful activation of the target CPC or image if the target CPC or image is already active.
POWER _CONTROL	HWI_CMD_POWER _CONTROL_PARM	XML_Ptr (non-REXX)	A character string pointer that points to the address of the XML fragment describing the power characteristics to be applied to the CPC specified by the connect token (required).
		XML (REXX)	XML fragment describing the power characteristics to be applied to the CPC specified by the connection token (required).
		XML_Size (non-REXX)	A 4-byte integer (required). Length in bytes of the XML that the XML_Ptr points to.
SCSI_LOAD	HWI_CMD_SCSICMD _LOAD_PARM	LoadAddr	A 4-character string (optional) consisting only of hexadecimal characters (0-9, A-F) identifying the device address to be used when performing the SCSI load. Defaults to value last used when previous SCSI Load was performed.
		LoadParm	An 8-character string (optional) as determined by the operating system being loaded. Defaults to value last used when previous SCSI Load was performed.
		WW_Portname	A 16-character string (optional) identifying the World Wide Port Name to be used when performing a SCSI Load. Defaults to value last used when previous SCSI Load was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).

Table 64. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)			
CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
		LU_Num	A 16-character string (optional) identifying the logical unit number (LUN) to be used when performing the SCSI Load. Defaults to value last used when previous SCSI Load was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		Boot_Pgm_Selector	A 4-byte integer (optional) identifying the boot program selector to be used for the SCSI Load. Defaults to value last used when previous SCSI Load was performed.
		Opsys_Loadparm	A 256-character string (optional) representing the operating system-specific load parameters to be used for the SCSI Load. Defaults to value last used when previous SCSI Load was performed.
		*	A 3-byte character string of any value to force proper boundary alignment.
		Bootrec_Blkc_Addr	A 16-character string (optional) representing the boot record logical block address to be used for the SCSI Load. Defaults to value last used when previous SCSI Load was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) Note: Only a ForceType of HWI_CMD_FORCE will result in a successful load of the target CPC or image if the target CPC or image is already active.
	HWI_CMD_SCASICMD2_LOAD_PARM	LoadAddr	A 5-character string (optional) consisting only of hexadecimal characters (0-9, A-F) identifying the device address to be used when performing the SCSI load. Defaults to value last used when previous SCSI Load was performed.
		LoadParm	An 8-character string (optional) as determined by the operating system being loaded. Defaults to value last used when previous SCSI Load was performed.
		WW_Portname	A 16-character string (optional) identifying the World Wide Port Name to be used when performing a SCSI Load. Defaults to value last used when previous SCSI Load was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		LU_Num	A 16-character string (optional) identifying the logical unit number (LUN) to be used when performing the SCSI Load. Defaults to value last used when previous SCSI Load was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).

Table 64. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
		*	A 3-byte character string of any value to force proper boundary alignment.
		Boot_Pgm_Selector	A 4-byte integer (optional) identifying the boot program selector to be used for the SCSI Load. Defaults to value last used when previous SCSI Load was performed.
		Opsys_Loadparm	A 256-character string (optional) representing the operating system-specific load parameters to be used for the SCSI Load. Defaults to value last used when previous SCSI Load was performed. Note: If less than 256 bytes, a null terminator signifies the end of the string.
		*	A 3-byte character string of any value to force proper boundary alignment.
		Bootrec_Blkc_Addr	A 16-character string (optional) representing the boot record logical block address to be used for the SCSI Load. Defaults to value last used when previous SCSI Load was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) Note: Only a ForceType of HWI_CMD_FORCE will result in a successful load of the target CPC or image if the target CPC or image is already active.
SCSI_DUMP	HWI_CMD_SCASICMD _DUMP_PARM	LoadAddr	A 4-character string (optional) consisting only of hexadecimal characters (0-9, A-F) identifying the device address to be used when performing the SCSI Dump. Defaults to value last used when previous SCSI Dump was performed.
		LoadParm	An 8-character string (optional) used when performing the SCSI dump. Defaults to value last used when previous SCSI Dump was performed.
		WW_Portname	A 16-character string (optional) identifying the World Wide Port Name to be used when performing a SCSI Dump. Defaults to value last used when previous SCSI Dump was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		LU_Num	A 16-character string (optional) identifying the logical unit number (LUN) to be used when performing the SCSI Dump. Defaults to value last used when previous SCSI Load was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).

Table 64. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
		Boot_Pgm_Selector	A 4-byte integer (optional) identifying the boot program selector to be used for the SCSI Dump. Defaults to value last used when previous SCSI Load was performed.
		Opsys_Loadparm	A 256-character string (optional) representing the operating system-specific load parameters to be used for the SCSI Dump. Defaults to value last used when previous SCSI Dump was performed. Note: If less than 256 bytes, a null terminator signifies the end of the string.
		*	A 3- byte character string of any value to force proper boundary alignment.
		Bootrec_Blkl_Addr	A 16-character string (optional) representing the boot record logical block address to be used for the SCSI Dump. Defaults to value last used when previous SCSI Dump was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) Currently, either ForceType value will cause the same result. The target image will be dumped in either case. IBM recommends that an application omit this parameter.
	HWI_CMD_SCSCMD2 _DUMP_PARM	LoadAddr	A 5-character string (optional) consisting only of hexadecimal characters (0-9, A-F) identifying the device address to be used when performing the SCSI Dump. Defaults to value last used when previous SCSI Dump was performed.
		LoadParm	An 8-character string (optional) used when performing the SCSI dump. Defaults to value last used when previous SCSI Dump was performed.
		WW_Portname	A 16-character string (optional) identifying the World Wide Port Name to be used when performing a SCSI Dump. Defaults to value last used when previous SCSI Dump was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		LU_Num	A 16-character string (optional) identifying the logical unit number (LUN) to be used when performing the SCSI Dump. Defaults to value last used when previous SCSI Load was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		*	A 3- byte character string of any value to force proper boundary alignment.

Table 64. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
		Boot_Pgm_Selector	A 4-byte integer (optional) identifying the boot program selector to be used for the SCSI Dump. Defaults to value last used when previous SCSI Load was performed.
		Opsys_Loadparm	A 256-character string (optional) representing the operating system-specific load parameters to be used for the SCSI Dump. Defaults to value last used when previous SCSI Dump was performed. Note: If less than 256 bytes, a null terminator signifies the end of the string.
		*	A 3- byte character string of any value to force proper boundary alignment.
		Bootrec_Blks_Addr	A 16-character string (optional) representing the boot record logical block address to be used for the SCSI Dump. Defaults to value last used when previous SCSI Dump was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) Note: Only a ForceType of HWI_CMD_FORCE will result in a successful load of the target CPC or image if the target CPC or image is already active.
SYSPLEX_TIME_SWAP_CTS	HWI_CMD_SYSPLXTIME_SWAP_CTS_PARM	STP_ID	An 8-character non-terminated string (required) representing the current STP identifier associated with this CPC.
SYSPLEX_TIME_SET_STP_CONFIG	HWI_CMD_SYSPLXTIME_SET_STP_CONFIG_PARM	STP_ID	An 8-character non-terminated string (required) representing the current STP identifier associated with this CPC.
		ForceType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE)
		XML_Ptr (non-REXX)	A character string pointer (required) points to the address of the XML fragment describing the configuration for the STP-only CTN.
		XML (REXX)	XML fragment describing the configuration for the STP-only CTN. (required)
		XML_Size (non-REXX)	A 4-byte integer (required). Length in bytes of the XML that the XML_Ptr points to.
SYSPLEX_TIME_CHANGE_STP_ONLY_CTN	HWI_CMD_SYSPLXTIME_CHG_STPONLYCTN_PARM	STP_ID	An 8-character non-terminated string (required) representing the desired STP identifier for the CPC and all CPCs that are members of the same STP-only CTN.
SYSPLEX_TIME_JOIN_STP_ONLY_CTN	HWI_CMD_SYSPLXTIME_JOIN_STPONLYCTN_PARM	STP_ID	An 8-character non-terminated string (required) representing the current STP identifier for the CPC.

Table 64. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)			
CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
SYSPLEX_TIME _LEAVE_STP _ONLY_CTN	0	N/A	N/A

CmdParmVersion (HWICMD2 only - non-REXX)

Supplied parameter.

- Type: Integer.
- Length: 4 bytes.

CmdParmVersion specifies the version of the CmdParm structure to be used, which allows multiple mappings of data to be specified to a particular command. See CmdType under “Parameters” on page 276 for specifications regarding the use of this parameter. If CmdParmVersion is not mentioned for a particular CmdType, the value must be set to Hwi_ParmListVers_1.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- Type: Character string (non-REXX), stem variable (REXX).
- Length: 32 bytes (non-REXX).

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.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value that is specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code that is returned from the console application API or the BCPii transport layer.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, “BCPii communication error reason codes,” on page 819 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with reason code X'0001yyyy' for HWICMD or X'0008yyyy' for HWICMD2 for one of the following reasons:

Table 65. Reasons for abend X'042', RC X'0001yyyy' for HWICMD or X'0008yyyy' for HWICMD2 for

yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See [z/OS MVS System Codes](#) for additional information.

Return codes

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

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
0 HWI_OK	0 HWI_OK	<p>Meaning: The command has been accepted by the support element. An SMF record has been written.</p> <p>Action: Determine the final command completion result by consulting the return code value found in the data returned by the command response event. This ENF event is signaled if the application has already registered to receive this event (HWIEVENT or HwiManageEvents service).</p>
100 HWI_CONNECT_TOKEN_INV	256 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified connect token is not valid. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • 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. <p>Action: Check for probable coding error.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
101 HWI_COMMUNICATION_ERROR	257 HWI_COMMUNICATION_ERROR	<p>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.</p> <p>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.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, “BCPii communication error reason codes,” on page 819.</p>
102 HWI_DIAGAREA_INV	258 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field.</p>
103 HWI_CONNECT_TOKEN_INACTIVE	259 HWI_CONNECT_TOKEN_INACTIVE	<p>Meaning: The specified connect token is no longer valid. The connection has been disconnected or it is in the progress of being disconnected.</p> <p>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.</p>
104 HWI_TARGET_CPC_CHANGED	260 HWI_TARGET_CPC_CHANGED	<p>Meaning: The CPC name represented by the specified token is valid but does not represent the same physical machine that was targeted by the initial HWICONN call. All connections that were established prior to the name change can no longer be used.</p> <p>Action: The application should cease using this connect token. If the application intends to target the CPC using the name represented by the specified connect token, it must first reconnect to the CPC before issuing any BCPii service call.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
602 HWI_CMDTYPE_INV	1538 HWI_CMDTYPE_INV	<p>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:</p> <ul style="list-style-type: none"> • 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'. • The CmdType specified applies only to image connections, but the ConnectToken specified represents an image group connection. The Diag_Text will indicate this error with the text of 'Mismatch'. <p>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.</p>
603 HWI_CMDPARM_INV	1539 HWI_CMDPARM_INV	<p>Meaning: Program error. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • Required parameters are missing. • One or more parameters specified are not valid. <p>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.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
604 HWI_CMD_TARGET_DEST_NOT_ALLOWED	1560 HWI_CMD_TARGET_DEST_NOT_ALLOWED	<p>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:</p> <ul style="list-style-type: none"> • Hwi_Cmd_Activate • Hwi_Cmd_Activate_With_Actprof • Hwi_Cmd_Deactivate <p>Commands that cannot target the local image include:</p> <ul style="list-style-type: none"> • Hwi_Cmd_Activate_With_Actprof • Hwi_Cmd_Sysreset_IPLT <p>Commands that cannot target the local image (by itself or as a member of a user-defined image group) are:</p> <ul style="list-style-type: none"> • Hwi_Cmd_Activate • Hwi_Cmd_Deactivate • Hwi_Cmd_Load • Hwi_Cmd_PswRestart • Hwi_Cmd_Start • Hwi_Cmd_Stop • Hwi_Cmd_Sysreset • Hwi_Cmd_SCSI_Load • Hwi_Cmd_SCSI_Dump <p>Action: BCPii does not allow this command to be executed against the local CPC or local image. Validate the name of the target represented by the input connection 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.</p> <p>If the ConnectToken represents a user-defined image group, verify that the group does not contain the local image where this command is executing.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
605 HWI_CMDPARM_INACCESSIBLE	1561 HWI_CMDPARM_INAC CESSIBLE	<p>Meaning: Program error. The CmdParm data area cannot be accessed. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • 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. <p>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.</p>
606 HWI_CMDTYPE_NOT_SUPPORTED	1562 HWI_CMDTYPE_NOT_S UPPORTED	<p>Meaning: The targeted hardware of the HWICMD request does not recognize the type of command being requested.</p> <p>Action: Verify that the targeted hardware is at a level that supports the type of command being issued.</p>
607 HWI_CMD_NOT_SUPPORTED	1563 HWI_CMD_NOT_SUPP ORTED	<p>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 298 for more information.</p> <p>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.</p>
608 HWI_CMD_IMAGE_GROUP_IS_EM PTY	1564 HWI_CMD_IMAGE_GR OUP_IS_EMPTY	<p>Meaning: Command did not execute because the connect token represents an image group that contains no images.</p> <p>Action: Ensure that the correct connect token was specified on the HWICMD request. If so, check with the SE/HMC engineer to determine the members that are in the group.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
609 HWI_CMDPARMVERSION_INV	1565 HWI_CMDPARMVERSI ON_INV	<p>Meaning: Program error. The requested CmdParmVersion specified in the call is not in the acceptable value range for the specified command.</p> <p>Note: This return code applies to only HWICMD2.</p> <p>Action: Check for probable error. Verify that the specified CmdParmVersion is not zero or is in the acceptable value range. If no CmdParmVersion is specified for the particular command, the value must be set to Hwi_ParmListVers_1. See the CmdType documentation prior for a list of acceptable values for the specified command.</p>
F00 HWI_NOT_AVAILABLE	3840 HWI_NOT_AVAILABLE	<p>Meaning: BCPii services are not available, and the system rejects the service request.</p> <p>Action: Notify the system programmer to start the BCPii address space and try the request again. See “Restarting the HWIBCPii address space” on page 261 about how to start the BCPii address space.</p> <p>Programs can also listen to ENF68 to determine when BCPii services are available. See <i>z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG</i> for how to listen for BCPii activation messages.</p>
F01 HWI_AUTH_FAILURE	3841 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.</p> <p>Action: Check the calling program for a probable coding error.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
F02 HWI_NO_SAF_AUTH	3842 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • 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.<i>netid.nau</i> for a CPC or image group connection. • Define control access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau.image</i>name for an image connection. • Define CONTROL access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau.image</i>name for each image within the target image group for an image group connection. Note: It is possible that an application may have the proper authority to all images in a user-defined image group returned on a prior HWILIST invocation, yet still receive this error return code. This could be because HWILIST will only return image names that the user has the proper authority to view. In this case, it will be necessary to contact the HMC/SE administrator to find out if there are other image names contained in the user-defined image group that were not returned on the HWILIST invocation. Once these names have been acquired, the security administrator may be contacted to give CONTROL or higher access to these additional image names. • Ensure that the referenced Facility Class Profile is RACLIST-specified.
F03 HWI_INTERRUPT_STATUS_INV	3843 HWI_INTERRUPT_STAT US_INV	<p>Meaning: The calling program is disabled. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWI_MODE_INV	3844 HWI_MODE_INV	<p>Meaning: The calling program is not in task mode. The system rejects this service request.</p> <p>Action: Check the calling program for a probable error.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
F05 HWI_LOCKS_HELD	3845 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	3846 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.
F07 HWI_UNSUPPORTED_ENVIRONMENT	3847 HWI_UNSUPPORTED_ENVIRONMENT	Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine). Action: Issue the BCPii service from a different execution environment.
FFF HWI_UNEXPECTED_ERROR	4095 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.

```
.
.
CmdType = HWI_CMD_ACTIVATE;
HWI_CmdTypeParm.ForceType = HWI_CMD_Force;
CmdParm_Ptr = addr(HWI_CmdTypeParm);
CALL HWICMD (ReturnCode, ConnectToken, CmdType,
             CmdParm_Ptr, DiagArea)
.
.
```

To issue the exact same command using HWICMD2, use the following pseudocode example:

```
CmdType = HWI_CMD_ACTIVATE;
HWI_CmdTypeParm.ForceType = HWI_CMD_Force;
CmdParmVersion = 1;
CmdParm_Ptr = addr(HWI_CmdTypeParm);
CALL HWICMD2 (ReturnCode, ConnectToken, CmdType,
              CmdParm_Ptr, CmdParmVersion, DiagArea);
```

A REXX programming example for the HWICMD service:

Note: The command parm field names must exactly match the field names in the command parm structure declarations.

```

myCmdType = HWI_CMD_OSCMD                               /* oscmd */
myCmdParm.PriorityType = Hwi_CMD_Priority
myCmdParm.OSCMDString = 'd a,l'
address bcpii
    "hwicmd RetCode myImgConnectToken myCmdType myCmdParm. myDiag."

If (RC <> 0) | (Retcode <> 0) Then
    Do
        Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
        If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
            Do
                Say ' Diag_index=' myDiag.DIAG_INDEX
                Say ' Diag_key=' myDiag.DIAG_KEY
                Say ' Diag_actual=' myDiag.DIAG_ACTUAL
                Say ' Diag_expected=' myDiag.DIAG_EXPECTED
                Say ' Diag_commer=' myDiag.DIAG_COMMERR
                Say ' Diag_text=' myDiag.DIAG_TEXT
            End
        End
    End
End

```

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, different types of activation profiles, a user-defined image group, a group profile or an LPAR Capacity group. This facilitates subsequent services to perform operations that are related to that CPC, image, capacity record, activation profile, a user-defined image group, a group profile or an LPAR Capacity group.

BCPii limits the total number of system-wide connections from all BCPii users to be no more than 5000 simultaneous connections.

Note: A connection remains active until one of the following occurs:

- A Disconnect service call (HWIDISC) has been invoked.
- A parent connection has been disconnected.
- A loss of connectivity to the associated CPC has been detected by BCPii.
- The address space of the caller has terminated.
- The current task of the caller has terminated if the connection has task affinity (TSO/E REXX or ISV-provided REXX execution environments).
- The BCPii address space has terminated.

Under normal circumstances, a connection remains active indefinitely. Since there are a finite number of total BCPii connections available in the entire system, a BCPii application should disconnect any BCPii connection it no longer needs.

Note: BCPii requires the FACILITY class to be RACLIST-specified. BCPii also automatically transforms the following to all uppercase characters when building the profile names passed to the security product: CPC, image, and caprec values pointed to by the ConnectTypeValue_Ptr.

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
Dispatchable unit mode:	Task

Requirement	Details
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 263 for details about how to call BCPii services in the various programming languages.

REXX programming considerations for the HWICONN service

All information for the HWICONN service applies for REXX requests except:

- ConnectTypeValue replaces ConnectTypeValue_Ptr.

Restrictions

BCPii does not allow HWICONN to be issued from within a BCPii ENF exit routine.

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_ACTPROF, HWI_IMAGE_ACTPROF, HWI_LOAD_ACTPROF, HWI_IMAGE_GROUP, HWI_GROUP_PROFILE or HWI_LPAR_GROUP connections.
- Read access to the SAF-protected FACILITY class resource HWI.TARGET.*netid.nau.image*name for HWI_IMAGE connections.
- Read access to the SAF-protected FACILITY class resource HWI.CAPREC.*netid.nau.cap*recid for HWI_CAPREC connections.

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

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWICONN(ReturnCode, InConnectToken, OutConnectToken, ConnectType, ConnectTypeValue_Ptr, DiagArea);</pre>	<pre>address bcpaii "hwiconn ReturnCode InConnectToken OutConnectToken ConnectType ConnectTypeValue DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

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, activation profile, user-defined image group, group profile, and LPAR Capacity group connections.

In most cases, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call. For BCPII REXX execs running under TSO/E or ISV-provided REXX environments, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same task as this service call.

InConnectToken is not relevant to a connect type of HWI_CPC, and either must be left uninitialized or initialized to HWI_NULL_CONNECTTOKEN constant.

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), activation profile, group profiles or an LPAR Capacity group.

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_ACTPROF, HWI_IMAGE_ACTPROF, or HWI_LOAD_ACTPROF connection can be specified on subsequent HWIQUERY or HWISET / HWISET2 service calls to query or set specific values associated with the specified Reset, image, or Load activation profile respectively.

A connection token returned for an HWI_IMAGE_GROUP can be specified on a subsequent HWIQUERY service call to query values associated with the group profile, on a subsequent HWICMD service call to issue commands to all members in the image group, or on a subsequent HWILIST service call to list the images in the image group.

A connect token returned for an HWI_GROUP_PROFILE connection can be specified on subsequent HWIQUERY or HWISET / HWISET2 service calls to query or set specific values associated with the specified group profile respectively.

A connect token returned for an HWI_LPAR_GROUP connection can be specified on subsequent HWIQUERY or HWISET / HWISET2 service calls to query or set specific values associated with the specified LPAR Group profile respectively.

ConnectType

Supplied parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

ConnectType specifies the type of connection to be established.

Constant in Hexadecimal (Decimal) Equate Symbol	Description
1 (1) HWI_CPC	Requests to establish a connection to a target CPC that the application is to communicate with.
2 (2) HWI_IMAGE	Requests to establish a connection to an image of a CPC that the application is to communicate with. The input connection token must represent an active CPC connection.
3 (3) HWI_CAPREC	Requests to establish a connection to a capacity record of a CPC that the application is to communicate with. The input connection token must represent an active CPC connection.
4 (4) HWI_RESET_ACTPROF	Requests to establish a connection to a reset activation profile associated with a particular CPC. The input connection token must represent an active CPC connection.
5 (5) HWI_IMAGE_ACTPROF	Requests to establish a connection to an image activation profile associated with a particular CPC. The input connection token must represent an active CPC connection.
6 (6) HWI_LOAD_ACTPROF	Requests to establish a connection to a load activation profile associated with a particular CPC. The input connection token must represent an active CPC connection.
7 (7) HWI_IMAGE_GROUP	Requests to establish a connection to a user-defined image group on a particular CPC. The input connection token must represent an active CPC connection. Note: This ConnectType is only available when targeting a z10 or higher CPC.
8 (8) HWI_GROUP_PROFILE	Requests to establish a connection to a group profile on a particular CPC. The input connection token must represent an active CPC connection.
9 (9) HWI_LPAR_GROUP	Requests to establish a connection to an LPAR capacity group on a particular CPC. The input connection token must represent an active CPC connection. Note: This ConnectType is only available when targeting a z14 GA2 or higher CPC.

ConnectTypeValue_Ptr (non-REXX)

ConnectTypeValue (REXX)

Supplied parameter

- Type: Pointer (non-REXX), character (REXX)
- Length: 4 bytes (non-REXX)

Non-REXX:

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

REXX:

ConnectTypeValue is the name of the requested target to be connected to. The type of connection determines the value required.

Connect Types	Values to be specified
HWI_CPC	<ul style="list-style-type: none"> 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. <p>Note: Both the netid and NAU name must consist of alphanumeric characters (0-9, A-Z).</p> <p>Example: net1.cpc01</p> <ul style="list-style-type: none"> 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 (<i>netid.nau</i>). <p>Note: An HWILIST HWI_LIST_CPCS operation returns a list of CPCs available to be connected to in the form of <i>netid.nau</i>.</p>
HWI_IMAGE	<p>An 8-character image name padded with trailing blanks.</p> <p>Note: The LPAR name is a 1- through 8-alphanumeric (0-9, A-Z,a-z) character name that must have an alphabetic first character. Special characters (\$, #, @), although currently allowed, are being reserved for future use. See <i>PR/SM Planning Guide</i> for details.</p>
HWI_CAPREC	<p>An 8-character capacity record (CAPREC) name padded with trailing blanks.</p> <p>Note: The CAPREC name is a 1- to 8-alphanumeric (0-9, A-Z, a-z) character name.</p>
HWI_RESET_ACTPROF	<p>A 16-character alphanumeric (0-9, A-Z,a-z) reset activation profile name padded with trailing blanks.</p>
HWI_IMAGE_ACTPROF	<p>A 16-character alphanumeric (0-9, A-Z,a-z) image activation profile name padded with trailing blanks.</p>
HWI_LOAD_ACTPROF	<p>A 16-character alphanumeric (0-9, A-Z,a-z) load activation profile name padded with trailing blanks.</p>
HWI_IMAGE_GROUP	<p>A 30 character null-terminated image group name.</p>
HWI_GROUP_PROFILE	<p>An 8-character alphanumeric (0-9, A-Z, a-z) group profile name padded with trailing blanks.</p>
HWI_LPAR_GROUP	<p>An 8-character alphanumeric (0-9, A-Z, a-z) LPAR capacity group name padded with trailing blanks.</p>

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

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.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value that is specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code that is returned from the console application API or the BCPii transport layer.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See [Appendix A, “BCPii communication error reason codes,”](#) on page 819 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0002yyyy' because of one of the following reasons:

<i>Table 66. Reasons for abend X'042', RC X'0002yyyy'</i>	
yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See [z/OS MVS System Codes](#) for additional information.

Return codes

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
0 HWI_OK	0 HWI_OK	Meaning: Successful completion. Action: None.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
4 HWI_CMD_NOT_SUPPORTED_WARNING	4 HWI_CMD_NOT_SUPPORTED_WARNING	<p>Meaning: Successful completion. This warning return code is informational.</p> <p>The target CPC being connected to has a microcode level (MCL) that does not support HWICMD, 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). If a subsequent HWICMD is issued with this returned connect token, the call will be rejected with a return code of HWI_CMD_NOT_SUPPORTED.</p> <p>Action: 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.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
100 HWI_CONNECT_TOKEN_INV	256 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified input connection token is not valid. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The input connection token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken. • The input connection 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. • The input connection 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. <p>Action: Check for probable coding error.</p>
101 HWI_COMMUNICATION_ERROR	257 HWI_COMMUNICATION_ERROR	<p>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.</p> <p>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.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, “BCPii communication error reason codes,” on page 819.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
102 HWI_DIAGAREA_INV	258 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field.</p>
103 HWI_CONNECT_TOKEN_INACTIVE	259 HWI_CONNECT_TOKEN_INACTIVE	<p>Meaning: The specified connect token is no longer valid. The connection has been disconnected or it is in the progress of being disconnected.</p> <p>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.</p>
104 HWI_TARGET_CPC_CHANGED	260 HWI_TARGET_CPC_CHANGED	<p>Meaning: The CPC name represented by the specified token is valid but does not represent the same physical machine that was targeted by the initial HWICONN call. All connections that were established prior to the name change can no longer be used.</p> <p>Action: The application should cease using this connect token. If the application intends to target the CPC using the name represented by the specified connect token, it must first reconnect to the CPC before issuing any BCPII service call.</p>
201 HWI_CONNTYPE_INV	513 HWI_CONNTYPE_INV	<p>Meaning: Program error. The connection type specified in the call is not valid. The system rejects the service call.</p> <p>Action: Check for probable coding error. Validate that the conntype value passed to the service is one of the accepted values.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
202 HWI_CONNTYPE_VALUE_INV	514 HWI_CONNTYPE_VALUE_INV	<p>Meaning: Program error. This return code indicates that one of the following conditions has occurred:</p> <ol style="list-style-type: none"> 1. 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. 2. When targeting a z14 or higher CPC, the BCPii request does not have proper permission granted by the SE for the target object. <p>Action: Check for probable coding error. Verify that the connection name is syntactically correct, valid in the current HMC configuration, and currently available. If the target CPC/LPAR is a z14 or higher, verify that the proper BCPii firmware security has been granted to allow this BCPii application to access the CPC/LPAR.</p>
203 HWI_CONNTYPE_VALUE_INACCESSIBLE	515 HWI_CONNTYPE_VALUE_INACCESSIBLE	<p>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.</p> <p>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.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
204 HWI_MAX_CONNECTIONS_REACHED	516 HWI_MAX_CONNECTIONS_REACHED	<p>Meaning: The number of connections has reached the maximum number of system-wide connections (5000) that BCPii permits, or BCPii has run out of system resources to satisfy the HWICONN request, or both.</p> <p>Action: Disconnect connections that are no longer needed, and try the request again.</p>
205 HWI_CONNTYPE_NOT_SUPPORTED	517 HWI_CONNTYPE_NOT_SUPPORTED	<p>Meaning: The targeted hardware of the HWICONN request does not support the connect type specified.</p> <p>Action: Verify that the targeted hardware supports the type of request being made.</p>
F00 HWI_NOT_AVAILABLE	3840 HWI_NOT_AVAILABLE	<p>Meaning: BCPii services are not available, and the system rejects the service request.</p> <p>Action: Notify the system programmer to start the BCPii address space and try the request again. See “Restarting the HWIBCPii address space” on page 261 about how to start the BCPii address space.</p> <p>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.</p>
F01 HWI_AUTH_FAILURE	3841 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.</p> <p>Action: Check the calling program for a probable coding error.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
F02 HWI_NO_SAF_AUTH	3842 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • 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.<i>netid.nau</i> for a CPC, activation profile, image group connection, group profile, or LPAR Capacity group connection. • Define read access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau.imagen ame</i> for an image connection. • Define read access authorization to the FACILITY class resource profile HWI.CAPREC.<i>netid.nau.capreci d</i> for a capacity record connection. • Ensure that the referenced Facility Class Profiles are RACLIST-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 254 for further information regarding community name setup.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
F03 HWI_INTERRUPT_STATUS_INV	3843 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	3844 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	3845 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	3846 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.
F07 HWI_UNSUPPORTED_ENVIRONMENT	3847 HWI_UNSUPPORTED_ENVIRONMENT	Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine). Action: Issue the BCPii service from a different execution environment.
FFF HWI_UNEXPECTED_ERROR	4095 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.

```
.
.
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.)
.
.
```

A REXX programming example for the HWICONN service:

```
myConnectType      = HWI_CPC           /* CPC connect type */
myConnectTypeValue = 'IBM390xx.H123    ' /* 17-char CPC name */

address bcp11
    "hwiconn Retcode myInConnectToken myOutConnectToken myConnectType
    myConnectTypeValue myDiag."

If (RC <> 0) | (Retcode <> 0) Then
    Do
        Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
        If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
            Do
                Say ' Diag_index=' myDiag.DIAG_INDEX
                Say ' Diag_key=' myDiag.DIAG_KEY
                Say ' Diag_actual=' myDiag.DIAG_ACTUAL
                Say ' Diag_expected=' myDiag.DIAG_EXPECTED
                Say ' Diag_commerk=' myDiag.DIAG_COMMERR
                Say ' Diag_text=' myDiag.DIAG_TEXT
            End
        End
    End
```

HWIDISC — Release a BCPii connection

Call the HWIDISC service to release the logical connection between the application and the identified CPC, image, capacity record, different types of activation profiles, user-defined imagegroup, group profile or LPAR Capacity group. If the connect token represents a CPC, any subordinate image, capacity record, activation profile, user-defined image group, group profile or LPAR Capacity group connection associated with the same CPC connection is also released.

Description

Environment

The requirements for the callers are:

Requirement	Details
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)

Requirement	Details
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 263 for details about how to call BCPii services in the various programming languages.

REXX programming considerations for the HWIDISC service

All information for the HWIDISC service applies for REXX requests except:

- In the System REXX environment, BCPii connections are associated with the address space that issued the AXREXX macro service call. When this address space terminates, BCPii will implicitly disconnect the connection.
- In the TSO/E and ISV-provided REXX environments, BCPii connections are associated with the current running task. When this task terminates, BCPii will implicitly disconnect the connection.

Restrictions

BCPii does not allow HWIDISC to be issued from within a BCPii ENF exit routine.

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_ACTPROF, HWI_IMAGE_ACTPROF, HWI_LOAD_ACTPROF, HWI_IMAGE_GROUP, HWI_GROUP_PROFILE or HWI_LPAR_GROUP connections.
- The SAF-protected FACILITY class resource HWI.TARGET.*netid.nau.image*name for HWI_IMAGE connections.
- The SAF-protected FACILITY class resource HWI.CAPREC.*netid.nau.caprecid* for HWI_CAPREC connections.

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

Syntax

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

Non-REXX parameters	REXX parameters
CALL HWIDISC(ReturnCode, ConnectToken, DiagArea);	address bcpaii "hwidisc ReturnCode ConnectToken DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

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, activation profile, or user-defined image group, group profile or LPAR Capacity group and is returned as an output parameter on the HWICONN service call.

In most cases, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call. For BCPII REXX execs running under the TSO/E or ISV-provided REXX environments, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same task.

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

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.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code that is returned from the console application API or the BCPII transport layer.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See [Appendix A, “BCPII communication error reason codes,”](#) on page 819 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPII is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0003yyyy' because of one of the following reasons:

Table 67. Reasons for abend X'042', RC X'0003yyyy'

yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See [z/OS MVS System Codes](#) for additional information.

Return codes

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

Return Code in Hexadecimal Equate Symbol	Return Code in Hexadecimal Equate Symbol	Meaning and Action
0 HWI_OK	0 HWI_OK	Meaning: Successful completion. Action: None.
100 HWI_CONNECT_TOKEN_INV	256 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: <ul style="list-style-type: none"> • The input connection 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.

Return Code in Hexadecimal Equate Symbol	Return Code in Hexadecimal Equate Symbol	Meaning and Action
101 HWI_COMMUNICATION_ERROR	257 HWI_COMMUNICATION_ERROR	<p>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.</p> <p>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.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, “BCPii communication error reason codes,” on page 819.</p>
102 HWI_DIAGAREA_INV	258 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field.</p>
901 HWI_DISC_INPROGRESS	2305 HWI_DISC_INPROGRESS	<p>Meaning: Another Disconnect request is already in progress. This request is redundant.</p> <p>Action: None.</p>
F00 HWI_NOT_AVAILABLE	3840 HWI_NOT_AVAILABLE	<p>Meaning: BCPii services are not available, and the system rejects the service request.</p> <p>Action: Notify the system programmer to start the BCPii address space and try the request again. See “Restarting the HWIBCPii address space” on page 261 about how to start the BCPii address space.</p> <p>Programs can also listen to ENF68 to determine when BCPii services are available. See <i>z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG</i> for how to listen for BCPii activation messages.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Hexadecimal Equate Symbol	Meaning and Action
F01 HWI_AUTH_FAILURE	3841 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWI_NO_SAF_AUTH	3842 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • 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.<i>netid.nau</i> for a CPC, activation profile, image group, group profile or LPAR Capacity group connection. • Define read access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau.imagen ame</i> for an image connection. • Define read access authorization to the FACILITY class resource profile HWI.CAPREC.<i>netid.nau.capreci d</i> for a capacity record connection. • Ensure that the referenced Facility Class Profiles are RACLIST-specified.
F03 HWI_INTERRUPT_STATUS_INV	3843 HWI_INTERRUPT_STATUS_INV	<p>Meaning: The calling program is disabled. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Hexadecimal Equate Symbol	Meaning and Action
F04 HWI_MODE_INV	3844 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	3845 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	3846 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.
F07 HWI_UNSUPPORTED_ENVIRONMENT	3847 HWI_UNSUPPORTED_ENVIRONMENT	Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine). Action: Issue the BCPii service from a different execution environment.
FFF HWI_UNEXPECTED_ERROR	4095 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 release a connection between the application and a CPC.

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

```

.
```

A REXX programming example for the HWIDISC service:

```

address bcpaii
    "hwidisc Retcode myConnectToken myDiag."

If (RC <> 0) | (Retcode <> 0) Then
    Do
        Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
        If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
            Do
                Say ' Diag_index=' myDiag.DIAG_INDEX
                Say ' Diag_key=' myDiag.DIAG_KEY
                Say ' Diag_actual=' myDiag.DIAG_ACTUAL
                Say ' Diag_expected=' myDiag.DIAG_EXPECTED
                Say ' Diag_commerk=' myDiag.DIAG_COMMERR
                Say ' Diag_text=' myDiag.DIAG_TEXT
            End
        End
    End
End
```

HWIEVENT — Register or unregister for BCPii events

Call the HWIEVENT service for the following purposes:

1. Register an application and its connection to receive notification of:
 - One or more hardware or software events occurring on the connected CPC or image.
 - Communication errors between BCPii and the connected CPC or image.
2. Delete the registration for one or more previously registered events.

Monitoring events occurring on a particular CPC or image

For hardware and software events, an application can register with BCPii to be notified when an event occurs for the targeted CPC or image. Under the covers, BCPii communicates the registration request with the support element (SE) of the targeted CPC or image if necessary and also registers the user-provided exit with the Event Notification Facility (ENF). When the event occurs on the targeted CPC or image, BCPii receives notification and signals the appropriate ENF68. The user's exit receives control with data unique for the event that just occurred. The data mapping for these different events can be found in the public interface files shipped with BCPii (HWICIC for the C programming language, HWICIREX for the REXX programming language, and HWICIASM for the assembler programming language). BCPii also provides a sample of an ENF event exit in SYS1.SAMPLIB (HWIXMCX1) that can be a good starting point for coding a BCPii ENF exit.

Note: BCPii user-defined image groups are a powerful way to issue commands to all members of a group simultaneously. Commands targeted to a user-defined image group will result in one image command response event being generated for each image in the image group. If event notification is desired for an image in an image group, register the image for the command response event to enable delivery of the event to the BCPii ENF exit.

Monitoring operating system message events (Hwi_Event_OpSysMsg)

Your application can monitor all operating system messages appearing on a z/OS console by using the HWIEVENT service to register for the EventIDs parameter value Hwi_Event_OpSysMsg.

For the majority of messages issued on the image being monitored, a single BCPii operating system message event will contain the entire message in the returned event data (HWIENF68 data mapping).

For messages that are larger than approximately 3000 bytes, it is possible that the operating system message is longer than the architected maximum buffer size allowed by the communications protocol used by both the z/OS consoles component and BCPii to communicate with the support element. As a result, BCPii delivers these single large messages in multiple operating system message events. Each of these operating system message events representing a single large message will have the same values in

the HWIENF68 data mapping for the msgId, msgDate, and msgTime fields. An application can determine that all of the operating system message events have been delivered for the single large message by consulting the msgId of a subsequent message event. If it has changed from the previous msgId, the operating system message event represents a new operating system message.

Monitoring communication availability between BCPii and the CPC

While not common, BCPii may occasionally experience communication delays or interruptions of service between itself and the targeted CPC and its associated support element. BCPii provides a mechanism through its BCPii communication error class of events to detect these interruptions and to allow an application to know when these interruptions of service have been resolved.

BCPii keeps a heartbeat between itself and each CPC where its applications desire connectivity. If BCPii fails to receive its regular heartbeat from an SE associated with a CPC, BCPii signals either an expected or unexpected communication error (ENF QUAL value '02010004'x/'02010005'x). BCPii then attempts a communication flow to this SE (see [“BCPii communication monitoring”](#) on page 261 for more information). If the SE responds successfully to this communication attempt by BCPii, BCPii signals a *temporary communication error*, (ENF QUAL value 02010001), indicating the communication path between BCPii and the SE seems to be operational at this time. However, during the communication interruption, one or more events may have been lost.

Once a communication available event (ENF QUAL value '02010003'x) has been issued, applications currently having valid connections to that CPC and its images are allowed to resume issuing HWIEVENT and HWICMD APIs targeting that CPC and its images. Events originating from the CPC and its images now begin to arrive again.

If the SE does not respond to the BCPii communication attempt, BCPii assumes that there is a serious communication problem and signals a *permanent communication error*, (ENF QUAL value 02010002). At this point, no HWIEVENT or HWICMD API requests to this CPC are processed by BCPii and no event delivery take place for events registered on this CPC and its images. BCPii closes its internal connections with the CPC and cleans up resources associated with command processing and event delivery to and from this CPC.

In addition, to regaining communication after service interruptions, BCPii also signals the communication available event, (ENF QUAL value '02010003'x) when it begins to monitor a previously unmonitored CPC.

An application may choose to register for these communication availability events via the HWIEVENT ADD service (EventIDs parameter value Hwi_Event_HwCommError), or it may choose to use the ENFREQ LISTEN macro to listen for these events apart from any specific BCPii connection.

Monitoring the status of the BCPii address space

An application can monitor the status of the BCPii address space itself by using the ENFREQ LISTEN service and specifying the appropriate QUAL values to monitor when the BCPii address space becomes active and when it terminates:

- BCPii signals an ENF68 with a QUAL value of 01000002 when the BCPii address space becomes active.
- BCPii signals an ENF68 with a QUAL value of 01000001 when the BCPii address space becomes unavailable.

While it is possible to use the HWIEVENT service to allow an application to register for the Hwi_Event_BCPiiStatus event, this is not a recommended way to monitor initialization or termination of the BCPii address space. When the BCPii address space terminates, BCPii asynchronously asks the system to delete all ENF registrations made on behalf of applications that have issued HWIEVENT Add requests. If the deletion of the ENF registration occurs prior to the BCPii address space termination, the ENF exit will no longer receive control when BCPii signals that it is down.

Description

Environment

The requirements for the callers are:

Requirement	Details
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 263 for details about how to call BCPii services in the various programming languages. For programming language C, see [“Restrictions”](#) on page 320.

See [“HWIEVENT”](#) on page 823 for the summary table of the BCPii HWIEVENT types and the objects that can be registered or unregistered for each event.

REXX programming considerations for the HWIEVENT service

All information for the HWIEVENT service applies for REXX requests except:

- EventIDs is a 32-element stem-variable representing all of the event bits as defined in the HWICIREX include file.
- Because the Event Notification Facility (ENF) does not support REXX exits, the caller must provide the address of a non-REXX ENF exit routine.
- The EventExitAddr must be specified as the 8-character representation of a 4-byte hexadecimal value.

Restrictions

- This service is not used by C language callers running in a z/OS UNIX System Services environment. See [“HWIManageEvents — Manage the list of BCPii events”](#) on page 453.
- BCPii does not allow HWIEVENT to be issued from within a BCPii ENF exit routine.
- BCPii does not allow HWIEVENT to be issued from a REXX exec running in the TSO/E or ISV-provided REXX environments.

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.image*name for ConnectToken representing an image connection.

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

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWIEVENT(ReturnCode, ConnectToken, EventAction, EventIDs, EventExitMode, EventExitAddr, EventExitParm, DiagArea);</pre>	<pre>address bcp11 "hwievent ReturnCode ConnectToken EventAction EventIDs. EventExitMode EventExitAddr EventExitParm DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

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 (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

EventAction specifies the type of action for the service.

Constant in Hexadecimal (Decimal) Equate Symbol	Description
1 (1) HWI_EVENT_ADD	Registers to be notified when the requested events occur.
2 (2) HWI_EVENT_DELETE	Deletes the registration for notification.

EventIDs (non-REXX)

EventIDs. (REXX)

Supplied parameter

- Type: Integer (non-REXX), stem variable (REXX)
- Length: 128 bits (16 bytes) (non-REXX)

EventIDs specifies the events to be added or deleted.

Non-REXX:

Each event is a 1-bit field from bit position 97 to 128 in this data area. If the bit is on, the service performs the EventAction operation for the event on the requested connection.

REXX:

Each event is represented by an IBM-supplied EventIDs tail label or tail value constant. If the value is on, the service performs the EventAction operation for the event on the requested connection.

It is recommended to use the IBM-supplied EventIDs tail labels defined in HWICIREX.

Note: A single connection may not register for a particular event more than once.

The following event IDs or tail labels can be specified:

EventIDs (non-REXX) / tail label for EventIDs stem (REXX)	Bit position in structure specified on EventIDs (non-REXX)	Tail value constant of the user-defined EventIDs stem (REXX)	Description
Hwi_EventID_EyeCatcher	1-96	N/A	Control block identifier. Note: HWI_EVENTID_TEXT can be used to initialize this field.
Hwi_Event_CmdResp	97	1	Requests to add or delete the registration for notification of the command response events. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
Hwi_Event_StatusChg	98	2	Requests to add or delete the registration for notification of the status change events. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
Hwi_Event_NameChg	99	3	Requests to add or delete the registration for notification of the object name change events. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
Hwi_Event_ActProfChg	100	4	Requests to add or delete the registration for notification of the change events for the activation profile name. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
Hwi_Event_ObjCreate	101	5	Requests to add or delete the registration for notification of the object created events. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
Hwi_Event_ObjDestroy	102	6	Requests to add or delete the registration for notification of the object destroyed (deleted) events. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
Hwi_Event_ObjException	103	7	Requests to add or delete the registration for notification of the exception state events. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .

EventIDs (non-REXX) / tail label for EventIDs stem (REXX)	Bit position in structure specified on EventIDs (non-REXX)	Tail value constant of the user-defined EventIDs stem (REXX)	Description
Hwi_Event_ApplStarted	104	8	Requests to add or delete the registration for notification of the console application started events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_ApplEnded	105	9	Requests to add or delete the registration for notification of the console application ended events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_HwMsg	106	10	Requests to add or delete the registration for notification of the hardware message events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_HwMsgDel	107	11	Requests to add or delete the registration for notification of the hardware message deletion events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_SecurityEvent	108	12	Requests to add or delete the registration for notification of the support element (SE) console security events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_CapacityChg	109	13	Requests to add or delete the registration for notification of the capacity change events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_CapacityRecord	110	14	Requests to add or delete the registration for notification of the capacity record change events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_OpSysMsg	111	15	Requests to add or delete the registration for notification of the operating system message events. Note: The input connection token must only represent an <i>image connection</i> .
Hwi_Event_HwCommError	112	16	Requests to add or delete the registration for notification of the hardware communication error events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_BCPIIStatus	113	17	Requests to add or delete the registration for notification of BCPII status change events. Note: This method is not recommended for determining if the BCPII address space becomes available or unavailable. See the description of the HWIEVENT service for more information.
Hwi_Event_DisabledWait	114	18	Requests to add or delete the registration for notification of disabled wait events. Note: The input connection token must only represent an <i>image connection</i> .

EventIDs (non-REXX) / tail label for EventIDs stem (REXX)	Bit position in structure specified on EventIDs (non-REXX)	Tail value constant of the user-defined EventIDs stem (REXX)	Description
Hwi_Event_PowerChange	115	19	Requests to add or delete the registration for notification of any power characteristics change events. Note: The input connection token must represent a <i>CPC connection</i> .
Hwi_Event_Shutdown	116	20	Requests to add or delete the registration for notification of any shutdown events. The shutdown event was introduced in z16 and indicates that the Support Element associated with the target CPC will be shutting down or restarting soon. Note: The input connection token must represent a CPC connection. The target CPC must be a z16 or higher.
Hwi_Event_Reserved	117-128	N/A	Reserved, must be initialized to binary zeros.

EventExitMode

Supplied parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

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

Constant in Hexadecimal (Decimal) Equate Symbol	Description
1 (1) HWI_EVENT_TASK	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.

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 (non-REXX), character representation of a pointer (REXX)
- Length: 4 bytes (non-REXX), 8 characters (REXX)

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](#). 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](#).

EventExitParm

Supplied parameter

- Type: Pointer or integer (non-REXX), character representation of a pointer or integer (REXX)
- Length: 4 bytes (non-REXX), up to 8 numeric characters (REXX)

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 (non-REXX)**DiagArea. (REXX)**

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

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.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The return code that is returned from the console application API or the BCPii transport layer.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, “BCPii communication error reason codes,” on page 819 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0004yyyy' because of one of the following reasons:

<i>Table 68. Reasons for abend X'042', RC X'0004yyyy'</i>	
yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See [z/OS MVS System Codes](#) for additional information.

Return codes

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

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
0 HWI_OK	0 HWI_OK	Meaning: Successful completion. Action: None.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
100 HWI_CONNECT_TOKEN_INV	256 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified connect token is not valid. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • 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. <p>Action: Check for probable coding error.</p>
101 HWI_COMMUNICATION_ERROR	257 HWI_COMMUNICATION_ERROR	<p>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.</p> <p>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. BCPiis CTRACE might provide further diagnostic information if the problem can not easily be resolved. See z/OS MVS System Commands for further information about starting and stopping CTRACE.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, “BCPii communication error reason codes,” on page 819.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
102 HWI_DIAGAREA_INV	258 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field.</p>
103 HWI_CONNECT_TOKEN_INACTIVE	259 HWI_CONNECT_TOKEN_INACTIVE	<p>Meaning: The specified connect token is no longer valid. The connection has been disconnected, or it is in the progress of being disconnected.</p> <p>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.</p>
104 HWI_TARGET_CPC_CHANGED	260 HWI_TARGET_CPC_CHANGED	<p>Meaning: The CPC name represented by the specified token is valid but does not represent the same physical machine that was targeted by the initial HWICONN call. All connections that were established prior to the name change can no longer be used.</p> <p>Action: The application should cease using this connect token. If the application intends to target the CPC using the name represented by the specified connect token, it must first reconnect to the CPC before issuing any BCPii service call.</p>
701 HWI_EVENT_EXITMODE_INV	1793 HWI_EVENT_EXITMODE_INV	<p>Meaning: Program error. The requested EventExitMode on the call is not valid. The system rejects the service call.</p> <p>Action: Check for probable coding error.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
702 HWI_EVENT_EXITADDR_INV	1794 HWI_EVENT_EXITADDR_INV	<p>Meaning: Program error. The requested EventExitAddr on the call is not valid. The system rejects the service call.</p> <p>Action: Check for probable coding error.</p>
703 HWI_EVENT_ACTION_INV	1795 HWI_EVENT_ACTION_INV	<p>Meaning: Program error. The requested EventAction on the call is not valid. The system rejects the service call.</p> <p>Action: Check for probable coding error.</p>
704 HWI_EVENT_IDS_INV	1796 HWI_EVENT_IDS_INV	<p>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:</p> <ul style="list-style-type: none"> • The first 12 bytes of the EventIDs parameter is not equal to the expected Eyecatcher of HWIEVENTBLCK (non-REXX only). • 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. <p>Action: Check for probable coding error.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
F00 HWI_NOT_AVAILABLE	3840 HWI_NOT_AVAILABLE	<p>Meaning: BCPii is not available, and the system rejects the service request.</p> <p>Action: Notify the system programmer to start the BCPii address space and try the request again. See “Restarting the HWIBCPii address space” on page 261 about how to start the BCPii address space.</p> <p>Programs can also listen to ENF68 to determine when BCPii services are available. See <i>z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG</i> for how to listen for BCPii activation messages.</p>
F01 HWI_AUTH_FAILURE	3841 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWI_NO_SAF_AUTH	3842 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • 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.<i>netid.nau</i> for CPC connection. • Define read access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau.imagen ame</i> for an image connection. • Ensure that the referenced FACILITY class profiles are RACLIST-specified.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
F03 HWI_INTERRUPT_STATUS_INV	3843 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	3844 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	3845 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	3846 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.
F07 HWI_UNSUPPORTED_ENVIRONMENT	3847 HWI_UNSUPPORTED_ENVIRONMENT	Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine). Action: Issue the BCPii service from a different execution environment.
F08 HWI_UNSUPPORTED_HWRELEASE	3848 HWI_UNSUPPORTED_HWRELEASE	Meaning: The targeted hardware of the HWIEVENT request does not support the event being requested. Action: Verify that the targeted hardware is at a level that supports the type of event being requested.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
FFF HWI_UNEXPECTED_ERROR	4095 HWI_UNEXPECTED_ERROR	<p>Meaning: System error. The service that was called encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

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 DiagArea 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_Reserved = 0;
EventExitMode = HWI_EVENT_TASK;
EventExitAddr = ADDR(EventExit);
EventExitParm = 0;

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

```

A REXX programming example for the HWIEVENT service:

```

myAction = HWI_EVENT_ADD
myEventIDs. = 0 /*Initialize all EventIds to 0 */
myEventIDs.Hwi_Event_CmdResp = 1
myEventIDs.Hwi_Event_StatusChg = 1
myEventIDs.Hwi_Event_ActProfChg = 1

myMode = HWI_EVENT_TASK
myEventExitAddr = 0F123456 /* char rep of 4 byte hex address */
myEventExitParm = 0

address bcpii
"hwievent RetCode myConnectToken myEventAction myEventIDs. myEventExitMode
 myEventExitAddr myEventExitParm myDiag."

If (RC <> 0) | (Retcode <> 0) Then
  Do
    Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
    If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
      Do
        Say ' Diag_index=' myDiag.DIAG_INDEX
        Say ' Diag_key=' myDiag.DIAG_KEY
        Say ' Diag_actual=' myDiag.DIAG_ACTUAL
        Say ' Diag_expected=' myDiag.DIAG_EXPECTED
        Say ' Diag_commerr=' myDiag.DIAG_COMMERR
        Say ' Diag_text=' myDiag.DIAG_TEXT
      End
    End
  End

```

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, load activation profile, group profile or LPAR Capacity group using the HWICONN API.
- Register for the proper events (HWIEVENT) using the HWIEVENT API.
- Connect to the local CPC or image.
- Connect to a user-defined image group.

Note: A returned CPC name does not guarantee that an application will be able to connect to that particular resource using the HWICONN API. Connecting to a CPC involves setup issues such as setting up connectivity to a support element and defining the necessary BCPii community name on both the support element and the security product. For more information about the steps that need to be completed before connectivity to a particular CPC is complete, see [“Setting up connectivity to the support element” on page 246](#) and [“Community name defined in the security product for each CPC” on page 254](#). In addition, if the CPC configuration changes, the previously returned list may include CPC names which are no longer valid. A new HWILIST should be issued to retrieve the most current information.

Description

Environment

The requirements for the callers are:

Requirement	Details
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 263](#) for details about how to call BCPii services in the various programming languages.

REXX programming considerations for the HWILIST service

All information for the HWILIST service applies for REXX requests except:

- An answer area stem variable (for example, AnswerArea) replaces AnswerArea_Ptr.
- AnswerArea.0 replaces NumOfDataItemsReturned.

- AnswerArea.*i* will contain the *i*-th list value on return. For a list type of HWI_LIST_EVENTS, AnswerArea.*i* will contain the *i*-th event bit value on return.
- AnswerAreaLen is not returned.

Restrictions

BCPii does not allow HWILIST to be issued from within a BCPii ENF exit routine.

BCPii does not allow HWILIST with a ListType of HWI_LIST_EVENTS to be issued by a REXX exec running in the TSO/E REXX or ISV-provided REXX environments.

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 and if the input ConnectToken represents a CPC or Image Group connection, 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.

If the input ConnectToken represents an LPAR capacity group connection, all image names associated with the group will be listed (provided the Application has at least read access to the HWI.TARGET.*netid.nau* FACILITY class resource for the CPC where the group is defined).

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_ACTPROF, HWI_LIST_IMAGE_ACTPROF, or HWI_LIST_LOAD_ACTPROF, when BCPii is creating the list of activation profiles names and if the input ConnectToken represents a CPC connection, 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.

For a ListType of HWI_LIST_IMAGE_ACTPROF, if the input ConnectToken represents a group profile connection, all image activation profile names associated with the corresponding group profile will be listed (provided the Application has at least read access to the HWI.TARGET.*netid.nau* FACILITY class resource for the CPC to which the group profile applies).

For a ListType of HWI_LIST_IMAGEGROUPS, an application must have at least read access to the HWI.TARGET.*netid.nau* FACILITY class resource for the CPC on which image groups may be defined.

For a ListType of HWI_LIST_GROUP_PROFILES, when BCPii is creating the list of group profile 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 group profiles apply.

For a ListType of HWI_LIST_LPAR_GROUPS, an application must have at least read access to the HWI.TARGET.*netid.nau* FACILITY class resource for the CPC to which the LPAR Capacity groups apply.

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

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWILIST(ReturnCode, ConnectToken, ListType, NumOfDataItemsReturned, AnswerArea_Ptr, AnswerAreaLen, DiagArea);</pre>	<pre>address bcpii "hwilist ReturnCode ConnectToken ListType AnswerArea. DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

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 other entity. The ConnectToken is an output parameter on the HWICONN service call.

In most cases, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call. For BCPii REXX execs running under TSO/E or ISV-provided REXX environments, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same task.

If the ListType is HWI_LIST_CPCS, HWI_LIST_LOCALCPC, or HWI_LIST_LOCALIMAGE, this parameter is not relevant and is ignored.

If the ListType is HWI_LIST_IMAGES, this request must either be directed to a specific CPC, one of its defined LPAR Capacity groups or to a specific user-defined image group. Therefore, a connect token that represents an active CPC, LPAR Capacity group, or user-defined image group connection must be specified.

If the ListType is HWI_LIST_CAPRECS, any of the activation profile (APROF) list types, HWI_LIST_IMAGEGROUPS, HWI_LIST_GROUP_PROFILES, or HWI_LIST_LPAR_GROUPS, this request must be directed to a specific CPC. Therefore, a connect token that represents an already active HWI CPC connection must be specified.

If the ListType is HWI_LIST_IMAGE_ACTPROF, this request must be directed to a specific group profile. Therefore, a connect token that represents an already active HWI group profile connection must be specified.

For a ListType of HWI_LIST_EVENTS, the connect token must represent an already active HWI 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 (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

ListType specifies the type of request for the service.

Constant in Hexadecimal (Decimal) Equate Symbol	Description
1 (1) HWI_LIST_CPCS	Requests a list of CPCs that can be accessed.
2 (2) HWI_LIST_IMAGES	Requests a list of image names. The specified connection token can represent a CPC, user defined image group, or an LPAR Capacity group connection. Note: This ListType for LPAR Capacity group (HWI_LPAR_GROUP) connection is only available when targeting a z14 GA2 or higher CPC.
3 (3) HWI_LIST_EVENTS	Requests a list of previously subscribed events. Note: This ListType is not supported for REXX execs running in the TSO/E or ISV-provided REXX environments.
4 (4) HWI_LIST_CAPRECS	Requests a list of capacity record ID names that can be accessed.
5 (5) HWI_LIST_LOCALCPC	Requests the name of the local CPC on which the caller is currently executing.
6 (6) HWI_LIST_LOCALIMAGE	Requests the name of the local image (LPAR) on which the HWILIST caller is currently executing.
7 (7) HWI_LIST_RESET_ACTPROF	Requests a list of the currently defined reset activation profiles.
8 (8) HWI_LIST_IMAGE_ACTPROF	Requests a list of the currently defined image activation profiles. The specified connection token can represent a CPC connection token or a group profile connection token. Note: This ListType for group profile (HWI_GROUP_PROFILE) connection is only available when targeting a z14 GA2 or higher CPC.
9 (9) HWI_LIST_LOAD_ACTPROF	Requests a list of the currently defined load activation profiles.
A (10) HWI_LIST_IMAGEGROUPS	Requests a list of the currently defined user-defined image groups. Note: This ListType is only available when targeting a z10 or higher CPC.
B (11) HWI_LIST_GROUP_PROFILES	Requests a list of the currently defined group profiles.

Constant in Hexadecimal (Decimal) Equate Symbol	Description
C (12) HWI_LIST_LPAR_GROUPS	Requests a list of the currently defined LPAR Capacity groups. Note: This ListType is only available when targeting a z14 GA2 or higher CPC.

NumofDataItemsReturned (non-REXX)

Returned parameter

- Type: Integer
- Length: 4 bytes

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

AnswerArea_Ptr (non-REXX)**AnswerArea. (REXX)**

Supplied parameter

- Type: Pointer (non-REXX), stem variable (REXX)
- Length: 4 bytes (non-REXX)

Non-REXX:

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

REXX:

A list of the requested objects is returned in an array form of $x.n$; where x is the user-defined AnswerArea stem variable and n is the n -th element in the stem array.

The AnswerArea.0 stem variable counter holds the number of items returned.

The ListType specified determines the format of the returned data.

ListType	Data to be returned (non-REXX)	Data to be returned (REXX)
HWI_LIST_CPCS	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.	A stem array list of 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. Example: net1.cpc01.
HWI_LIST_IMAGES	A string comprised of a list of blank-separated concatenated 8-character image names padded with trailing blanks.	A stem array list of image names.
HWI_LIST_EVENTS	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.	A stem array list of Boolean values of the EventIDs, which are represented by the EventIDs tail labels defined in HWICIREX. For example, if x is the answerarea stem variable, the returned Boolean data indicates the event registration status. $x.Hwi_Event_CmdResp = 1$ (on) $x.Hwi_Event_StatusChg = 0$ (off) : : $x.Hwi_Event_PowerChange = 0$ (off) Note: This ListType is not supported for REXX execs running in the TSO/E or ISV-provided REXX environments.

ListType	Data to be returned (non-REXX)	Data to be returned (REXX)
HWI_LIST_CAPRECS	A string comprised of a list of blank-separated concatenated 8-character CAPREC names padded with trailing blanks.	A stem array list of CAPREC names.
HWI_LIST_LOCALCPC	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.	The CPC network address of the local CPC is returned in the first and only element in the stem array. The 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.
HWI_LIST_LOCALIMAGE	An 8-character string representing the image name of the local image (LPAR) padded with trailing blanks.	The image name of the local image (LPAR) is returned in the first and only element in the stem array.
HWI_LIST_RESET_ACTPROF	A string comprised of a list of concatenated 16-character reset activation profile names padded with trailing blanks.	A stem array list of reset activation profile names.
HWI_LIST_IMAGE_ACTPROF	A string comprised of a list of concatenated 16-character image activation profile names padded with trailing blanks.	A stem array list of image activation profile names.
HWI_LIST_LOAD_ACTPROF	A string comprised of a list of concatenated 16-character load activation profile names padded with trailing blanks.	A stem array list of load activation profile names.
HWI_LIST_IMAGEGROUPS	A null-terminated string of null-separated user-defined image group names.	A stem array list of user-defined image group names.
HWI_LIST_GROUP_PROFILES	A string comprised of a list of concatenated 8-character alphanumeric group profile names padded with trailing blanks.	A stem array list of group profile names.
HWI_LIST_LPAR_GROUPS	A string comprised of a list of concatenated 8-character alphanumeric LPAR Capacity group names padded with trailing blanks.	A stem array list of LPAR Capacity Group names.

AnswerAreaLen (non-REXX)

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 (non-REXX)**DiagArea. (REXX)**

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

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.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code that is returned from the console application API or the BCPii transport layer.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See [Appendix A, “BCPii communication error reason codes,”](#) on page 819 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0005yyyy' because of one of the following reasons:

<i>Table 69. Reasons for abend X'042', RC X'0005yyyy'</i>	
yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See [z/OS MVS System Codes](#) for additional information.

Return codes

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

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
0 HWI_OK	0 HWI_OK	Meaning: Successful completion. Action: None.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
100 HWI_CONNECT_TOKEN_INV	256 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified connect token is not valid. This return code indicates one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • 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. <p>Action: Check for probable coding error.</p>
101 HWI_COMMUNICATION_ERROR	257 HWI_COMMUNICATION_ERROR	<p>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.</p> <p>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.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, “BCPii communication error reason codes,” on page 819.</p>
102 HWI_DIAGAREA_INV	258 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify the specified DiagArea is defined as a 32-byte character field.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
103 HWI_CONNECT_TOKEN_INACTIVE	259 HWI_CONNECT_TOKEN_INACTIVE	<p>Meaning: The specified connect token is no longer valid. The connection has been disconnected, or it is in the progress of being disconnected.</p> <p>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.</p>
104 HWI_TARGET_CPC_CHANGED	260 HWI_TARGET_CPC_CHANGED	<p>Meaning: The CPC name represented by the specified token is valid but does not represent the same physical machine that was targeted by the initial HWICONN call. All connections that were established prior to the name change can no longer be used.</p> <p>Action: The application should cease using this connect token. If the application intends to target the CPC using the name represented by the specified connect token, it must first reconnect to the CPC before issuing any BCPii service call.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
301 HWI_LISTTYPE_INV	769 HWI_LISTTYPE_INV	<p>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:</p> <ul style="list-style-type: none"> • The ListType specified is not in the acceptable value range of possible list types. • The ListType specified is incompatible with the InConnectToken specified. For example: <ul style="list-style-type: none"> – The ListType specified applies only to CPC connections, but the ConnectToken specified represents an image connection. – The ListType specified applies only to image connections, but the ConnectToken specified represents a CPC 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. <p>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.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
302 HWI_DATA_EXCEEDED	770 HWI_DATA_EXCEEDED	<p>Meaning: Program error. The amount of returned data exceeded the size of the answer area. No data or only partial data is returned.</p> <p>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.</p>
303 HWI_ANSWERAREA_INACCESSIBLE	771 HWI_ANSWERAREA_INACCESSIBLE	<p>Meaning: Program error. The answer area data area is either partially or completely inaccessible by the application and the Base Control Program internal interface (BCPii) address space.</p> <p>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.</p>
304 HWI_LIST_NODATA_RETURNED	772 HWI_LIST_NODATA_RETURNED	<p>Meaning: There is no data to be returned or the caller does not have enough access to display the listed values.</p> <p>Action: Check for probable coding error. Verify that proper access is granted for the request.</p>
305 HWI_LISTTYPE_NOT_SUPPORTED	773 HWI_LISTTYPE_NOT_SUPPORTED	<p>Meaning: The targeted hardware of the HWILIST request does not support the request attempted by the program.</p> <p>Action: Verify that the targeted hardware supports the type of request being made.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
F00 HWI_NOT_AVAILABLE	3840 HWI_NOT_AVAILABLE	<p>Meaning: BCPii services are not available, and the system rejects the service request.</p> <p>Action: Notify the system programmer to start the BCPii address space and try the request again. See “Restarting the HWIBCPii address space” on page 261 about how to start the BCPii address space.</p> <p>Programs can also listen to ENF68 to determine when BCPii services are available. See <i>z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG</i> for how to listen for BCPii activation messages.</p>
F01 HWI_AUTH_FAILURE	3841 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.</p> <p>Action: Check the calling program for a probable coding error.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
F02 HWI_NO_SAF_AUTH	3842 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV. • For a ListType of HWI_LIST_IMAGES, if the ConnectToken represents an LPAR Capacity group connection, define read access authorization to the FACILITY class resource profile HWI.TARGET.nedit.nau. For other connection types, define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagen ame. • 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 for a CPC connection, and HWI.TARGET.netid.nau.imagen ame 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 <i>netid.nau</i> represents the local CPC network address.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
F02 HWI_NO_SAF_AUTH (<i>continued</i>)	3842 HWI_NO_SAF_AUTH	<ul style="list-style-type: none"> For a ListType of HWI_LIST_LOCALIMAGE, define read access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau.imagen ame</i> where <i>netid.nau</i> represents the local CPC network address and <i>imagen ame</i> represents the local image (LPAR) name. For the ListType of HWI_LIST_RESET_ACTPROF, HWI_LIST_IMAGE_ACTPROF, HWI_LIST_LOAD_ACTPROF, HWI_LIST_IMAGEGROUPS, HWI_LIST_GROUP_PROFILES or HWI_LIST_LPAR_GROUPS define read access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau</i> for the CPC where the activation profiles, image groups, group profiles or LPAR Capacity groups to be listed are defined. Ensure that the referenced facility class profiles are RACLIST-specified.
F03 HWI_INTERRUPT_STATUS_INV	3843 HWI_INTERRUPT_STATUS_INV	<p>Meaning: The calling program is disabled. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWI_MODE_INV	3844 HWI_MODE_INV	<p>Meaning: The calling program is not in task mode. The system rejects this service request.</p> <p>Action: Check the calling program for a probable error.</p>
F05 HWI_LOCKS_HELD	3845 HWI_LOCKS_HELD	<p>Meaning: The calling program is holding one or more locks. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
F06 HWI_UNSUPPORTED_RELEASE	3846 HWI_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects this service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again.</p>
F07 HWI_UNSUPPORTED_ENVIRONMENT	3847 HWI_UNSUPPORTED_ENVIRONMENT	<p>Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine).</p> <p>Action: Issue the BCPii service from a different execution environment.</p>
FFF HWI_UNEXPECTED_ERROR	4095 HWI_UNEXPECTED_ERROR	<p>Meaning: System error. The service that was called encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Example

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

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

A REXX programming example for the HWILIST service:

```
myListType = HWI_LIST_IMAGES

address bcp11
"hwilist RetCode myConnectToken myListType myAnswerArea. myDiag."

If (RC <> 0) | (Retcode <> 0) Then
  Do
    Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
    If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
      Do
        Say ' Diag_index=' myDiag.DIAG_INDEX
        Say ' Diag_key=' myDiag.DIAG_KEY
```

```

        Say ' Diag_actual=' myDiag.DIAG_ACTUAL
        Say ' Diag_expected=' myDiag.DIAG_EXPECTED
        Say ' Diag_commer=' myDiag.DIAG_COMMERR
        Say ' Diag_text=' myDiag.DIAG_TEXT
    End
Else
    Do
        Say 'Number of items returned = 'myAnswerArea.0 /* Count of items returned */

        If myAnswerArea.0 > 0 Then
            Do n=1 to myAnswerArea.0
                Say 'Image #'n' = 'myAnswerArea.n
            End
        End
    End
End

```

HWIQUERY – BCPii retrieval of SE/HMC-managed attributes

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, different types of activation profiles, user-defined image groups, group profiles or LPAR Capacity groups.

For some connection types (HWI_CPC and HWI_IMAGE in particular), grouping multiple attributes together into a single HWIQUERY service call may result in significantly reduced waiting times rather than querying the same number of attributes one at a time. Whenever possible, an application should consolidate its HWIQUERY service calls to query multiple attributes using the same query request.

Description

Environment

The requirements for the callers are:

Requirement	Details
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 263](#) for details about how to call BCPii services in the various programming languages.

See [“HWIQUERY and HWISET / HWISET2 attributes” on page 825](#) for the summary table of the BCPii HWIQUERY and HWISET / HWISET2 attributes and the objects that can be targeted for each function.

REXX programming considerations for the HWIQUERY service

All information for the HWIQUERY service applies for REXX requests except:

- A query parameter stem variable (for example, QueryParm) replaces QueryParm_Ptr.
 - QueryParm.0 replaces NumOfAttributes. QueryParm.0 is required to specify the number of attributes to be queried. The maximum number of attributes allowed is 64.
 - QueryParm.*n*.ATTRIBUTEIDENTIFIER must contain the *n*-th attribute identifier to be returned.
 - QueryParm.*n*.ATTRIBUTEVALUE will contain the *n*-th attribute value on return.
- AttributeValue_Ptr is replaced with AttributeValue.
- AttributeValueLen is not used.
- AttributeValueLenReturned is not used.
- For the PSW (HWI_PSW) attribute:
 - QueryParm.*n*.ATTRIBUTEVALUE.0 will contain the number of PSWs returned (*j*).
 - QueryParm.*n*.ATTRIBUTEVALUE.*m*.CPUID will contain the *m*-th CPU identifier.
 - QueryParm.*n*.ATTRIBUTEVALUE.*m*.PSW will contain the *m*-th PSW.
- For the supported processor power savings mode (HWI_SUPPPPOWERMODE) attribute:
 - QueryParm.*n*.ATTRIBUTEVALUE.0 will contain the number of supported power savings modes returned (*m*).
 - QueryParm.*n*.ATTRIBUTEVALUE.*m*.PSMODE will contain the *m*-th supported power savings mode.
- For the list of IP addresses (HWI_LIST_IP_ADDRESSES) attribute:
 - QueryParm.*n*.ATTRIBUTEVALUE.0 will contain the number of IP addresses returned (*j*).
 - QueryParm.*n*.ATTRIBUTEVALUE.*m*.IPADDR will contain the *m*-th IP address.

Restrictions

BCPii does not allow HWIQUERY to be issued from within a BCPii ENF exit routine.

Code page consideration

All returned data from the Support Element is in ASCII format. BCPii attempts to translate and return the data in EBCDIC. Due to the nature of EBCDIC-to-ASCII and ASCII-to-EBCDIC conversions, certain irregularities exist in the conversion tables. These conversion irregularities, including characters like ¢, !, [,] and |, will not translate correctly. BCPii users of the HWIQUERY service should be aware that these characters may not be correct in the returned data.

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, activation profile, user-defined image group, group profile or LPAR Capacity group queries, or HWI.TARGET.*netid.nau.imagename* for image queries, or HWI.CAPREC.*netid.nau.caprecid* for capacity record queries.

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

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWIQUERY(ReturnCode, ConnectToken, QueryParm_Ptr, NumOfAttributes, DiagArea);</pre>	<pre>address bcpii "hwquery ReturnCode ConnectToken QueryParm. DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

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, activation profile, or user-defined image group. The ConnectToken is an output parameter on the HWICONN service call.

In most cases, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call. For BCPII REXX execs running under the TSO/E or ISV-provided REXX environments, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same task.

QueryParm_Ptr (non-REXX)

QueryParm. (REXX)

Supplied parameter

- Type: Pointer (non-REXX), stem variable (REXX)
- Length: 4 bytes (non-REXX)

Non-REXX:

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 NumOfAttributes 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 in the following table:

Field name	Field type
AttributeIdentifier	32-bit unsigned integer
AttributeValue_Ptr	Pointer
AttributeValueLen	32-bit unsigned integer
AttributeValueLenReturned	32-bit unsigned integer

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 263 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.

REXX:

QueryParm is a compound (stem) variable which contains one or more requested attributes to be queried and returned.

The compound (stem) variable is specified as follows (where *x* is the user-defined QueryParm stem variable and *n* is the *n*-th attribute for the request):

- **x.0 specifies the number of attributes to be queried.** The maximum number of attributes allowed is 64. (Supplied parameter)
- **x.n.ATTRIBUTEIDENTIFIER** specifies the requested attribute. Set this variable to one of the query attribute constants defined in HWICIREX. (Supplied parameter)
- **x.n.ATTRIBUTEVALUE** is the data value to be returned for most attributes. (Returned parameter)
- Some single attributes can return multiple objects in a formatted structure. For those attributes, **x.n.ATTRIBUTEVALUE.0** (Returned parameter) is the total number of returned objects. See the following query attribute table for the following attributes that are in a different format. These attributes include: HWI_SUPPPPOWERMODE, HWI_LIST_IP_ADDRESSES, and HWI_PSWS.

The following table lists the valid query attribute identifiers. For more information about these attributes, see the following publications:

- *IBM z SNMP Application Programming Interfaces* (SB10-7171-06)
- *System z10 and eServer zSeries Application Programming Interfaces* (SB10-7030-09)
- *System z9 and eServer zSeries Application Programming Interfaces* (SB10-7030-08)
- Publication appropriate to the level of hardware that the HWIQUERY is targeted

Table 70. Valid query attribute identifiers	
Constant in hexadecimal (Decimal) Equate symbol	Description
1 (1) HWI_NAME	Requests to retrieve the name that represents the ConnectToken parameter to the service. Note: The input connection token must represent a <i>CPC connection</i> , an <i>image connection</i> , a <i>reset activation profile connection</i> , an <i>image activation profile connection</i> , a <i>load activation profile connection</i> , an <i>image group connection</i> , a <i>group profile</i> or an <i>LPAR Capacity group connection</i> .
2 (2) HWI_ERRSTAT	Requests to retrieve whether the status is acceptable. Note: The input connection token must represent a <i>CPC connection</i> an <i>image connection</i> , or an <i>image group connection</i> .
3 (3) HWI_BUSYSTAT	Requests to retrieve whether the status is busy. Note: The input connection token must represent a <i>CPC connection</i> an <i>image connection</i> , or an <i>image group connection</i> .
4 (4) HWI_MSGSTAT	Requests to retrieve whether hardware messages are present. Note: The input connection token must represent a <i>CPC connection</i> or an <i>image connection</i> .
5 (5) HWI_OPERSTAT	Requests to retrieve the current status. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .

Table 70. Valid query attribute identifiers (continued)																					
Constant in hexadecimal (Decimal) Equate symbol	Description																				
6 (6) HWI_ACCSTAT	Requests to retrieve the acceptable status values. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .																				
7 (7) HWI_APROF	Requests to retrieve the next activation reset profile name. Note: The input connection token must represent a <i>CPC connection</i> or an <i>image connection</i> .																				
8 (8) HWI_LUAPROF	Requests to retrieve the last used activation profile. Note: The input connection token must represent a <i>CPC connection</i> or an <i>image connection</i> .																				
9 (9) HWI_OBJTYPE	Requests to retrieve the object type. <table border="0"> <tr> <td>Input connection token represents</td><td>Returns</td></tr> <tr> <td>CPC</td><td>HWMCA_CPC_OBJECT</td></tr> <tr> <td>CPC image</td><td>HWMCA_CPC_IMAGE_OBJECT</td></tr> <tr> <td>Capacity record</td><td>HWMCA_CAPACITY_RECORD</td></tr> <tr> <td>Reset activation profile</td><td>HWMCA_ACT_PROFILE_RESET</td></tr> <tr> <td>Image activation profile</td><td>HWMCA_ACT_PROFILE_IMAGE</td></tr> <tr> <td>Load activation profile</td><td>HWMCA_ACT_PROFILE_LOAD</td></tr> <tr> <td>Image Group</td><td>HWMCA_CPC_IMAGE_USER_GROUP</td></tr> <tr> <td>Group profile</td><td>HWCMA_ACT_PROFILE_GROUP</td></tr> <tr> <td>LPAR Capacity group</td><td>HWCMA_LPAR_GROUP</td></tr> </table> Note: The input connection token must represent a <i>CPC connection</i> , an <i>image connection</i> , a <i>capacity record connection</i> , a <i>reset activation profile connection</i> , an <i>image activation profile connection</i> , a <i>load activation profile connection</i> , an <i>image group connection</i> , a <i>group profile connection</i> , or an <i>LPAR Capacity group connection</i> .	Input connection token represents	Returns	CPC	HWMCA_CPC_OBJECT	CPC image	HWMCA_CPC_IMAGE_OBJECT	Capacity record	HWMCA_CAPACITY_RECORD	Reset activation profile	HWMCA_ACT_PROFILE_RESET	Image activation profile	HWMCA_ACT_PROFILE_IMAGE	Load activation profile	HWMCA_ACT_PROFILE_LOAD	Image Group	HWMCA_CPC_IMAGE_USER_GROUP	Group profile	HWCMA_ACT_PROFILE_GROUP	LPAR Capacity group	HWCMA_LPAR_GROUP
Input connection token represents	Returns																				
CPC	HWMCA_CPC_OBJECT																				
CPC image	HWMCA_CPC_IMAGE_OBJECT																				
Capacity record	HWMCA_CAPACITY_RECORD																				
Reset activation profile	HWMCA_ACT_PROFILE_RESET																				
Image activation profile	HWMCA_ACT_PROFILE_IMAGE																				
Load activation profile	HWMCA_ACT_PROFILE_LOAD																				
Image Group	HWMCA_CPC_IMAGE_USER_GROUP																				
Group profile	HWCMA_ACT_PROFILE_GROUP																				
LPAR Capacity group	HWCMA_LPAR_GROUP																				
A (10) HWI_IMLMODE	Requests to retrieve the initial machine load (IML) mode (LPAR). Note: The input connection token must only represent a <i>CPC connection</i> or an <i>image connection</i> .																				
B-16 (11–22) RESERVED	Reserved for attributes that are common to CPC and image connections unless otherwise noted.																				
17 (23) HWI_IPADDR	Requests to retrieve the internet address (IPv4 format). Note: The input connection token must only represent a <i>CPC connection</i> .																				
18 (24) HWI_SNAADDR	Requests to retrieve the SNA address (<i>netid.nau</i>). Note: The input connection token must only represent a <i>CPC connection</i> .																				
19 (25) HWI_MMODEL	Requests to retrieve the machine model. Note: The input connection token must only represent a <i>CPC connection</i> .																				
1A (26) HWI_MTYPE	Requests to retrieve the machine type. Note: The input connection token must only represent a <i>CPC connection</i> .																				
1B (27) HWI_MSERIAL	Requests to retrieve the machine serial. Note: The input connection token must only represent a <i>CPC connection</i> .																				
1C (28) HWI_CPCSERIAL	Requests to retrieve the CPC serial number. Note: The input connection token must only represent a <i>CPC connection</i> .																				

Table 70. Valid query attribute identifiers (continued)

Constant in hexadecimal (Decimal) Equate symbol	Description
1D (29) HWI_CPCID	Requests to retrieve the CPC identifier. Note: The input connection token must only represent a <i>CPC connection</i> .
1E (30) HWI_RESERVEID	Requests to retrieve the name of the application that is holding the reserve (if any). Note: The input connection token must only represent a <i>CPC connection</i> .
1F (31) HWI_SVCEREQD	Requests to retrieve the service required. Note: The input connection token must only represent a <i>CPC connection</i> .
20 (32) HWI_CBUINST	Requests to retrieve the CBU installed. Note: The input connection token must only represent a <i>CPC connection</i> .
21 (33) HWI_CBUENABLD	Requests to retrieve the CBU enabled. Note: The input connection token must only represent a <i>CPC connection</i> .
22 (34) HWI_CBUACTIVE	Requests to retrieve the CBU activated. Note: The input connection token must only represent a <i>CPC connection</i> .
23 (35) HWI_CBUACTDT	Requests to retrieve the CBU activation date. Note: The input connection token must only represent a <i>CPC connection</i> .
24 (36) HWI_CBUEXPDT	Requests to retrieve the CBU expiration date. Note: The input connection token must only represent a <i>CPC connection</i> .
25 (37) HWI_CBUTESTAR	Requests to retrieve the CBU tests left (test activations remaining). Note: The input connection token must only represent a <i>CPC connection</i> .
26 (38) HWI_CBUREALAV	Requests to retrieve the CBU real activation available. Note: The input connection token must only represent a <i>CPC connection</i> .
27 (39) HWI_PRUNTYPE	Requests to retrieve the processor running time type. Note: The input connection token must only represent a <i>CPC connection</i> or a <i>reset activation profile connection</i> .
28 (40) HWI_PRUNTIME	Requests to retrieve the processor running time. Note: The input connection token must only represent a <i>CPC connection</i> or a <i>reset activation profile connection</i> .
29 (41) HWI_PRUNTSEW	Requests to retrieve the processor running time slice end wait processing. Note: The input connection token must only represent a <i>CPC connection</i> or a <i>reset activation profile connection</i> . This attribute is only available when targeting a z13 GA2 or lower CPC.
2A (42) HWI_OOCINST	Requests to retrieve the on and off capacity on demand installed. Note: The input connection token must only represent a <i>CPC connection</i> .

Table 70. Valid query attribute identifiers (continued)	
Constant in hexadecimal (Decimal) Equate symbol	Description
2B (43) HWI_OOACT	Requests to retrieve the on and off capacity on demand currently activated. Note: The input connection token must only represent a <i>CPC connection</i> .
2C (44) HWI_OOCENAB	Requests to retrieve the on and off capacity on demand enabled. Note: The input connection token must only represent a <i>CPC connection</i> .
2D (45) HWI_OOCADT	Requests to retrieve the on and off capacity on demand activation date. Note: The input connection token must only represent a <i>CPC connection</i> .
2E (46) HWI_PCPCSWM	Requests to retrieve the permanent CPC software model. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
2F (47) HWI_PPBPMSW	Requests to retrieve the permanent plus billable processor software model. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
30 (48) HWI_PPTPSWM	Requests to retrieve the permanent plus (all) temporary processor software model. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
31 (49) HWI_PCPCMSU	Requests to retrieve the permanent CPC millions of service units (MSU) value. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
32 (50) HWI_PPBPMSU	Requests to retrieve the permanent plus billable processor MSU value. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
33 (51) HWI_PPTPMSU	Requests to retrieve the permanent plus (all) temporary processor MSU value. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
34 (52) HWI_NUMGPP	Requests to retrieve the number of general purpose processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
35 (53) HWI_NUMSAP	Requests to retrieve the number of service assist processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
36 (54) HWI_NUMIFAP	Requests to retrieve the number of the integrated facility for applications (IFA) processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
37 (55) HWI_NUMIFLP	Requests to retrieve the number of the integrated facility for Linux (IFL) processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
38 (56) HWI_NUMICFP	Requests to retrieve the number of the internal coupling facility (ICF) processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .

Table 70. Valid query attribute identifiers (continued)

Constant in hexadecimal (Decimal) Equate symbol	Description						
39 (57) HWI_NUMIIPP	Requests to retrieve the number of integrated information processors (IIP). This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .						
3A (58) HWI_NUMFLTP	Requests to retrieve the number of defective (faulty) processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .						
3B (59) HWI_NUMSPARE	Requests to retrieve the number of spare processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .						
3C (60) HWI_NUMPENDP	Requests to retrieve the number of pending (activation) processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .						
3D (61) HWI_CAPCHGALLWD	Requests to determine if activate/deactivate of capacity are permitted. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .						
3E (62) HWI_DGRSTAT	Requests to retrieve degraded status. Note: The input connection token must only represent a <i>CPC connection</i> .						
3F (63) HWI_CURRPOWERMODE	Requests to retrieve the current processor power savings mode active on the targeted CPC. This attribute is only available when targeting zEnterprise and higher CPC levels, up to and including the z14 level. Power saving capabilities are not supported on the z15. In addition, z13 and earlier levels require the Power saving feature which is only available if the Automate management enablement feature is installed. For more details about the power saving function, see <i>IBM Z Hardware Management Console Web Services API</i> . Note: The input connection token must only represent a <i>CPC connection</i> .						
40 (64) HWI_SUPPPPOWERMODE	Requests to retrieve the supported processor power savings modes available on the targeted CPC. This attribute is only available when targeting zEnterprise and higher CPC levels, up to and including the z14 level. Power saving capabilities are not supported on the z15. In addition, z13 and earlier levels require the Power saving feature which is only available if the Automate management enablement feature is installed. For more details about the power saving function, see <i>IBM Z Hardware Management Console Web Services API</i> . Non-REXX: The returned data is mapped as follows: <table> <tr> <td>Field Name</td> <td>Field Type</td> </tr> <tr> <td>Number of supported powersave modes</td> <td>32-bit integer</td> </tr> </table> For each supported powersave mode, the following is returned: <table> <tr> <td>Powersave mode</td> <td>32-bit integer value</td> </tr> </table> Note: The query parameter for this attribute must specify a data area large enough to contain all of the structure (that is, 32 bits + 32 bits per supported powersave mode returned). For example, if there are 2 supported powersave modes on the targeted CPC, then the structure must be at least 32 + (32 x 2) = 96 bits (12 bytes). REXX: The returned data is mapped as follows (where <i>x</i> is the user-defined QueryParm stem, <i>n</i> is the <i>n</i> -th requested attribute and <i>m</i> is the <i>m</i> -th returned powersave mode value): <ul style="list-style-type: none"> <i>x.n.ATTRIBUTEVALUE.0</i> is the number of supported powersave modes (<i>m</i>). <i>x.n.ATTRIBUTEVALUE.m.PSMODE</i> is the <i>m</i>-th powersave mode value. Note: The input connection token must only represent a <i>CPC connection</i> .	Field Name	Field Type	Number of supported powersave modes	32-bit integer	Powersave mode	32-bit integer value
Field Name	Field Type						
Number of supported powersave modes	32-bit integer						
Powersave mode	32-bit integer value						
41 (65) HWI_STPCONFIG	Requests to retrieve the Server Timer Protocol (STP) configuration data. Note: The input connect token must only represent a <i>CPC connection</i> .						


Table 70. Valid query attribute identifiers (continued)	
Constant in hexadecimal (Decimal) Equate symbol	Description
42 (66) HWI_NUMPGPP	Requests to retrieve the number of pending general purpose processors. Note: The input connect token must only represent a <i>CPC connection</i> .
43 (67) HWI_NUMPSAP	Requests to retrieve the number of pending service assist processors. Note: The input connect token must only represent a <i>CPC connection</i> .
44 (68) HWI_NUMPAAP	Requests to retrieve the number of pending Application Assist Processor (AAP) processors. Note: The input connect token must only represent a <i>CPC connection</i> .
45 (69) HWI_NUMPIFLP	Requests to retrieve the number of pending Integrated Facility for Linux (IFL) processors. Note: The input connect token must only represent a <i>CPC connection</i> .
46 (70) HWI_NUMPICFP	Requests to retrieve the number of pending Internal Coupling Facility (ICF) processors. Note: The input connect token must only represent a <i>CPC connection</i> .
47 (71) HWI_NUMPIIPP	Requests to retrieve the number of pending Integrated Information (IIP) processors. Note: The input connect token must only represent a <i>CPC connection</i> .
48 (72) HWI_POWERMODEALLOWED	Requests to retrieve the processor power savings mode allowed. This attribute is only available when zEnterprise and higher CPC levels, up to and including the z14 level. Power saving capabilities are not supported on the z15. In addition, z13 and earlier levels require the Power saving feature which is only available if the Automate management enablement feature is installed. For more details about the power saving function, see <i>IBM Z Hardware Management Console Web Services API</i> . HWMCA_TRUE The processor currently allows switching to power savings mode. HWMCA_FALSE The processor currently does not allow switching to power savings mode. Note: The input connection token must only represent a <i>CPC connection</i> .
49 (73) HWI_VERSION	Requests to retrieve the CPC version number. Note: The input connection token must only represent a <i>CPC connection</i> .
4A (74) HWI_EC_MCL_INFO	Requests to retrieve an XML string that describes the Engineering Change (EC) and Microcode Level (MCL) levels. Note: The input connection token must only represent a <i>CPC connection</i> .  Attention: The data returned by the support element can be quite large. Consider using a larger data area when requesting this attribute.

Table 70. Valid query attribute identifiers (continued)

Constant in hexadecimal (Decimal) Equate symbol	Description						
4B (75) HWI_LIST_IP_ADDRESSES	<p>Requests to retrieve all the IP addresses (in either IPv4 or IPv6 format, or both) used for the targeted CPC.</p> <p>Non-REXX: The returned data is mapped as follows:</p> <table border="1"> <thead> <tr> <th>Field Name</th><th>Field Type</th></tr> </thead> <tbody> <tr> <td>Number of IP addresses</td><td>32-bit unsigned integer</td></tr> <tr> <td>IP address value</td><td>39-character value padded with blanks</td></tr> </tbody> </table> <p>Note: The query parameter for this attribute must specify a data area large enough to contain all of the structure (that is, a 4-byte length field plus a 39-byte field for each IP address returned). For example, if there are 3 IP addresses returned, the AttributeValueLen specified for this attribute must be at least $(4 + (39 \times 3)) = 121$ bytes.</p> <p>REXX: The returned data is mapped as follows (where x is the user-defined QueryParm stem, n is the n-th requested attribute and m is the m-th returned IP address value):</p> <ul style="list-style-type: none"> $x.n.ATTRIBUTEVALUE.0$ is the number of IP addresses (m). $x.n.ATTRIBUTEVALUE.m.IPADDR$ is the m-th IP address value. <p>Note: The input connection token must only represent a <i>CPC connection</i>.</p>	Field Name	Field Type	Number of IP addresses	32-bit unsigned integer	IP address value	39-character value padded with blanks
Field Name	Field Type						
Number of IP addresses	32-bit unsigned integer						
IP address value	39-character value padded with blanks						
4C (76) HWI_AUTO_SWITCH_ENABLED	<p>Requests to retrieve a value used to determine if automatic switching between primary and alternate support elements is enabled.</p> <p>A 4-byte integer type value is returned:</p> <p>HWMCA_TRUE Automatic switching is enabled.</p> <p>HWMCA_FALSE Automatic switching is disabled.</p> <p>Note: The input connection token must only represent a <i>CPC connection</i>.</p>						
4D-68 (77-104) RESERVED	Reserved for CPC attributes unless otherwise noted.						
69 (105) HWI_CPCNAME	<p>Requests to retrieve the parent (CPC) name.</p> <p>Note: The input connection token must only represent an <i>image connection</i>.</p>						
6A (106) HWI_OSNAME	<p>Requests to retrieve the SW operating system name.</p> <p>The values returned on the HWI_OSNAME attribute are not owned by z/OS BCPII and are subject to change. The possible values returned by the various operating systems at the time of this publication include:</p> <p>HWI_OSTYPE value: MVS The HWI_OSNAME value returned is the SYSNAME parameter as defined in IEASYSxx parmlib member for the targeted image.</p> <p>HWI_OSTYPE value: VM The HWI_OSNAME value returned is the system identifier or system name as defined in the SYSTMID field in the SYSCM (System Common Area) control block.</p> <p>HWI_OSTYPE value: LINUX The HWI_OSNAME value returned is N/A.</p> <p>HWI_OSTYPE value: VSE The HWI_OSNAME value returned is the VSE system name.</p> <p>HWI_OSTYPE value: Z TPF EE The HWI_OSNAME value returned is the id value representing the targeted image's CPU designation in the z/TPF complex.</p> <p>HWI_OSTYPE value: CFCC The HWI_OSNAME value returned is the Coupling Facility name.</p> <p>HWI_OSTYPE value: SSC The HWI_OSNAME value returned is the system name defined in the Control Program Identification.</p> <p>Note: The input connection token must only represent an <i>image connection</i>.</p>						

Table 70. Valid query attribute identifiers (continued)

Constant in hexadecimal (Decimal) Equate symbol	Description
6B (107) HWI_OSTYPE	<p>Requests to retrieve the SW operating system type.</p> <p>The values returned on the HWI_OSTYPE attribute are not owned by z/OS BCPii and are subject to change. Possible values include MVS, VM, LINUX, VSE, and Z TPF EE, CFCC, and SSC (formerly zACI).</p> <p>Note: The input connection token must only represent an <i>image connection</i>.</p>
6C (108) HWI_OSLEVEL	<p>Requests to retrieve the SW operating system level.</p> <p>The values returned on the HWI_OSLEVEL attribute are not owned by z/OS BCPii and are subject to change. The possible values returned by the various operating systems at the time of this publication include:</p> <p>HWI_OSTYPE value: MVS The HWI_OSLEVEL value is mapped by the CVTOSLVL field of the CVT control block.</p> <p>HWI_OSTYPE value: VM The HWI_OSLEVEL value is mapped as follows:</p> <ul style="list-style-type: none"> • 4-bit release # • 4-bit modification level • 8-bit version # • 16-bit service level • 8-bit MVS guest count • 8-bit LINUX guest count • 8-bit VSE guest count • 8-bit Solaris guest count <p>HWI_OSTYPE value: LINUX The HWI_OSLEVEL value is mapped as follows, in hexadecimal:</p> <ul style="list-style-type: none"> • 40 bits N/A • 8-bit major kernel revision • 8-bit major release • 8-bit minor release <p>HWI_OSTYPE value: VSE The HWI_OSLEVEL value is mapped as follows:</p> <ul style="list-style-type: none"> • 32-bit VSE/AF release level • 32-bit latest service level (if available) <p>HWI_OSTYPE value: Z TPF EE The HWI_OSLEVEL value is mapped as follows:</p> <ul style="list-style-type: none"> • 16-bit version # • 8-bit PUT level <p>HWI_OSTYPE value: CFCC The HWI_OSLEVEL value is mapped as follows:</p> <ul style="list-style-type: none"> • 2 byte release level • 2 byte service level • 1 byte dynamic dispatch setting • remaining 3 bytes unused

Table 70. Valid query attribute identifiers (continued)

Constant in hexadecimal (Decimal) Equate symbol	Description
6C (108) HWI_OSLEVEL <i>continued</i>	<p>HWI_OSTYPE value: SSC</p> <p>The HWI_OSLEVEL is an 8 byte hexadecimal system-level in this format: <code>0x<ab><cc><dd><eeee><ff><gg><hh></code> where:</p> <ul style="list-style-type: none"> • <code><a></code> 1 byte, Bit 0 indicates hypervisor use. • <code></code> 1 digit that indicates the distribution as follows: <ul style="list-style-type: none"> – 0 Generic Linux – 1 Red Hat Enterprise Linux – 2 SUSE Linux Enterprise – 3 Canonical Ubuntu – 4 Fedora – 5 openSUSE Leap – 6 Debian GNU/Linux – 7 Red Hat Enterprise Linux CoreOS • <code><cc></code> 2 digits for a distribution-specific encoding of the major version of the distribution. • <code><dd></code> 2 digits for a distribution-specific encoding of the minor version of the distribution. • <code><eeee></code> 4 digits for the patch level of the distribution. • <code><ff></code> 2 digits for the major version of the kernel. • <code><gg></code> 2 digits for the minor version of the kernel. • <code><hh></code> 2 digits for the stable version of the kernel.
Examples of 6C (108) HWI_OSLEVEL <i>continued</i>	<p>Examples:</p> <p>For MVS, FFFFFFFFEF7F0000 implies that the target is running z/OS V1R13 because the CVTZOS_V1R13 bit is the last supported release flag that is on.</p> <p>For VM, 4005100200320000 implies that the target is running z/VM Release 4, Modification Level 0, Version 5, Service Level 1002, MVS guest count 0, Linux guest count 32, VSE guest count 0, and Solaris guest count 0.</p> <p>For LINUX, 0000000000020620 implies that the target is running z/LINUX major kernel revision 2, major release 6, and minor release 32.</p> <p>For VSE, 0830000000000000 implies that the target is running at the VSE/AF 8.3 release level and no service level is available.</p> <p>For Z TPF EE, 0101070000000000 implies that the target is running z/TPF version 1.1, PUT level 7.</p> <p>Note: The input connection token must represent an <i>image connection</i>.</p>
6D (109) HWI_SYSPLEX	<p>Requests to retrieve the SW sysplex name (z/OS only).</p> <p>Note: The input connection token must only represent an <i>image connection</i>.</p>
6E (110) HWI_CLUSTER	<p>Requests to retrieve the LPAR cluster name.</p> <p>Note: The input connection token must only represent an <i>image connection</i>.</p>
6F (111) HWI_PARTITIONID	<p>Requests to retrieve the partition ID. If the connection token represents an <i>image connection</i>, the image partition ID is returned; if the connection token represents an <i>image activation profile connection</i>, the image activation profile partition ID is returned. The image partition ID is only retrievable when the partition has been activated.</p> <p>Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i>.</p>
70 (112) HWI_DEFCAP	<p>Requests to retrieve the current defined capacity.</p> <p>Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i>.</p>
71 (113) HWI_SGPIPW	<p>Requests to retrieve the shared general processor initial processing weight (SGPIPW).</p> <p>Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i>.</p>
72 (114) HWI_SGPIPWCAP	<p>Requests to retrieve the SGPIPW to be capped or not capped.</p> <p>Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i>.</p>

Table 70. Valid query attribute identifiers (continued)	
Constant in hexadecimal (Decimal) Equate symbol	Description
73 (115) HWI_SGPPWMIN	Requests to retrieve the minimum SGPPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
74 (116) HWI_SGPPWMAX	Requests to retrieve the maximum SGPPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
75 (117) HWI_SGPPW	Requests to retrieve the current SGPPW value. Note: The input connection token must only represent an <i>image connection</i> .
76 (118) HWI_SGPPWCAP	Requests to retrieve the SGPPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> .
77 (119) HWI_WLM	Requests to retrieve whether WLM is allowed to change processing weight-related attributes. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
78 (120) HWI_IFAIPW	Requests to retrieve the integrated facility for applications initial processing weight (IFAIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
79 (121) HWI_IFAIPWCAP	Requests to retrieve the IFAIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7A (122) HWI_IFAPWMIN	Requests to retrieve the minimum IFAPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7B (123) HWI_IFAPWMAX	Requests to retrieve the maximum IFAPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7C (124) HWI_IFAPW	Requests to retrieve the current IFAPW value. Note: The input connection token must only represent an <i>image connection</i> .
7D (125) HWI_IFAPWCAP	Requests to retrieve the IFAPW to be currently capped or not capped. Note: The input connection token must only represent an <i>image connection</i> .
7E (126) HWI_IFLIPW	Requests to retrieve the integrated facility for Linux initial processing weight. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7F (127) HWI_IFLIPWCAP	Requests to retrieve the IFLIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
80 (128) HWI_IFLPWMIN	Requests to retrieve the minimum IFLPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .

Table 70. Valid query attribute identifiers (continued)

Constant in hexadecimal (Decimal) Equate symbol	Description
81 (129) HWI_IFLPWMAX	Requests to retrieve the maximum IFLPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
82 (130) HWI_IFLPW	Requests to retrieve current IFLPW value. Note: The input connection token must only represent an <i>image connection</i> .
83 (131) HWI_IFLPWCAP	Requests to retrieve the IFLPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> .
84 (132) HWI_ICFIPW	Requests to retrieve the internal coupling facility initial processing weight (ICFIPW). Note: The input connection token must only represent an <i>image connection</i> (Coupling Facility images only) or an <i>image activation profile connection</i> .
85 (133) HWI_ICFIPWCAP	Requests to retrieve the ICFIPW be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> (Coupling Facility images only) or an <i>image activation profile connection</i> .
86 (134) HWI_ICFPWMIN	Requests to retrieve the minimum ICFPW value. Note: The input connection token must only represent an <i>image connection</i> (Coupling Facility images only) or an <i>image activation profile connection</i> .
87 (135) HWI_ICFPWMAX	Requests to retrieve the maximum ICFPW value. Note: The input connection token must only represent an <i>image connection</i> (Coupling Facility images only) or an <i>image activation profile connection</i> .
88 (136) HWI_ICFPW	Requests to retrieve the current ICFPW value. Note: The input connection token must only represent an <i>image connection</i> (Coupling Facility images only).
89 (137) HWI_ICFPWCAP	Requests to retrieve the ICFPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> (Coupling Facility images only).
8A (138) HWI_IIIPW	Requests to retrieve the integrated information processors initial processing weight (IIIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8B (139) HWI_IIIPWCAP	Requests to retrieve the IIIPW be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8C (140) HWI_IIPWMIN	Requests to retrieve the minimum IIPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8D (141) HWI_IIPWMAX	Requests to retrieve the maximum IIPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8E (142) HWI_IIPW	Requests to retrieve the current IIPW value. Note: The input connection token must only represent an <i>image connection</i> .

Table 70. Valid query attribute identifiers (continued)											
Constant in hexadecimal (Decimal) Equate symbol	Description										
8F (143) HWI_IIPWCAP	Requests to retrieve the IIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> .										
90 (144) HWI_IPLTOKEN	Requests to retrieve the IPL token associated with the current IPL of the image targeted. Note: The input connection token must only represent an <i>image connection</i> .										
91 (145) HWI_PSW	Requests to retrieve the program status word (PSW) for each of the central processors (CP) associated with this image. Non-REXX: The returned data is mapped as follows: <table border="1"> <thead> <tr> <th>Field Name</th><th>Field Type</th></tr> </thead> <tbody> <tr> <td>Number of CPs</td><td>32-bit unsigned integer</td></tr> <tr> <td colspan="2">For each CP, the following is returned:</td></tr> <tr> <td>CPUID</td><td>32-bit unsigned integer</td></tr> <tr> <td>PSW</td><td>128-bit unsigned integer</td></tr> </tbody> </table> <p>Note: The query parameter for this attribute must specify a data area large enough to contain all of the above structure (that is 32 bits + 160 bits per CP). For example, if there are 4 CPs on the targeted image, the AttributeValueLen specified for this attribute must be 32 + (160 x 4) = 672 bits (84 bytes).</p> <p>REXX: The returned data is mapped as follows (where <i>x</i> is the user-defined QueryParm stem, <i>n</i> is the <i>n</i>-th requested attribute and <i>m</i> is the <i>m</i>-th returned CPUID or PSW value):</p> <ul style="list-style-type: none"> • x.n.ATTRIBUTEVALUE.0 is the number of CPs (<i>m</i>). • x.n.ATTRIBUTEVALUE.m.CPUID is the <i>m</i>-th CPUID value. • x.n.ATTRIBUTEVALUE.m.PSW is the <i>m</i>-th PSW value. <p>Note: The input connection token must represent an <i>image connection</i>.</p>	Field Name	Field Type	Number of CPs	32-bit unsigned integer	For each CP, the following is returned:		CPUID	32-bit unsigned integer	PSW	128-bit unsigned integer
Field Name	Field Type										
Number of CPs	32-bit unsigned integer										
For each CP, the following is returned:											
CPUID	32-bit unsigned integer										
PSW	128-bit unsigned integer										
92 (146) HWI_GROUP_PROFILE_CAPACITY	<ul style="list-style-type: none"> • For group profile connection (the input ConnectToken represents a group profile connection), requests to retrieve the workload unit capacity for a group profile. • For LPAR Capacity group connection (the input ConnectToken represents an LPAR Capacity group connection), requests to retrieve the dynamic workload unit capacity for an LPAR Capacity group. This attribute for this connection type is supported on z14 GA2 or higher CPC. • For image connection (the input ConnectToken represents an image connection), requests to retrieve the dynamic workload unit capacity for a group of images in which the target image is a member. This attribute for this connection type requires that the target image be on a z196 (zEnterprise) or higher CPC and is a member of an LPAR Capacity group. If these requirements are not met, the HWI_QUERY_ATTRIBUTE_NOT_SUPPORTED '406'x return code will be returned. 										
93 (147) HWI_LAST_USED_LOADADDR	Requests to retrieve the load address that was used when the image was last loaded. This attribute is only available when targeting an image residing on a z196 or later CPC. Note: The input connection token must represent an <i>image connection</i> .										
94 (148) HWI_LAST_USED_LOADPARAM	Requests to retrieve the load parameters that were used when the image was last loaded. This attribute is only available when targeting an image residing on a z196 or later CPC. Note: The input connection token must represent an <i>image connection</i> .										
95 (149) HWI_ABSCAP	Request to retrieve whether absolute capping is enabled for general purpose processors. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.										
96 (150) HWI_ABSCAPVAL	Requests to retrieve the maximum general purpose processor consumption for the target image. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.										
97 (151) HWI_IFAABSCAP	Requests to retrieve whether absolute capping is enabled for AAP processors. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.										

Table 70. Valid query attribute identifiers (continued)

Constant in hexadecimal (Decimal) Equate symbol	Description
98 (152) HWI_IFAABSCAPVAL	Requests to retrieve the maximum AAP processor consumption for the target image. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.
99 (153) HWI_IFLABSCAP	Requests to retrieve whether absolute capping is enabled for IFL processors. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.
9A (154) HWI_IFLABSCAPVAL	Requests to retrieve the maximum IFL processor consumption for the target image. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.
9B (155) HWI_ICFABSCAP	Requests to retrieve whether absolute capping is enabled for IFC processors. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.
9C (156) HWI_ICFABSCAPVAL	Requests to retrieve the maximum IFC processor consumption for the target image. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.
9D (157) HWI_IIPABSCAP	Requests to retrieve whether absolute capping is enabled for IIP processors. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.
9E (158) HWI_IIPABSCAPVAL	Requests to retrieve the maximum IIP processor consumption for the target image. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.
9F (159) HWI_GROUP_PROF_ABSCAP	<ul style="list-style-type: none"> For group profile connection (the input ConnectToken represents a group profile connection), requests to retrieve whether absolute capping is enabled for general purpose processors for a group profile. For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to retrieve whether the dynamic absolute capping is enabled for general purpose processors for an LPAR Capacity group. This attribute is only available when targeting a z14 GA2 or higher CPC. For image connection (the input ConnectToken represents an image connection), requests to retrieve whether the dynamic absolute capping is enabled for general purpose processors for a group of images in which the target image is a member. This attribute is only available when targeting a z13 GA2 or higher CPC.
A0 (160) HWI_GROUP_PROF_ABSCAPVAL	<ul style="list-style-type: none"> For group profile connection (the input ConnectToken represents a group profile connection), requests to retrieve the maximum general-purpose processor consumption for a group profile. For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to retrieve the dynamic maximum general-purpose processor consumption for an LPAR Capacity group. This attribute is only available when targeting a z14 GA2 or higher CPC. For image connection (the input ConnectToken represents an image connection), requests to retrieve the dynamic maximum general-purpose processor consumption for a group of images in which the target image is a member. This attribute is only available when targeting a z13 GA2 or higher CPC.
A1 (161) HWI_GROUP_PROF_ICFABSCAP	<ul style="list-style-type: none"> For group profile connection (the input ConnectToken represents a group profile connection), requests to retrieve whether the absolute capping is enabled for ICF processors for a group profile. For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to retrieve whether the dynamic absolute capping is enabled for ICF processors for an LPAR Capacity group. This attribute is only available when targeting a z14 GA2 or higher CPC. For image connection (the input ConnectToken represents an image connection), requests to retrieve whether the dynamic absolute capping is enabled for ICF processors for a group of images in which the target image is a member. This attribute is only available when targeting a z13 GA2 or higher CPC.
A2 (162) HWI_GROUP_PROF_ICFABSCAPVAL	<ul style="list-style-type: none"> For group profile connection (the input ConnectToken represents a group profile connection), requests to retrieve the maximum ICF processor consumption for a group profile. For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to retrieve the dynamic maximum ICF processor consumption for an LPAR Capacity group. This attribute is only available when targeting a z14 GA2 or higher CPC. For image connection (the input ConnectToken represents an image connection), requests to retrieve the dynamic maximum ICF processor consumption for a group of images in which the target image is a member. This attribute is only available when targeting a z13 GA2 or higher CPC.

Table 70. Valid query attribute identifiers (continued)	
Constant in hexadecimal (Decimal) Equate symbol	Description
A3 (163) HWI_GROUP_PROF_IFLABSC AP	<ul style="list-style-type: none"> For group profile connection (the input ConnectToken represents a group profile connection), requests to retrieve whether the absolute capping is enabled for IFL processors for a group profile. For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to retrieve whether the dynamic absolute capping is enabled for IFL processors for an LPAR Capacity group. This attribute is only available when targeting a z14 GA2 or higher CPC. For image connection (the input ConnectToken represents an image connection), requests to retrieve whether the dynamic absolute capping is enabled for IFL processors for a group of images in which the target image is a member. This attribute is only available when targeting a z13 GA2 or higher CPC.
A4 (164) HWI_GROUP_PROF_IFLABSC APVAL	<ul style="list-style-type: none"> For group profile connection (the input ConnectToken represents a group profile connection), requests to retrieve the maximum IFL processor consumption for a group profile. For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to retrieve the maximum IFL processor consumption for an LPAR Capacity group. This attribute is only available when targeting a z14 GA2 or higher CPC. For image connection (the input ConnectToken represents an image connection), requests to retrieve the dynamic maximum IFL processor consumption for a group of images in which the target image is a member. This attribute is only available when targeting a z13 GA2 or higher CPC.
A5 (165) HWI_GROUP_PROF_IIPABSC AP	<ul style="list-style-type: none"> For group profile connection (the input ConnectToken represents a group profile connection), requests to retrieve whether the absolute capping is enabled for IIP processors for a group profile. For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to retrieve whether the dynamic absolute capping is enabled for IIP processors for an LPAR Capacity group. This attribute is only available when targeting a z14 GA2 or higher CPC. For image connection (the input ConnectToken represents an image connection), requests to retrieve whether the dynamic absolute capping is enabled for IIP processors for a group of images in which the target image is a member. This attribute is only available when targeting a z13 GA2 or higher CPC.
A6 (166) HWI_GROUP_PROF_IIPABSC APVAL	<ul style="list-style-type: none"> For group profile connection (the input ConnectToken represents a group profile connection), requests to retrieve the maximum IIP processor consumption for a group profile. For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to retrieve the dynamic maximum IIP processor consumption for an LPAR Capacity group. This attribute is only available when targeting a z14 GA2 or higher CPC. For image connection (the input ConnectToken represents an image connection), requests to retrieve the dynamic maximum IIP processor consumption for a group of images in which the target image is a member. This attribute is only available when targeting a z13 GA2 or higher CPC.
A7-B6 (167-182) RESERVED	Additional attributes and reserved numbers for attributes that are for <i>image connections</i> only.
B7 (183) HWI_RECID	Requests to retrieve the record ID. Note: The input connection token must only represent a <i>capacity record connection</i> .
B8 (184) HWI_RECTYPE	Requests to retrieve the record type. Note: The input connection token must only represent a <i>capacity record connection</i> .
B9 (185) HWI_ACTSTAT	Requests to retrieve the record activation status. Note: The input connection token must only represent a <i>capacity record connection</i> .
BA (186) HWI_ACTDATE	Requests to retrieve the record activation date. Note: The input connection token must only represent a <i>capacity record connection</i> .
BB (187) HWI_EXPDATE	Requests to retrieve the record expiration date. Note: The input connection token must only represent a <i>capacity record connection</i> .
BC (188) HWI_ACTEXP	Requests to retrieve the record activation expiration date. Note: The input connection token must only represent a <i>capacity record connection</i> .

Table 70. Valid query attribute identifiers (continued)	
Constant in hexadecimal (Decimal) Equate symbol	Description
BD (189) HWI_MAXRADS	Requests to retrieve the maximum real activation days. Note: The input connection token must only represent a <i>capacity record connection</i> .
BE (190) HWI_MAXTADS	Requests to retrieve the maximum test activation days. Note: The input connection token must only represent a <i>capacity record connection</i> .
BF (191) HWI_REMRADS	Requests to retrieve the remaining real activation days. Note: The input connection token must only represent a <i>capacity record connection</i> .
C0 (192) HWI_REMTADS	Requests to retrieve the remaining test activation days. Note: The input connection token must only represent a <i>capacity record connection</i> .
C1 (193) HWI_OOCODREC	Request to retrieve all aspects of a capacity record in XML format. Note: The input connection token must only represent a <i>capacity record connection</i> .
C3-C8 (195-200) RESERVED	Reserved for capacity record attributes.
C9 (201) HWI_IOCDS	Requests to retrieve the IOCDS. Note: The input connection token must represent a <i>reset activation profile</i> .
CA (202) HWI_IPL_ADDRESS	Requests to retrieve the IPL address. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CB (203) HWI_IPL_PARM	Requests to retrieve the IPL parameter. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CC (204) HWI_IPL_TYPE	Requests to retrieve the IPL type for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CD (205) HWI_VW_PORTNAME	Requests to retrieve the worldwide port name for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CE (206) HWI_BOOT_PGM_SELECTOR	Requests to retrieve the boot program selector for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CF (207) HWI_LU_NUM	Requests to retrieve the logical unit number value for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
D0 (208) HWI_BOOTREC_BLK_ADDR	Requests to retrieve the boot record logical block address for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .

Table 70. Valid query attribute identifiers (continued)

Constant in hexadecimal (Decimal) Equate symbol	Description
D1 (209) HWI_OPSYS_LOADPARAM	Requests to retrieve the operating system specific load parameter. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
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 or image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
DC (220) HWI_NUM_IFL	Requests to retrieve the number of dedicated integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .

Table 70. Valid query attribute identifiers (continued)

Constant in hexadecimal (Decimal) Equate symbol	Description
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 connection token must represent an <i>image activation profile</i> .
E0 (224) HWI_NUM_ZIIP	Requests to retrieve the number of dedicated z System Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E1 (225) HWI_NUM_RESZIIP	Requests to retrieve the number of reserved dedicated z System Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
E3 (227) HWI_NUM_RES_SHARED_GPP	Requests to retrieve the number of reserved shared general purpose processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E4 (228) HWI_NUM_SHARED_IFA	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. Note: The input connection token must represent an <i>image activation profile</i> .
E5 (229) HWI_NUM_RES_SHARED_IFA	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. Note: The input connection token must represent an <i>image activation profile</i> .
E6 (230) HWI_NUM_SHARED_IFL	Requests to retrieve the number of shared integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E7 (231) HWI_NUM_RES_SHARED_IFL	Requests to retrieve 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 connection token must represent an <i>image activation profile</i> .
E8 (232) HWI_NUM_SHARED_ICF	Requests to retrieve the number of shared internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E9 (233) HWI_NUM_RES_SHARED_ICF	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. Note: The input connection token must represent an <i>image activation profile</i> .
EA (234) HWI_NUM_SHARED_ZIIP	Requests to retrieve the number of shared z System Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
EB (235) HWI_NUM_RES_SHARED_ZIIP	Requests to retrieve the number of reserved shared z System Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .

Table 70. Valid query attribute identifiers (continued)

Constant in hexadecimal (Decimal) Equate symbol	Description
EC (236) HWI_BASIC_CPU_AUTH _COUNT_CNTL	Requests to retrieve the enablement value of the Basic CPU counter facility for the CPC image. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
ED (237) HWI_PROBSTATE_CPU_AUTH _COUNT_CNTL	Requests to retrieve the enablement value of the Problem state CPU counter facility for the CPC image. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
EE (238) HWI_CRYPTACTIVITY_CPU _AUTH_COUNT_CNTL	Requests to retrieve the enablement value of the crypto activity CPU counter facility for the CPC image. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
EF (239) HWI_EXTENDED_CPU_AUTH _COUNT_CNTL	Requests to retrieve the enablement value of the extended CPU counter facility for the CPC image. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F0 (240) HWI_COPROCESSOR_CPU _AUTH_COUNT_CNTL	Requests to retrieve the enablement value of the coprocessor group CPU counter facility for the CPC image. This attribute is only available when targeting a z10 or z196 CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F1 (241) HWI_BASIC_CPU_SAMPLING _AUTH_CNTL	Requests to retrieve the enablement value of the basic CP CPU sampling facility for the CPC image. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F2 (242) HWI_APROF_STORE_STATUS	Requests to retrieve the store status function value. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent a <i>load activation profile</i> .
F3 (243) HWI_APROF_LOADTYPE	Requests to retrieve the type of load being requested. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent a <i>load activation profile</i> .
F4 (244) HWI_PROFILE_DESCRIPTION	Requests to retrieve the activation profile description. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> , <i>reset activation profile</i> , <i>load activation profile</i> or <i>group profile</i> .
F5 (245) HWI_PROFILE_PARTITION _ID	Requests to retrieve the partition identifier for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F6 (246) HWI_OPERATING_MODE	Requests to retrieve the operating mode value for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F7 (247) HWI_CLOCK_TYPE	Requests to retrieve the clock type assignment (time source setting) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .

Table 70. Valid query attribute identifiers (continued)

Constant in hexadecimal (Decimal) Equate symbol	Description
F8 (248) HWI_TIME_OFFSET_DAYS	Requests to retrieve the time offset days (the number of days currently set as the offset from the external time source's time of day) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F9 (249) HWI_TIME_OFFSET_HOURS	Requests to retrieve the time offset hours (the number of hours currently set as the offset from the external time source's time of day) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FA (250) HWI_TIME_OFFSET_MINUTES	Requests to retrieve the time offset minutes (the number of minutes currently set as the offset from the external time source's time of day) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FB (251) HWI_TIME_OFFSET_INCREASE	Requests to retrieve the time offset increase or decrease value for the activation profile. The time offset, as specified in days, hours, and minutes, is increased or decreased from GMT. TRUE means that the time offset is east of GMT. FALSE means that the time offset is west of GMT. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FC (252) HWI_LICCC_VALIDATION_ENABLED	Requests to retrieve whether the activation profile must conform to the current Licensed Internal Code Configuration Control (LICCC) configuration. This attribute is only available when targeting a zEnterprise or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FD (253) HWI_GLOBAL_PERFORMANCE_DATA_CONTROL	Requests to retrieve whether the logical partition can be used to view the processing unit activity data for all other LPARs activated on the same CPC. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FE (254) HWI_IO_CONFIGURATION_CONTROL	Requests to retrieve whether the logical partition can be used to read and write any Input/Output Configuration Data Set (IOCDs) in the configuration. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FF (255) HWI_CROSS_PARTITION_AUTHORITY	Requests to retrieve whether the logical partition can be used to issue control program instructions that reset or deactivate other LPARs. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
100 (256) HWI_LOGICAL_PARTITION_ISOLATION	Requests to retrieve whether reconfigurable channel paths assigned to the logical partition are reserved for its exclusive use. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
101-109 (257–265) RESERVED	Reserved for activation profile attributes.

NumOfAttributes (non-REXX)

Supplied parameter

- Type: Integer
- Length: 4 bytes

NumOfAttributes specifies the number of attributes to be queried. The maximum number of attributes allowed is 64.

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

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.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code that is returned from the console application API or the BCPii transport layer.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, “BCPii communication error reason codes,” on page 819 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0006yyyy' because of one of the following reasons:

<i>Table 71. Reasons for abend X'042', RC X'0006yyyy'</i>	
yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See [z/OS MVS System Codes](#) for additional information.

Return codes

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

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
0 HWI_OK	0 HWI_OK	Meaning: Successful completion. Action: None.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
100 HWI_CONNECT_TOKEN_INV	256 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified connect token is not valid. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • 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. <p>Action: Check for probable coding error.</p>
101 HWI_COMMUNICATION_ERROR	257 HWI_COMMUNICATION_ERROR	<p>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.</p> <p>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. In some cases, the Diag_Index and Diag_Key may contain additional details.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, “BCPii communication error reason codes,” on page 819.</p>
102 HWI_DIAGAREA_INV	258 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
103 HWI_CONNECT_TOKEN_INACTIVE	259 HWI_CONNECT_TOKEN_INACTIVE	<p>Meaning: The specified connect token is no longer valid. The connection has been disconnected or it is in the progress of being disconnected.</p> <p>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.</p>
104 HWI_TARGET_CPC_CHANGED	260 HWI_TARGET_CPC_CHANGED	<p>Meaning: The CPC name represented by the specified token is valid but does not represent the same physical machine that was targeted by the initial HWICONN call. All connections that were established prior to the name change can no longer be used.</p> <p>Action: The application should cease using this connect token. If the application intends to target the CPC using the name represented by the specified connect token, it must first reconnect to the CPC before issuing any BCPii service call.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
401 HWI_QUERYPARM_ATTRIB_INV	1025 HWI_QUERYPARM_ATTRIB_INV	<p>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:</p> <ul style="list-style-type: none"> • 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. <p>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.</p> <p>See the DiagArea for further diagnostic information:</p> <ul style="list-style-type: none"> • 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.”

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
402 HWI_QUERYPARM_INACCESSIBLE	1026 HWI_QUERYPARM_INACCESSIBLE	<p>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.</p> <p>Action: Check for probable coding error. Consider the following possibilities:</p> <ul style="list-style-type: none"> • 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_ATTRIBUTOR_INACCESSIBLE	1027 HWI_QUERYPARM_ATTRIBUTOR_INACCESSIBLE	<p>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.</p> <p>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.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
404 HWI_QUERYPARM_ATTRIB_LENGTH_INV	1028 HWI_QUERYPARM_ATTRIB_LENGTH_INV	<p>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.</p> <p>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.</p>
405 HWI_QUERY_NUMOFATTRIB_INV	1029 HWI_QUERY_NUMOFATTRIB_INV	<p>Meaning: Program error. The NumOfAttributes specified on the call is not valid. The NumOfAttributes value must be in the range of 1 to 64.</p> <p>Action: Check for probable error. Verify that the NumOfAttributes specified is greater than zero and less than or equal to 64.</p>
406 HWI_QUERY_ATTRIBUTE_NOT_SUPPORTED	1030 HWI_QUERY_ATTRIBUTE_NOT_SUPPORTED	<p>Meaning: The targeted hardware of the HWIQUERY request does not recognize the attribute attempted to be retrieved.</p> <p>Action: Verify that the targeted hardware is at a level that supports the type of attribute being queried.</p>
407 HWI_QUERY_TARGET_DEACTIVATED	1031 HWI_QUERY_TARGET_DEACTIVATED	<p>Meaning: A query attribute could not be retrieved because the targeted object is deactivated.</p> <p>Action: Verify that the targeted object is activated. Activate the object before attempting to retrieve this same attribute again.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
408 HWI_QUERY_ATTRIB_TEMP_NOT_AVAILABLE	1032 HWI_QUERY_ATTRIB_TEMP_NOT_AVAILABLE	<p>Meaning: One or more query attributes could not be retrieved because the support element (SE) is temporarily unavailable.</p> <p>Action: Try this request again at a later time. If the problem persists, contact the IBM Support Center.</p>
409 HWI_QUERY_ATTRIB_NOT_AVAILABLE	1033 HWI_QUERY_ATTRIB_NOT_AVAILABLE	<p>Meaning: One or more query attributes could not be retrieved due to one of the following:</p> <ol style="list-style-type: none"> 1. If retrieving one of the absolute capping value attributes, its absolute capping type is not currently enabled. <p>For example, if the HWI_ABSCAPVAL attribute is requested, the HWI_ABSCAP attribute must be set to HWMCA_TRUE.</p> <ol style="list-style-type: none"> 2. The returned value from the Support Element for one of the requested attributes is not valid or in the expected value type. The Diag_Text in the DiagArea may contain additional diagnostic information. <p>Action: If an absolute capping value is requested, verify that its absolute capping type is enabled. If the absolute capping type is currently enabled, search the problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
F00 HWI_NOT_AVAILABLE	3840 HWI_NOT_AVAILABLE	<p>Meaning: BCPii services are not available, and the system rejects the service request.</p> <p>Action: Notify the system programmer to start the BCPii address space and try the request again. See “Restarting the HWIBCPii address space” on page 261 about how to start the BCPii address space.</p> <p>Programs can also listen to ENF68 to determine when BCPii services are available. See <i>z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG</i> for how to listen for BCPii activation messages.</p>
F01 HWI_AUTH_FAILURE	3841 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWI_NO_SAF_AUTH	3842 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for CPC, activation profile, user-defined image group connections, group profile or LPAR Capacity group connection. • Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagenam e 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.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
F03 HWI_INTERRUPT_STATUS_INV	3843 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	3844 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	3845 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	3846 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.
F07 HWI_UNSUPPORTED_ENVIRONME NT	3847 HWI_UNSUPPORTED_ENVIRONME NT	Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine). Action: Issue the BCPii service from a different execution environment.
FFF HWI_UNEXPECTED_ERROR	4095 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).AttributeIdentifier = HWI_NAME;
QueryParm(1).AttributeValue_Ptr = Addr(Value1);
QueryParm(1).AttributeValueLen = length of value1;
QueryParm(2).AttributeIdentifier = HWI_OPERSTAT;
QueryParm(2).AttributeValue_Ptr = Addr(Value2);
QueryParm(2).AttributeValueLen = 4;
CALL HWIQUERY (ReturnCode, ConnectToken, QueryParm_Ptr,
              NumOfAttributes, DiagArea)
.
.

```

A REXX programming example for the HWIQUERY service:

```

myQueryParm.0 = 4 /* Set number of attributes */
myQueryParm.n.ATTRIBUTEIDENTIFIER = HWI_NAME
myQueryParm.n.ATTRIBUTEIDENTIFIER = HWI_LUAPROF
myQueryParm.n.ATTRIBUTEIDENTIFIER = HWI_MSERIAL
myQueryParm.n.ATTRIBUTEIDENTIFIER = HWI_IPADDR

address bcp11 "hwiquery RetCode myConnectToken myQueryParm. myDiag."

If (RC <> 0) | (Retcode <> 0) Then
  Do
    Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
    If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
      Do
        Say ' Diag_index=' myDiag.DIAG_INDEX
        Say ' Diag_key=' myDiag.DIAG_KEY
        Say ' Diag_actual=' myDiag.DIAG_ACTUAL
        Say ' Diag_expected=' myDiag.DIAG_EXPECTED
        Say ' Diag_comerr=' myDiag.DIAG_COMMERR
        Say ' Diag_text=' myDiag.DIAG_TEXT
      End
    End
  Else
    Do n=1 to myQueryParm.0
      Say ' myQueryParm.'n'.ATTRIBUTEVALUE = 'myQueryParm.n.ATTRIBUTEVALUE
    End

```

HWIREST — Issue RESTlike requests to the SE

Call the HWIREST service to issue REST API operations. The requests will continue to be sent as SCLP packets over the current internal transport network to the local support element (SE).

Description

Environment

The requirements for the callers are:

Requirement	Details
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

Requirement**Linkage:****Console setup:****Details**

Standard MVS linkage conventions are used

The main console on the HMC must be activated in order for the operating system commands to be sent successfully. To activate the main console, use the vary command: `v cn(*),activate`.

Programming requirements

An application using this service is expected to begin by issuing the List CPC Objects or the List Targetable HMCs operation. (See Note below.) The List CPC Objects operation obtains the URI and targeting information for the CPC(s). The List Targetable HMCs operation lists the HMCs that can potentially be targeted from the local system for HWIREST requests.

Note: List CPC Objects and List Targetable HMCs are the only REST API operations that do not require targeting information and will always be directed to the local SE.

The response for this request will contain an array of CPC or HMC objects. Each CPC object will include its corresponding URI, value of the (object-uri) property, and targeting information, value of the (target-name) property. All subsequent HWIREST requests will build or re-use that information when interacting with one or more of the specific CPC's.

Each HMC Object will contain its target-name and other relevant information regarding its current configurations and ability to be targeted. This includes whether Web Services is enabled and if a BCpii authorization record exists on the HMC. Both of these conditions must be true for the HMC to be a valid target. Note the HMC name must begin with *HMC://* when used to target HMCs with supporting URIs.

If your application is only interested in interacting with the LOCAL CPC, it will issue List CPC Objects and search the response body for the CPC entity that contains "local" as the value for the "location" property (see first step in [Figure 25 on page 379](#)). Alternatively, if your application knows the name of the CPC, it can issue List CPC Objects and take advantage of the supported query parm to filter the response to only return CPC's with a specific name pattern. (see first step in [Figure 26 on page 380](#)).

Similar List operations are available for other types of resources—LPARs, Capacity Records, Activations Profiles—that your application may want to interact with.

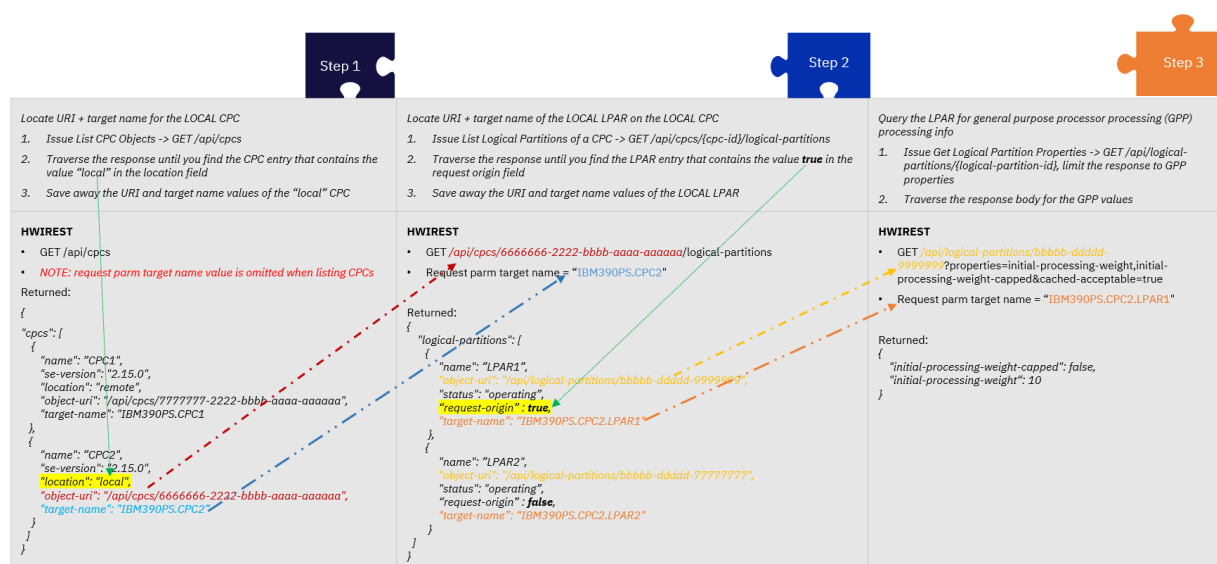


Figure 25. Retrieve LPAR GPP weight for the LOCAL LPAR

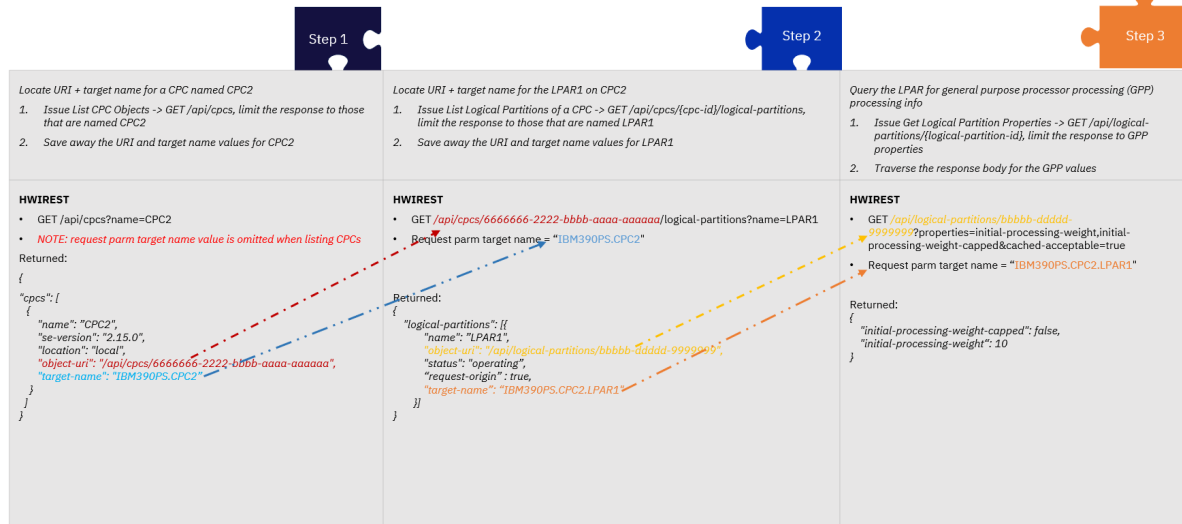


Figure 26. Retrieve LPAR GPP weight when the CPC and LPAR name are known

In the case of an asynchronous operation (Figure 27 on page 380), the application should reuse the targeting information from the originating asynchronous request.

Note: On z17 Systems and above, users may also use the HWIREST2 callable service to have BCPii configure an exit for users to monitor asynchronous notifications. Users also have the option to continue using the HWIREST callable service and may choose to configure their own ENF 68 Listener Exit or simply continue to poll the job status. Use of any of the new monitoring features for asynchronous notifications requires the configuration and use of JSON Web Tokens (JWTs) on the local and target systems. If users configure their own ENF68 Listener Exits, they should configure a BITQUAL filter with the value of their registration identifier encoded as raw hex (not the string value), or they may also choose to inspect the list of registration identifiers provided with the notification to ensure proper event correlation. When using the HWIREST2 Callable Service, BCPii takes care of these exit filters on the user's behalf.

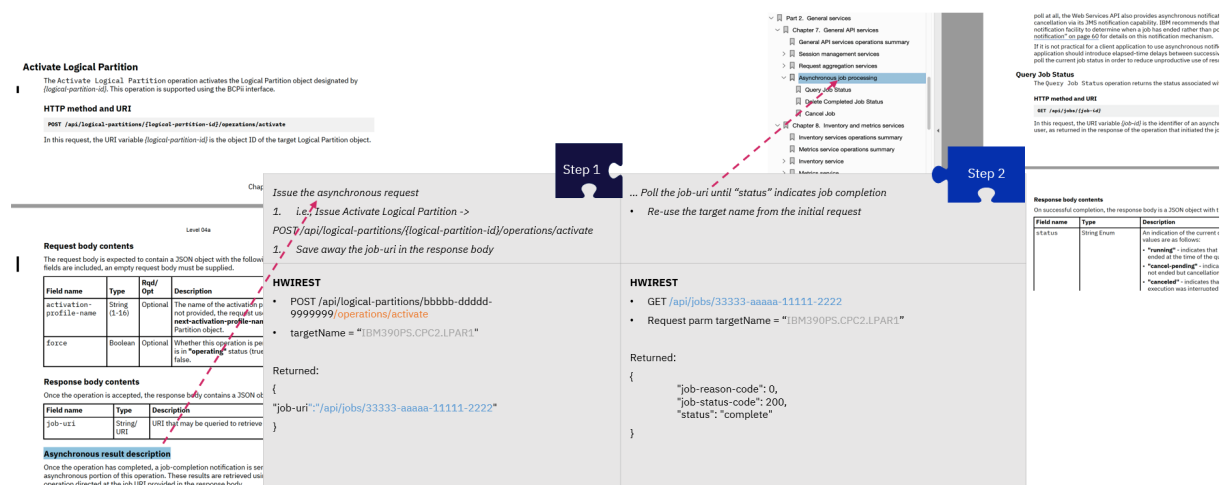


Figure 27. Polling result of an asynchronous operation

See "Syntax, linkage and programming considerations" on page 263 for details about how to call BCPii services in the various programming languages.

Restrictions

- This BCPii interface requires the SE and HMC associated with the local and target CPC to be at a minimum IBM Z15 hardware level. In addition, the minimum BCPii microcode level applied to the corresponding SE and HMC must be:

SE 2.15.0

MCL P46598.370, Bundle S38

HMC 2.15.0

MCL P46686.001, Bundle H25

- BCPii does not allow HWIREST to be issued from within a BCPii ENF exit routine.
- BCPii supports the use of asynchronous notifications on SEs and HMCs associated with a minimum IBM z17 hardware level. Asynchronous notifications may be configured to notify ENF exits to learn of events. For SEs and HMCs with z16® and prior hardware levels, users may poll for results or continue to use existing HWIEVENT services.
- BCPii supports the targeting of HMCs only on systems with a minimum IBM z17 hardware level.

Authorization

Given the nature of the BCPii API 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.

Authority to the particular resource accessed by HWIREST

Prior to z17 Systems, the main difference between HWIREST and the other APIs is how the identity of the resource is obtained.

Unlike the other APIs which have the concept of a connection token to uniquely identify the resource, for HWIREST, BCPii uses the content of the URI and targeting information to determine the identity of the resource. That identity is used to build a SAF resource name, which is used to authorize the request. In the case of a LIST request, the identity of each returned entity is also used to determine if it should be included in the response body. A BCPii application issuing the HWIREST API needs to have the appropriate authority to the particular resource that it is trying to access.

On IBM z17 Systems and above, there are three ways authorization may be checked which are configurable on the system.

1. **FACILITY:** (Default) The main difference between HWIREST and the other APIs is how the identity of the resource is obtained. Unlike the other APIs which have the concept of a connection token to uniquely identify the resource, for HWIREST, BCPii uses the content of the URI and targeting information to determine the identity of the resource. That identity is used to build a SAF resource name, which is used to authorize the request. In the case of a LIST request, the identity of each returned entity is also used to determine if it should be included in the response body. A BCPii application issuing the HWIREST API needs to have the appropriate authority to the particular resource that it is trying to access. (See note about when JWTs are always used.)
2. **JWT:** The user is checked to have access to the HWI.APPLNAME.HWISERV FACILITY class profile before obtaining a signed JSON Web Token (JWT) on behalf of the user and sending the JWT along with the request to the target system, ensuring both the local and target systems support JWT usage from BCPii. The signed JWT is used by the managing Hardware Management Console (HMC) to map the z/OS user ID to an HMC User or Template. A session is created for the HMC User or Template and the authorization capabilities assigned to this HMC User or Template are applied to the request issued through BCPii.
3. **JWTBYBRID:** Use JWTs whenever possible, when capable on both the local and target systems, as well as when available for the user issuing the request. In the event any of those checks fail, check if

the current FACILITY Class Profile based authorizations permit the current user to issue the request, and if the user is permitted, issue the request without the usage of a JWT.

To configure the HWIREST authorization mode to something other than the default, the HWI.AUTHMODE.HWIREST.<cpcname>.<sysname> FACILITY class profile's APPLDATA field may be configured with FACILITY, JWT, or JWTHYBRID. See note regarding when HWIREST authorization mode applies.

Note: The user defined HWIREST authorization mode does not apply in the following circumstances:

- The target is an HMC
- The URI is associated with asynchronous event notifications
- The URI was introduced for the IBM z17 system or later

Under these circumstances, JWT based authorization is required and used automatically. Not configuring JWTs and attempting to use BCPii under these circumstances will result in failure of the request.

SMF recording

The HWIREST API calls that issue a POST or PUT or DELETE and complete with an HTTP status in the 200 range will have SMF type 106 (X'6A') records written if the installation has activated recording of this record type in its active configuration.

For more information, see [Record type 106 \(X'6A'\) — BCPii activity in z/OS MVS System Management Facilities \(SMF\)](#).

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0009yyyy' for HWIREST for one of the following reasons:

Table 72. Reasons for abend X'042', RC X'0009yyyy' for HWIREST	
yyyy	Reason
0003	The parameters passed by the caller are not in the primary address space.
0004	The parameters passed by the caller are not accessible.
0005	The number of parameters passed by the caller is not correct.
0006	The response parameter passed in by the caller is not addressable.
0007	The parameters passed in by the caller are not addressable.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See [z/OS MVS System Codes](#) for additional information.

Non-REXX interface parms

Syntax

Write the call as shown in the syntax diagram.

CALL HWIREST(requestParmPtr, responseParmPtr);*Table 73. Non-REXX parameters***Non-REXX parameters**

```
CALL HWIREST(
    requestParmPtr,
    responseParmPtr);
```

Parameters

The parameters are explained as follows:

requestParmPtr

Supplied parameter

- Type: Pointer
- Length: 4 bytes

requestParmPtr specifies the address of a pre-defined structure, in the form specified by [Table 74](#) on [page 383](#), and contains the values associated with the REST API operation. The contents of this table are mapped by data structure **REQUEST_PARAM_TYPE** in the data mappings provided for the various programming languages supported. For more information, see [“Syntax, linkage and programming considerations”](#) on [page 263](#).

Table 74. RequestParmPtr parameter. RequestParmPtr field table

Field	Field Type	Field Description
httpMethod	4 byte unsigned integer	A REQUIRED field, correspond to the HTTP Method entity for the specific REST API operation. Supported values are: 1 (decimal)/HWI_REST_POST(constant) for HTTP POST Method 2 (decimal)/HWI_REST_GET(constant) for HTTP GET Method 3 (decimal)/HWI_REST_PUT(constant) for HTTP PUT Method 4 (decimal)/HWI_REST_DELETE(constant) for HTTP DELETE Method
uri	4 byte Pointer, address of character string	A REQUIRED field, the contents of the string correspond to the URI entity defined by the specific REST API operation and must be in the character set specified by the encoding field.
uriLen	4 byte unsigned integer	Length of the URI string specified, maximum is 2048 bytes.

Table 74. RequestParmPtr parameter. RequestParmPtr field table (continued)

Field	Field Type	Field Description
targetName	4 byte Pointer, address of character string	<p>A REQUIRED field for all but List CPC Object operation.</p> <p>The contents of the string corresponds to the X-API-Target-Name request header in <i>Hardware Management Console Web Services API</i>. If provided, must be in the character set specified by the encoding field.</p> <p>This field represents the routing information associated with the operation and is also used to verify that the user ID initiating the operation has the required authority for the resource the operation is targeting.</p> <p>Note: Set to NULL if target name is not provided</p>
targetNameLen	4 byte unsigned integer	Length of the target name string specified.
requestBody	4 byte Pointer, address of character string	<p>An optional field, the contents of the string represent a valid JSON body and must be in the character set specified by the encoding field.</p> <p>Note: Set to NULL if a request body is not provided.</p>
requestBodyLen	4 byte unsigned integer	Length of the request body if one is provided. Maximum is 64KB.
clientCorrelator	4 byte Pointer, address of character string	<p>An optional field, and if provided, must be in the character set specified by the encoding field.</p> <p>Note: Set to NULL if client correlator is not provided.</p>
clientCorrelatorLen	4 byte unsigned integer	Length of the client correlator string if one is provided. Maximum is 64 bytes.
encoding	4 byte unsigned integer	<p>Represents the character set being used for all the data associated with this REST API operation. This means that the URI, headers, and any request body data will use the specified character set. It also means that the response headers and the response body data will use this character set.</p> <p>Supported values are:</p> <p>1 (decimal)/HWI_ENCODING_UTF8(constant) for UTF-8</p> <p>2 (decimal)/HWI_ENCODING_IBM1047(constant) for IBM-1047</p> <p>Default, if initialized to 0, is IBM-1047.</p>

Table 74. RequestParmPtr parameter. RequestParmPtr field table (continued)		
Field	Field Type	Field Description
requestTimeout	4 byte unsigned integer	<p>Represents the amount of time the request, once it has reached the SE, is limited to. The total time of the request, including BCPii processing, maybe slightly longer.</p> <p>The 4 byte unsigned integer represents the time in milliseconds.</p> <ul style="list-style-type: none"> • valid range is from 0 - 0x005265C0 (90 minutes) • if the value is > 0, but < 5 seconds, a default of 5 seconds will be used • if the value is > 90 minutes, the default of 90 minutes will be used <p>Default, if initialized to 0, is 60 minutes.</p>

responseParmPtr

- Type: Pointer
- Length: 4 bytes

responseParmPtr specifies the address of a pre-defined structure in the form specified by [Table 75 on page 385](#). On input, the structure contains the addresses and lengths associated with the pre-allocated response data areas that will be filled in with the resulting response content. The contents of [Table 75 on page 385](#) are mapped by data structure **RESPONSE_PARM_TYPE** in the data mappings provided for the various programming languages supported. For more information, see [“Syntax, linkage and programming considerations” on page 263](#)

Table 75. ResponseParmPtr parameter. ResponseParmPtr field table		
Field	Field Type	Field Description
responseDate	4 byte Pointer, address of character string	<p>An optional field that points to a pre-allocated data area for the resulting Date response header.</p> <p>If the application does not want this information returned, the pointer should be set to NULL.</p> <p>Note: Will not be set if an error occurred prior to SE processing.</p>
responseDateLen	4 byte unsigned integer	<p>Size of the pre-allocated, 29 byte, data area pointed to by the responseDate pointer.</p> <p>On return from the service, this field will be updated with the actual size of the content.</p>
requestId	4 byte Pointer, address of character string	<p>An optional field that points to a pre-allocated data area for the resulting X-Request-Id response header.</p> <p>If the application does not want this information returned, the pointer should be set to NULL.</p> <p>Note: Will not be set if an error occurred prior to SE processing.</p>

Table 75. ResponseParmPtr parameter. ResponseParmPtr field table (continued)

Field	Field Type	Field Description
requestIdLen	4 byte unsigned integer	Size of the pre-allocated, 64 byte, data area that is pointed to by the requestId pointer. On return from the service, this field will be updated with the actual size of the content.
location	4 byte Pointer, address of character string	An optional field that points to a pre-allocated data area for the resulting Location response header. If the application does not want this information returned, the pointer should be set to NULL.
locationLen	4 byte unsigned integer	Size of the pre-allocated, 2048 byte, data area pointed to by the location pointer. On return from the service, this field will be updated with the actual size of the content.
responseBody	4 byte Pointer, address of character string	A REQUIRED field that points to a pre-allocated data area for the resulting response body . On failure, HTTP Status Codes 4xx or 5xx, the response body may contain a valid JSON object with further information regarding the error.
responseBodyLen	4 byte unsigned integer	Size of the pre-allocated, minimum 500 bytes, data area pointed to by the responseBody pointer. Maximum is 15MB. On return from the service, this field will be updated with the actual size of the content.
HTTPStatus	4 byte unsigned integer	HTTP Status Codes. Additional information may be provided in the response body for failed operations that result in the HTTP Status Codes 4xx or 5xx.
reasonCode	4 byte Integer	The reason code for failed operations provides additional information associated with the failed HTTP Status Code.
targetName		A REQUIRED field for all but List CPC / HMC Object operations. Supports HMC targeting with syntax: HMC : //<hmcname>

Example

```

bool getCPCInfo()
{
    static const int defaultLen2K = 2048;
    static const int defaultLen = 256;
    static const int defaultLen15MB = 15728640;

    bool listSuccess = false;

    REQUEST_PARM_TYPE request;
    RESPONSE_PARM_TYPE response;

```

```

char *uri = (char *)malloc(defaultLen2K);
char *responseBody = (char *)malloc(defaultLen15MB);
char *responseData = (char *)malloc(defaultLen);
char *requestId = (char *)malloc(defaultLen);

memset(&request, 0, sizeof(REQUEST_PARM_TYPE));
memset(&response, 0, sizeof(RESPONSE_PARM_TYPE));

/* Issue a CPC LIST request to obtain the uri
   and target name associated with CPC named
   T115

   GET /api/cpcs?name=<CPCname>

   NOTE: CPC LIST is the only request that does
   not require a target name value because it
   will automatically be sent to the local SE
*/
memset(uri, 0, defaultLen2K);
strcpy(uri, "/api/cpcs?name=T115");

/* initialize all the required input data for the request */
request.uri = uri;
request.uriLen = strlen(uri);
request.httpMethod = HWI_REST_GET;
request.requestTimeout = 0x00002688;

/* initialize the response structure with
   the address and length of the pre-allocated
   data areas

   when the service returns, the data areas
   will contain the response value for that
   specific field and the data area length
   will be updated to reflect the length of
   that value
*/
memset(responseBody, 0, defaultLen15MB);
memset(responseData, 0, defaultLen);
memset(requestId, 0, defaultLen);
response.responseBody = responseBody;
response.responseBodyLen = defaultLen15MB;
response.responseDate = responseData;
response.responseDateLen = defaultLen;
response.requestId = requestId;
response.requestIdLen = defaultLen;

hwirest(
    &request,
    &response);

/* An httpStatus in the 200 range indicates the request was successful
   NOTE: A success does not mean the cpc info was returned,
   the response body may contain an empty cpcs array because
   the SE was not able to match the CPC name or the user ID was
   not authorized to that CPC
*/
if ((response.httpStatus > 199 && response.httpStatus < 300) &&
    response.responseBodyLen > 0)
{
    /* Parse the response JSON text. */
    if (parse_json_text((char *)response.responseBody))
    {
        HWTJ_HANDLE_TYPE arrayhandle;
        HWTJ_HANDLE_TYPE arrayentry;
        int entryNum = 0;

        arrayhandle = find_array(0, "cpcs");
        if (arrayhandle != NULL)
        {
            entryNum = getnumberOfEntries(arrayhandle);
            if (entryNum == 1)
            {

```

```

        arrayentry = getArrayEntry(arrayhandle, 0);
        CPCuri = find_string(arrayentry, "object-uri");
        CPCTargetName = find_string(arrayentry, "target-name");

        if (CPCuri != NULL && CPCTargetName != NULL)
        {
            printf("CPCuri:%s\n", CPCuri);
            printf("CPCTargetName:%s\n", CPCTargetName);
            listSuccess = true;
        }
    }
    else
    {
        printf("empty cpcs array returned\n");
    }
}
else
{
    printf("cpc array not found\n");
}
}
} else {
    traceFailureResponse(&response);
}

free(uri);
free(responseBody);
free(responseDate);
free(requestId);

return listSuccess;
}

void traceFailureResponse(RESPONSE_PARAM_TYPE *pParm)
{
    int isBCPiiError = 0;
    if (pParm->httpStatus < 200 || pParm->httpStatus > 299)
    {
        printf("*>>\n");
        printf("*>>REQUEST failed \n");

        if (pParm->responseBodyLen > 0)
        {
            printf("* >responseBodyLen: %d (dec) %X (hex)\n",
                pParm->responseBodyLen, pParm->responseBodyLen);

            if (parse_json_text((char *)pParm->responseBody))
            {
                HWTJ_HANDLE_TYPE arrayentry;
                char *errorMsg;

                isBCPiiError = find_boolvalue(0, "bcpii-error");

                if (isBCPiiError == 0)
                {
                    printf("bcpii-error is false\n");
                } else if (isBCPiiError == 1) {
                    printf("bcpii-error is true\n");
                } else {
                    printf("bcpii-error not found\n");
                }

                errorMsg = find_string(0, "message");
                if (errorMsg != NULL)
                {
                    printf("error: %s\n", errorMsg);
                }
            }
        }
    }

    /* In the case of BCPii flagging the error, if that occurred
       when processing the SE response, then some of the
       other response fields may contain content to tie the
    */
}

```



```

        'failed' response back to the SE
    */
    printTextStr(pParm->requestIdLen, (char *)pParm->requestId, "requestId",
                (char **)&pParm->requestId);
    printTextStr(pParm->locationLen, (char *)pParm->location, "location",
                (char **)&pParm->location);
    printTextStr(pParm->responseDateLen, (char *)pParm->responseDate, "responseDate",
                (char **)&pParm->responseDate);
}
else
{
    printf("error logic, request was good but inside traceFailureResponse\n");
}
}

```

REXX interface parms

Syntax

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

address bcpii "hwirest requestParm, responseParm."

Table 76. REXX parameters

REXX parameters

```
address bcpii "hwirest requestParm. responseParm."
```

Parameters

- All parameters passed on BCPii REXX service calls must be REXX variables. Literals are not supported (for example, a variable name which has been assigned the value of a */api/cpcs/{cpc-id}/logical-partitions* should be specified on the call instead of the value itself).
- Variable names, (for example, requestParm and responseParm), specified on BCPii REXX service calls, are limited to 40 characters in length.
- Stem variables utilized by BCPii have hard-coded stem variable tail values which correspond the documented stem variable tail names in [Table 77 on page 390](#) and [Table 78 on page 392](#). For example, to set the URI value for the request, the requestParm. stem must be prepared in REXX with the exact stem tail variable "URI":

```
requestParm.URI = "/api/cpcs/{cpc-id}/logical-partitions"
```

The parameters are explained as follows:

requestParm.

Supplied parameter

- compound (stem) variable

The requestParm content reflects the values associated with the operation that is being issued. It is initialized in the form of x.<stem variable tail> where the stem variable tail is defined by [Table 77 on page 390](#) and x is the name of the stem variable specified on the parameter list. If the value of an input variable is incompatible with the stem variable tail type required, an error is flagged. Any stem variable tails that are not applicable to the request, or ones that should use the default value, should be left uninitialized. If the requestParm stem variable is being reused for multiple operations, it is recommended to use the DROP keyword

```
DROP requestParm.
```

to unassign variables before each HWIREST call.

Table 77. RequestParm stem tail variables. RequestParm stem tail variables

Stem tail variable	Stem tail variable type	Stem tail variable description
httpMethod	integer	<p>A REQUIRED field, corresponds to the HTTP Method entity for the specific REST API operation.</p> <p>Supported values are:</p> <p>1 (decimal)/HWI_REST_POST(constant) for HTTP POST Method</p> <p>2 (decimal)/HWI_REST_GET(constant) for HTTP GET Method</p> <p>3 (decimal)/HWI_REST_PUT(constant) for HTTP PUT Method</p> <p>4 (decimal)/HWI_REST_DELETE(constant) for HTTP DELETE Method</p>
uri	string	<p>A REQUIRED field, the contents correspond to the URI entity as defined by the specific REST API operation and must be in the character set specified by the encoding field.</p> <p>Maximum 2048 bytes.</p>
targetName	string	<p>A REQUIRED field for all but List CPC Object operation.</p> <p>This field corresponds to the X-API-Target-Name request header in <i>Hardware Management Console Web Services API</i>.</p> <p>This field represents the routing information associated with the operation and is also used to verify that the user ID initiating the operation has the required authority for the resource the operation is targeting.</p> <p>If provided, must be in the character set specified by the encoding field.</p> <p>Note: Leave uninitialized if a value is not available.</p>
requestBody	string	<p>If provided, must be in the character set specified by the encoding field.</p> <p>System REXX/ISV REXX:</p> <p>Maximum supported is 64KB.</p> <p>TSO/E REXX:</p> <p>Maximum supported is 32767 bytes.</p> <p>Note: Leave uninitialized if a value is not available.</p>

Table 77. RequestParm stem tail variables. RequestParm stem tail variables (continued)		
Stem tail variable	Stem tail variable type	Stem tail variable description
clientCorrelator	string	<p>An optional field corresponding to the X-Client-Correlator request header.</p> <p>If provided, must be in the character set specified by the encoding field. Maximum supported is 64 bytes.</p> <p>Note: Leave uninitialized if a value is not available.</p>
encoding	integer	<p>An optional field that represents the character set being used for all the data associated with this REST API operation. This means that the URI, headers, and any request body data will use the specified character set. It also means that the response headers and the response body data will use this character set.</p> <p>Supported values are:</p> <p>1 (decimal)/HWI_ENCODING_UTF8(constant) for UTF-8</p> <p>2 (decimal)/HWI_ENCODING_IBM1047(constant) for IBM-1047</p> <p>Default, if not specified or initialized to 0, is IBM-1047.</p>
requestTimeout	integer	<p>An optional field that represents the amount of time the request, once it has reached the SE, is limited to. The total time of the request, including BCPii processing, maybe slightly longer.</p> <p>The 4 byte integer represents the time in milliseconds.</p> <ul style="list-style-type: none"> • valid range is from 0 - 0x005265C0 (90 minutes) • if the value is > 0, but < 5 seconds, a default of 5 seconds will be used • if the value is > 90 minutes, the default of 90 minutes will be used <p>Default, if not specified or initialized to 0, is 60 minutes.</p>

responseParm.

Returned parameter

- compound (stem) variable

The responseParm content reflects the result of the operation issued. It is returned using stem variables in the form of x.<stem variable tail> where the stem variable tail is defined by [Table 78 on page 392](#) and x is the name of the stem variable specified on the parameter list. The responseParm stem variable should be left uninitialized and will be set by BCPii. If the responseParm stem variable is being reused for multiple operations, it is recommended to use the DROP keyword

DROP responseParm.

to unassign variables before each HWIREST call. For more information, see [DROP](#) in *z/OS TSO/E REXX Reference*.

Table 78. ResponseParm stem variables. ResponseParm stem variables		
Stem variable	Stem variable type	Stem variable description
responseDate	string	The date and time, from the perspective of the SE's clock, at which the response message was generated. Note: Will not be set if an error occurred prior to SE processing.
requestId	string	A string that provides diagnostic information identifying the request from the perspective of the SE. Note: Will not be set if an error occurred prior to SE processing.
location	string	This is returned for newly created objects, containing the URI of the newly created object.
responseBody	string	On failure, HTTP Status Codes 4xx or 5xx, the response body may contain a valid JSON object with further information regarding the error. Note: The response body size for the REXX interface is limited to maximum 2.5 MB. Consider using C or ASM if the returned response body exceeds 2.5 MB.
HTTPStatus	Integer	HTTP Status Codes. Additional information may be provided in the response body for failed operations that result in HTTP Status Codes 4xx or 5xx.
reasonCode	Integer	Provides additional information associated with failed operations.
targetName		A REQUIRED field for all but List CPC / HMC Object operations. Supports HMC targeting with syntax: HMC://<hmcname>

Example

```
drop userRequest.
drop userResponse.

/* Every application needs to start off by issuing a LIST CPCs
   to obtain URI and target name information for the specific
   CPC that will be interacted with.
   The following LIST CPCs request is tailored to return information
   for a CPC with the name T115.

   NOTE: A LIST CPCs is the only request that does not require a TARGETNAME
   value.
*/
userRequest.HTTPMETHOD = HWI_REST_GET
userRequest.URI = '/api/cpcs?name=T115'
userRequest.CLIENTCORRELATOR = 'restSample'
userRequest.ENCODING = HWI_ENCODING_IBM1047
userRequest.REQUESTTIMEOUT = 0 /* use default of 60 minutes */
```

```

Address BCPII "HWIREST userRequest. userResponse."

/* For non-zero REXX RC, still continue to inspect the
   response parm stem variable for additional error
   details that may have been provide.
*/
say 'Rexx RC: (||RC||)'
say 'HTTP Status: (||userResponse.httpstatus||)'
successIndex = INDEX(userResponse.httpstatus, '2')

/* If the HTTP Status was in the 200 range then
   display the values expected on success, otherwise
   isolate the error information.
*/
if successIndex = 1 then
do /* SE responded successfully */
  say 'SE DateTime: (||userResponse.respondedate||)'
  say 'SE requestId: ( ' || userResponse.requestId || ' )'

  if userResponse.httpstatusNum = '201' Then
    say 'Location Response: ( ' || userResponse.location || ' )'

  /* NOTE: A success does not mean the cpc info was returned,
     the response body may contain an empty cpcs array because
     the SE was not able to match the CPC name or the user ID was
     not authorized to that CPC
  */
  if userResponse.responsebody <> '' Then
  do /* response body */
    say 'Response Body: ( ' || userResponse.responsebody || ' )'
    emptyCPCResponse = '{"cpcs":[]}'
    CPCInfoResponse = userResponse.responsebody
    emptyCPCArray = INDEX(CPCInfoResponse, emptyCPCResponse)
    if emptyCPCArray > 0 | CPCInfoResponse = '' Then
      do
        say 'fatalError ** failed to get CPC info **'
        return 0
      end

    /* Parse the response to obtain the uri
       and target name associated with CPC,
       which will be used to query storage info
    */
    call JSON_parseJson CPCInfoResponse

    CPCuri = JSON_findValue(0,"object-uri", HWTJ_STRING_TYPE)
    if CPCuri = '' then
      do
        say 'fatalError ** failed to get CPC uri**'
        return 0
      end

    CPCtargetName = JSON_findValue(0,"target-name", HWTJ_STRING_TYPE)
    if CPCtargetName = '' then
      do
        say 'fatalError ** failed to get CPC target name**'
        return 0
      end
    end /* response body */
  end /* SE responded successfully */
else
do /* error path */
  say 'Reason Code: (||userResponse.reasoncode||)'

  if userResponse.respondedate <> '' then
    say 'DateTime: (||userResponse.respondedate||)'

  if userResponse.requestId <> '' Then
    say 'requestId: ( ' || userResponse.requestId || ' )'

  if response.responsebody <> '' Then
  do /* response body */
    say 'responseBody: (||userResponse.responsebody||)'
    call JSON_parseJson userResponse.responsebody

    if RESULT <> 0 then
      say 'failed to parse response'
    else
      do
        bcpiiErr=JSON_findValue(0, "bcpii-error", HWTJ_BOOLEAN_TYPE)

```

```

        if bcpiiErr = 'true' then
            say '*** BCPii generated error message:'
        else
            say '*** SE generated error message:'

            errmessage=JSON_findValue(0,"message", HWTJ_STRING_TYPE)
            say '('||errmessage||')'
            say
            say 'Complete Response Body: (' || userResponse.responsebody || ')'
        end /* err */
    end /* response body */
end /* error path */

```

HWIREST2 Callable Service

Call the HWIREST2 callable service to perform synchronous REST API operations, including the new register for asynchronous event notifications, and to get, modify or delete the registrations. When registering for asynchronous event notifications, HWIREST2 provides parameters used by BCPii to setup the ENF68 listener exit on behalf of the user. The requests will continue to be sent as an SCLP packet over the current internal transport network to the local support element (SE) before being routed to target SEs or Hardware Management Consoles (HMCs).

Description

Environment

Requirement	Details
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
Console setup:	The main console on the HMC must be activated in order for the operating system commands to be sent successfully. To activate the main console, use the vary command: v cn(*),activate.
JSON Web Token (JWT) setup:	Signed JWTs must be configured on the LPAR in use and both the issuing user and the user associated with the BCPii started task must have JWTs configured. Additionally, mappings on the managing HMC must be defined for both users to map to a HMC User or Template with adequate permissions. The user associated with the BCPii started task must have permissions to list SEs, list HMCs, and list-features for SEs.

Programming requirements

An application using this service is expected to begin by issuing the List CPC Objects or the List Targetable HMCs operation. (See Note below.) The List CPC Objects operation obtains the URI and targeting

information for the CPC(s). The List Targetable HMCs operation lists the HMCs that can potentially be targeted from the local system for HWIREST2 requests.

Note: List CPC Objects and List Targetable HMCs are the only REST APIs that do not require targeting information and will always be directed to the local SE.

The response for this request will contain an array of CPC or HMC objects. Each CPC object will include its corresponding URI (value of the "object-uri" property) and targeting information (value of the "target-name" property). All subsequent HWIREST2 requests will build or re-use that information when interacting with one or more of the specific CPCs and HMCs.

Each HMC Object will contain its target-name and other relevant information regarding its current configurations and ability to be targeted. This includes whether Web Services is enabled and if a BCPii authorization record exists on the HMC. Both of these conditions must be true for the HMC to be a valid target. Note the HMC name must begin with *HMC://* when used to target HMCs with supporting URIs.

When using asynchronous URIs, the caller can optionally provide exit information along with the request to have BCPii configure the exit on the user's behalf. This exit will automatically be invoked with an ENF68 type response, containing a eventData of JSON information regarding the async event. The user is then expected to call the deregister URI providing their registration ID to have BCPii remove the exit once no longer interested in asynchronous notifications. See HWIREST, [“Programming requirements” on page 379](#) for more information regarding Local CPC targeting and other resource listing capabilities.

See [“Syntax, linkage and programming considerations” on page 263](#) for details about how to call BCPii services in the various programming languages.

Restrictions

- This BCPii interface requires the SE and HMC associated with the local and target CPC to be at a minimum IBM Z17 hardware level.
- BCPii does not allow HWIREST2 to be issued from within a BCPii ENF exit routine.

Authorization

HWIREST2 exclusively supports JSON Web Token (JWT) based authorization. BCPii obtains the JWT on behalf of the requesting z/OS user ID and supplies the JWT with the request. A mapping must exist on the managing Hardware Management Console (HMC) to map the z/OS user ID to an HMC User or Template. These HMC User or Template permissions are then used for all authorization checks needed for the session created to process the user request.

The user issuing the BCPii request must still have at least READ access to the HWI.APPLNAME.HWISERV FACILITY class profile to be able to issue the request. Ensure both the issuing user and the user associated with the BCPii started task have been configured to use signed JWTs and have a mapping defined on the managing HMC.

SMF recording

The HWIREST2 API calls that issue a POST or PUT or DELETE and complete with an HTTP status in the 200 range will have SMF type 106 (X'6A') records written if the installation has activated recording of this record type in its active configuration.

For more information, see [Record type 106 \(X'6A'\) — BCPii activity in z/OS MVS System Management Facilities \(SMF\)](#).

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0009yyyy' for HWIREST2 for one of the following reasons:

Table 79. Reasons for abend X'042', RC X'0009yyyy' for HWIREST2

yyyy	Reason
0003	The parameters passed by the caller are not in the primary address space.
0004	The parameters passed by the caller are not accessible.
0005	The number of parameters passed by the caller is not correct.
0006	The response parameter passed in by the caller is not addressable.
0007	The parameters passed in by the caller are not addressable.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See [z/OS MVS System Codes](#) for additional information.

Parameters

The parameters are explained as follows:

request2ParmPtr

Supplied parameter

- Type: Pointer
- Length: 4 bytes

request2ParmPtr specifies the address of a pre-defined structure, in the form specified below, and contains the values associated with the REST API operation. The contents of this table are mapped by data structure **REQUEST_REST2_PARM_TYPE** in the data mappings provided for the various programming languages supported. For more information, see [“Syntax, linkage and programming considerations”](#) on page 263.

Table 80. Request2ParmPtr parameter. Request2ParmPtr field table

Field	Field Type	Field Description
httpMethod	4 byte unsigned integer	A REQUIRED field, corresponds to the HTTP Method entity for the specific REST API operation. Supported values are: 1 (decimal)/HWI_REST_POST(constant) for HTTP POST Method 2 (decimal)/HWI_REST_GET(constant) for HTTP GET Method 3 (decimal)/HWI_REST_PUT(constant) for HTTP PUT Method 4 (decimal)/HWI_REST_DELETE(constant) for HTTP DELETE Method
uri	4 byte Pointer, address of character string	A REQUIRED field, the contents of the string correspond to the URI entity defined by the specific REST API operation and must be in the character set specified by the encoding field.
uriLen	4 byte unsigned integer	Length of the URI string specified, maximum is 2048 bytes.

Table 80. Request2ParmPtr parameter. Request2ParmPtr field table (continued)

Field	Field Type	Field Description
targetName	4 byte Pointer, address of character string	<p>A REQUIRED field for all but List CPC/HMC Object operation.</p> <p>The contents of the string corresponds to the X-API-Target-Name request header in <i>Hardware Management Console Web Services API</i>. If provided, must be in the character set specified by the encoding field.</p> <p>This field represents the routing information associated with the operation and is also used to verify that the user ID initiating the operation has the required authority for the resource the operation is targeting.</p> <p>Note: Set to NULL if target name is not provided</p>
targetNameLen	4 byte unsigned integer	Length of the target name string specified.
requestBody	4 byte Pointer, address of character string	<p>An optional field, the contents of the string represent a valid JSON body and must be in the character set specified by the encoding field.</p> <p>Note: Set to NULL if a request body is not provided.</p>
requestBodyLen	4 byte unsigned integer	Length of the request body if one is provided. Maximum is 64KB.
clientCorrelator	4 byte Pointer, address of character string	<p>An optional field, and if provided, must be in the character set specified by the encoding field.</p> <p>Note: Set to NULL if client correlator is not provided.</p>
clientCorrelatorLen	4 byte unsigned integer	Length of the client correlator string if one is provided. Maximum is 64 bytes.
encoding	4 byte unsigned integer	<p>Represents the character set being used for all the data associated with this REST API operation. This means that the URI, headers, and any request body data will use the specified character set. It also means that the response headers and the response body data will use this character set.</p> <p>Supported values are:</p> <p>1 (decimal)/HWI_ENCODING_UTF8(constant) for UTF-8</p> <p>2 (decimal)/HWI_ENCODING_IBM1047(constant) for IBM-1047</p> <p>Default, if initialized to 0, is IBM-1047.</p>

Table 80. Request2ParmPtr parameter. Request2ParmPtr field table (continued)

Field	Field Type	Field Description
requestTimeout	4 byte unsigned integer	<p>Represents the amount of time the request, once it has reached the SE, is limited to. The total time of the request, including BCPii processing, maybe slightly longer.</p> <p>The 4 byte unsigned integer represents the time in milliseconds.</p> <ul style="list-style-type: none"> Valid range is from 0 - 0x005265C0 (90 minutes) If the value is > 0, but < 5 seconds, a default of 5 seconds will be used If the value is > 90 minutes, the default of 90 minutes will be used <p>Default, if initialized to 0, is 60 minutes.</p>
eventExitMode	4 Byte Integer	<p>EventExitMode specifies the type of the exit mode for the service. At present, only one value is allowed for this parameter. In the future, IBM might choose to allow additional values to be specified.</p> <p>1: HWI_EVENT_TASK- The base control program internal interface gives control in task mode to an ENF listen-exit routine as specified on the EventExitAddr parameter.</p> <p>Task mode ENF exits must reside in common storage.</p>
eventExitAddr	4 Byte Pointer	<p>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.</p> <p>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.</p>
eventExitParm	4 Byte Pointer or Integer	<p>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.</p>

responseParmPtr

- Type: Pointer
- Length: 4 bytes

The responseParmPtr is reused from the HWIREST Callable Service parameters with the JSON content varying based on the specified URI in accordance with the *Hardware Management Console Web Services API*.

responseParmPtr specifies the address of a pre-defined structure in the form specified in the HWIREST, Interface Params, Parameters subsection. On input, the structure contains the addresses

and lengths associated with the pre-allocated response data areas that will be filled in with the resulting response content. The contents are mapped by data structure **RESPONSE_PARM_TYPE** in the data mappings provided for the various programming languages supported. For more information, see [“Syntax, linkage and programming considerations”](#) on page 263.

Syntax

Write the call as shown in the syntax diagram.

Note: Only C and Assembler Interfaces are provided for HWIREST2.

CALL HWIREST2(request2ParmPtr, responseParmPtr);

<i>Table 81. Non-REXX parameters</i>
Non-REXX parameters
<pre>CALL HWIREST2(request2ParmPtr, responseParmPtr);</pre>

HWISET/HWIREST2 – BCPii set single or multiple SE/HMC-managed attributes

Call the HWISET service to change or set a single hardware attribute associated with a Central Processor Complex (CPC), CPC image (LPAR), activation profile, group profile or LPAR Capacity group.

Call the HWIREST2 service to change or set one or more hardware attributes associated with a single Central Processor Complex (CPC). These attributes can be associated with the CPC, one or more CPC images (LPARs) or LPAR Capacity groups on the same CPC or any activation profile type or group profile on the same CPC.

- When targeting a CPC at the z13 GA2 level or later, all attributes are either set or all are unchanged after the HWIREST2 call. If one or more attributes are not able to be set, all the attributes will be rolled back to the state prior to the HWIREST2 call.
- When targeting a CPC at the z13 GA1 level or earlier and one or more attributes are not able to be set, the HWIREST2 request results in a return code of HWI_SET2_PARTIAL_UPDATE. Prior to calling the HWIREST2 service, it is good practice to call the HWIQUERY service to retrieve and save the current values of the attributes to be set. Applications can then take the appropriate action when the HWIREST2 request results in the HWI_SET2_PARTIAL_UPDATE return code. See the description of the HWI_SET2_PARTIAL_UPDATE return code for more information.

Description

Environment

The requirements for the callers are:

Requirement	Details
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)

Requirement	Details
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 263 for details about how to call BCPii services in the various programming languages.

See [“HWIQUERY and HWISET / HWISSET2 attributes”](#) on page 825 for the summary table of the BCPii HWIQUERY and HWISET / HWISSET2 attributes and the objects that can be targeted for each function.

REXX programming considerations for the HWISET / HWISSET2 service

All information for the HWISET service applies for REXX requests except:

- SetTypeValue replaces SetTypeValue_Ptr. The actual value to be set, represented in character form, is passed instead of a pointer.
- The SetTypeValueLen input parm is not used.

All information for the HWISSET2 service applies for REXX requests except:

- A **set** parameter stem variable (for example, SetParm) replaces SetParm_Ptr.
- SetParm.0 replaces NumOfAttributes. SetParm.0 is required to specify the number of attributes to be set. The valid range for the SetParm.0 value is 1 - 9.
- SetParm.n.SET2_CTOKEN replaces the Set2_Ctoken field in the structure pointed to by SetParm_Ptr. It must contain the ConnectionToken representing the *n*th attribute to be set.
- SetParm.n.SET2_SETTYPE replaces the Set2_Settype field in the structure pointed to by SetParm_Ptr. It must contain the SetType value of the *n*th attribute to be set. See the SetType parameter for details on the value choices, which can be specified.
- SetParm.n.SET2_SETVALUE replaces the Set2_SetValue_Ptr field in the structure pointed to by SetParm_Ptr. It must contain the value to be set. See the SetTypeValue parameter for details on the type of data to be set.

Restrictions

BCPii does not allow HWISET or HWISSET2 to be issued from within a BCPii ENF exit routine.

Code page consideration

All input data to be set via the HWISET or HWISSET2 service will be translated from EBCDIC to ASCII which is required by the SE. Due to the nature of EBCDIC-to-ASCII and ASCII-to-EBCDIC conversions, certain irregularities exist in the conversion tables. These conversion irregularities, including characters like €, !, [,] and |, will not translate correctly and therefore should be used with extreme caution.

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 values, activation profile-related values, group profile values, or LPAR Capacity group values. If setting image-related values, the client

application must have at least update access to the SAF-protected FACILITY class resource profile HWI.TARGET.netid.nau.imagename.

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

SMF recording

Requests that complete with a return code of zero will have SMF type 106 (X'6A') records written if the installation has activated recording of this record type in its active configuration.

Syntax

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

Table 82. HWISET syntax	
Non-REXX parameters	REXX parameters
<pre>CALL HWISET(ReturnCode, ConnectToken, SetType, SetTypeValue_Ptr, SetTypeValueLen, DiagArea);</pre>	<pre>address bcpii "hwiset ReturnCode ConnectToken SetType SetTypeValue DiagArea."</pre>

Table 83. HWISET2 syntax	
Non-REXX parameters	REXX parameters
<pre>CALL HWISET2(ReturnCode, ConnectToken, SetParm_Ptr, NumofAttributes, DiagArea);</pre>	<pre>address bcpii "hwiset2 ReturnCode ConnectToken SetParm. DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

ReturnCode contains the return code from the service.

ConnectToken

Supplied parameter

- Type: Character string
- Length: 16 bytes

For the HWISET service, the **ConnectToken** represents a logical connection between the application and a CPC, image, activation profile, group profile and LPAR Capacity group. For the HWISET2 service, the **ConnectToken** must represent a logical connection between the application and a CPC. **ConnectToken** is an output parameter on the HWICONN service call.

In most cases, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call. For BCPii REXX execs running under the TSO/E or ISV-provided REXX environments, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same task.

SetType (HWISET only)

Supplied parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

SetType specifies the type of set request.

The following table is the list of valid set types. See the following documentation for more information:

- *IBM z SNMP Application Programming Interfaces* (SB10-7171-06)
- *System z10 and eServer zSeries Application Programming Interfaces* (SB10-7030-09)
- *System z9 and eServer zSeries Application Programming Interfaces* (SB10-7030-08)
- Publication appropriate to the level of hardware that the HWISET / HWISSET2 is targeted.

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
6 (6) HWI_ACCSTAT	Requests to change or set the acceptable CPC status values. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
7 (7) HWI_APROF	Requests to change or set the next activation reset profile name. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
27 (39) HWI_PRUNTYPE	Requests to change or set the processor running time type. Note: The input connection token represents a <i>CPC connection</i> or a <i>reset activation profile connection</i> .
28 (40) HWI_PRUNTIME	Requests to change or set the processor running time type. Note: The input connection token must only represent a <i>CPC connection</i> or a <i>reset activation profile connection</i> .
29 (41) HWI_PRUNTSEW	Requests to change or set the processor running time slice end wait processing. Note: The input connection token must only represent a <i>CPC connection</i> or a <i>reset activation profile connection</i> . This attribute is only available when targeting a z13 GA2 or lower CPC.
70 (112) HWI_DEFCAP	Requests to change or set the current defined capacity. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
71 (113) HWI_SGPIPW	Requests to change or set the shared general processor initial processing weight (SGPIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
72 (114) HWI_SGPIPWCAP	Requests to change or set the SGPIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
73 (115) HWI_SGPPWMIN	Requests to change or set the minimum SGPPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
74 (116) HWI_SGPPWMAX	Requests to change or set the maximum SGPPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
77 (119) HWI_WLM	Requests to change or set whether WLM is allowed to change SGPPW values. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
78 (120) HWI_IFAIPW	Requests to change or set the integrated facility for applications initial processing weight (IFAIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
79 (121) HWI_IFAIPWCAP	Requests to change or set the IFAIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7A (122) HWI_IFAPWMIN	Requests to change or set the minimum IFAPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7B (123) HWI_IFAPWMAX	Requests to change or set the maximum IFAPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7E (126) HWI_IFLIPW	Requests to change or set the integrated facility for Linux initial processing weight (IFLIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
7F (127) HWI_IFLIPWCAP	Requests to change or set the IFLIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
80 (128) HWI_IFLPWMIN	Requests to change or set the minimum IFLPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
81 (129) HWI_IFLPWMAX	Requests to change or set the maximum IFLPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
84 (132) HWI_ICFIPW	Requests to change or set the internal coupling facility initial processing weight (ICFIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
85 (133) HWI_ICFIPWCAP	Requests to change or set the ICFIPW be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
86 (134) HWI_ICFPWMIN	Requests to change or set the minimum ICFPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
87 (135) HWI_ICFPWMAX	Requests to change or set the maximum ICFPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8A (138) HWI_IIPW	Requests to change or set the integrated information processors initial processing weight (IIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8B (139) HWI_IIPWCAP	Requests to change or set the IIPW be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
8C (140) HWI_IIPPWMIN	Requests to change or set the minimum IIPPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8D (141) HWI_IIPPWMAX	Requests to change or set the maximum IIPPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
92 (146) HWI_GROUP_PROFILE_CAPACITY	<ul style="list-style-type: none"> For group profile connection (the input ConnectToken represents a group profile connection), requests to change or set the workload unit capacity for a group profile. For LPAR Capacity group connection (the input ConnectToken represents an LPAR Capacity group connection), requests to change or set the dynamic workload unit capacity for an LPAR Capacity group. This attribute for this connection type is supported on z14 GA2 or higher CPC. For image connection (the input ConnectToken represents an image connection), requests to change or set the dynamic workload unit capacity for a group of images in which the target image is a member. This attribute for this connection type requires that the target image be on a z196 (zEnterprise) or higher CPC and is a member of an LPAR Capacity group. If these requirements are not met, the request will fail with RC=X'101(HWI_COMMUNICATION_ERROR), with the DiagCommErr value set to X'15' (HWMCA_DE_SNMP_ERROR). Note: The capacity value is changed until the image is activated again. HWI_GROUP_PROFILE_CAPACITY is not persistent across activations.
95 (149) HWI_ABSCAP	Request to change or set whether absolute capping is enabled for general purpose processors. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection. This attribute must be enabled prior to setting the absolute capping value using attribute HWI_ABSCAPVAL.
96 (150) HWI_ABSCAPVAL	Requests to change or set the maximum general purpose processor consumption for the target image. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
97 (151) HWI_IFAABSCAP	Requests to change or set whether absolute capping is enabled for AAP processors. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection. This attribute must be enabled prior to setting the absolute capping value using attribute HWI_IFAABSCAPVAL.
98 (152) HWI_IFAABSCAPVAL	Requests to change or set the maximum AAP processor consumption for the target image. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.
99 (153) HWI_IFLABSCAP	Requests to change or set whether absolute capping is enabled for IFL processors. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection. This attribute must be enabled prior to setting the absolute capping value using attribute HWI_IFLABSCAPVAL.
9A (154) HWI_IFLABSCAPVAL	Requests to change or set the maximum IFL processor consumption for the target image. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.
9B (155) HWI_ICFABSCAP	Requests to change or set whether absolute capping is enabled for IFC processors. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection. This attribute must be enabled before setting the absolute capping value using attribute HWI_ICFABSCAPVAL.
9C (156) HWI_ICFABSCAPVAL	Requests to change or set the maximum IFC processor consumption for the target image. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection.
9D (157) HWI_IIPABSCAP	Requests to change or set whether absolute capping is enabled for IIP processors. This attribute is only available when targeting a ZEC12 GA2 or higher CPC. Note: The input connection token must only represent an image or an image activation profile connection. This attribute must be enabled before setting the absolute capping value using attribute HWI_IIPABSCAPVAL.

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
9E (158) HWI_IIPABSCAPVAL	<p>Requests to change or set the maximum IIP processor consumption for the target image. This attribute is only available when targeting a ZEC12 GA2 or higher CPC.</p> <p>Note: The input connection token must only represent an image or an image activation profile connection.</p>
9F (159) HWI_GROUP_PROF_ABSCAP	<ul style="list-style-type: none"> • For group profile connection (the input ConnectToken represents a group profile connection), requests to change or set whether group absolute capping is enabled for general purpose processors for a group profile. • For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to dynamically change or set whether group absolute capping is enabled for general purpose processors for an LPAR Capacity group. This attribute for this connection type is only available when targeting a z14 GA2 or higher CPC. • For image connection (the input ConnectToken represents an image connection), requests to dynamically change or set whether group absolute capping is enabled for general-purpose processors for a LPAR capacity group in which the target image is a member. This attribute for this connection type is only available when targeting a z13 GA2 or higher CPC. <p>Note: This attribute must be enabled prior to setting the group absolute capping value using attribute HWI_GROUP_PROF_ABSCAPVAL.</p>
A0 (160) HWI_GROUP_PROF_ABSCAPVAL	<ul style="list-style-type: none"> • For group profile connection (the input ConnectToken represents a group profile connection), requests to change or set the maximum general-purpose processor consumption for a group profile. • For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to dynamically change or set the maximum general-purpose processor consumption for an LPAR Capacity group. This attribute for this connection type is only available when targeting a z14 GA2 or higher CPC. • For image connection (the input ConnectToken represents an image connection), requests to dynamically change or set the maximum general-purpose processor consumption for an LPAR Capacity group in which the target image is a member. This attribute is only available when targeting a z13 GA2 or higher CPC.

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
<p>A1 (161) HWI_GROUP_PROF_ICFABSCAP</p>	<ul style="list-style-type: none"> • For group profile connection (the input ConnectToken represents a group profile connection), requests to change or set whether group absolute capping is enabled for ICF processors for a group profile. • For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to dynamically change or set whether group absolute capping is enabled for ICF processor for an LPAR Capacity group. This attribute for this connection type is only available when targeting a z14 GA2 or higher CPC. • For image connection (the input ConnectToken represents an image connection), requests to dynamically change or set whether group absolute capping is enabled for ICF processors for an LPAR Capacity group in which the target image is a member. <p>Note: This attribute must be enabled prior to setting the group absolute capping value using attribute HWI_GROUP_PROF_ICFABSCAPVAL.</p>
<p>A2 (162) HWI_GROUP_PROF_ICFABSCAPVAL</p>	<ul style="list-style-type: none"> • For group profile connection (the input ConnectToken represents a group profile connection), requests to change or set the maximum ICF processor consumption for a group profile. • For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to dynamically change or set the maximum ICF processor consumption for an LPAR Capacity group. This attribute for this connection type is only available when targeting a z14 GA2 or higher CPC. • For image connection (the input ConnectToken represents an image connection), requests to dynamically change or set the maximum ICF processor consumption for an LPAR capacity group in which the target image is a member. This attribute for this connection type is only available when targeting a z13 GA2 or higher CPC.

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
<p>A3 (163) HWI_GROUP_PROF_IFLABSCAP</p>	<ul style="list-style-type: none"> • For group profile connection (the input ConnectToken represents a group profile connection), requests to change or set whether group absolute capping is enabled for IFL processors for a group profile. • For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to dynamically change or set whether group absolute capping is enabled for IFL processors for an LPAR Capacity group. This attribute for this connection type is supported on z14 GA2 or higher CPC. • For image connection (the input ConnectToken represents an image connection), requests to dynamically change or set whether group absolute capping is enabled for IFL processors for an LPAR Capacity group in which the target image is a member. This attribute for this connection type is only available when targeting a z13 GA2 or higher CPC. <p>Note: This attribute must be enabled prior to setting the group absolute capping value using attribute HWI_GROUP_PROF_IFLABSCAPVAL.</p>
<p>A4 (164) HWI_GROUP_PROF_IFLABSCAPVAL</p>	<ul style="list-style-type: none"> • For group profile connection (the input ConnectToken represents a group profile connection), requests to change or set the maximum IFL processor consumption for a group profile. • For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to change or set the maximum IFL processor consumption for an LPAR Capacity group. This attribute for this connection type is only available when targeting a z14 GA2 or higher CPC. • For image connection (the input ConnectToken represents an image connection), requests to dynamically change or set the maximum IFL processor consumption for an LPAR Capacity group in which the target image is a member. This attribute for this connection type is only available when targeting a z13 GA2 or higher CPC.

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
A5 (165) HWI_GROUP_PROF_IIPABSCAP	<ul style="list-style-type: none"> For group profile connection (the input ConnectToken represents a group profile connection), requests to change or set whether group absolute capping is enabled for IIP processors for a group profile. For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to dynamically change or set whether group absolute capping is enabled for IIP processors for an LPAR Capacity group. This attribute for this connection type is only available when targeting a z14 GA2 or higher CPC. For image connection (the input ConnectToken represents an image connection), requests to dynamically change or set whether group absolute capping is enabled for IIP processors for an LPAR Capacity group in which the target image is a member. This attribute for this connection type is only available when targeting a z13 GA2 or higher CPC. <p>Note: This attribute must be enabled prior to setting the group absolute capping value using attribute HWI_GROUP_PROF_IIPABSCAPVAL.</p>
A6 (166) HWI_GROUP_PROF_IIPABSCAPVAL	<ul style="list-style-type: none"> For group profile connection (the input ConnectToken represents a group profile connection), requests to change or set the maximum IIP processor consumption for a group profile. For LPAR Capacity group connection (the input ConnectToken represents an LPAR group connection), requests to dynamically change or set the maximum IIP processor consumption for an LPAR Capacity group. This attribute for this connection type is only available when targeting a z14 GA2 or higher CPC. For image connection (the input ConnectToken represents an image connection), requests to dynamically change or set the maximum IIP processor consumption for an LPAR Capacity group in which the target is a member. This attribute for this connection type is only available when targeting a z13 GA2 or higher CPC.
C9 (201) HWI_IOCDS	Requests to change or set the IOCDS. Note: The input connection token must represent a <i>reset activation profile</i> .
CA (202) HWI_IPL_ADDRESS	Requests to change or set the IPL address. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
CB (203) HWI_IPL_PARM	Requests to change or set the IPL parameter. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CC (204) HWI_IPL_TYPE	Requests to change or set the IPL type for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CD (205) HWI_WW_PORTNAME	Requests to change or set the worldwide port name for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CE (206) HWI_BOOT_PGM_SELECTOR	Requests to change or set the boot program selector for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CF (207) HWI_LU_NUM	Requests to change or set the logical unit number value for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
D0 (208) HWI_BOOTREC_BLK_ADDR	Requests to change or set the boot record logical block address for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
D1 (209) HWI_OPSYS_LOADPARM	Requests to change or set the operating system specific load parameter. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
D2 (210) HWI_GROUP_PROF_NAME	Requests to change or set the name of the group capacity profile that is to be used for the CPC image or image object activated with this profile. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
D3 (211) HWI_LOAD_AT_ACTIVATION	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. Note: The input connection token must represent an <i>image activation profile</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
D4 (212) HWI_CENTRAL_STOR	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. Note: The input connection token must represent an <i>image activation profile</i> .
D5 (213) HWI_RES_CENTRAL_STOR	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. Note: The input connection token must represent an <i>image activation profile</i> .
D6 (214) HWI_EXPANDED_STOR	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. Note: The input connection token must represent an <i>image activation profile</i> .
D7 (215) HWI_RES_EXPANDED_STOR	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. Note: The input connection token must represent an <i>image activation profile</i> .
D8 (216) HWI_NUM_GPP	Requests to change or set the number of dedicated general purpose processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
D9 (217) HWI_NUM_RESGPP	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. Note: The input connection token must represent an <i>image activation profile</i> .
DA (218) HWI_NUM_IFA	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. Note: The input connection token must represent an <i>image activation profile</i> .
DB (219) HWI_NUM_RESIFA	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. Note: The input connection token must represent an <i>image activation profile</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
DC (220) HWI_NUM_IFL	Requests to change or set the number of dedicated integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DD (221) HWI_NUM_RESIFL	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. Note: The input connection token must represent an <i>image activation profile</i> .
DE (222) HWI_NUM_ICF	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. Note: The input connection token must represent an <i>image activation profile</i> .
DF (223) HWI_NUM_RESICF	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. Note: The input connection token must represent an <i>image activation profile</i> .
E0 (224) HWI_NUM_ZIIP	Requests to change or set the number of dedicated z System Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E1 (225) HWI_NUM_RESZIIP	Requests to change or set the number of reserved dedicated z System Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
E6 (230) HWI_NUM_SHARED_IFL	Requests to change or set the number of shared integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
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 connection token must represent an <i>image activation profile</i> .
EA (234) HWI_NUM_SHARED_ZIIP	Requests to change or set the number of shared z System Integrated Information Processors (ZIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
EB (235) HWI_NUM_RES_SHARED _ZIIP	Requests to change or set the number of reserved shared z System Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
EC (236) HWI_BASIC_CPU_AUTH _COUNT_CNTL	Requests to change or set the enablement value of the basic CPU counter facility for the CPC image object activated with this profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
ED (237) HWI_PROBSTATE_CPU _AUTH_COUNT_CNTL	Requests to change or set the enablement value of the Problem state CPU counter facility for the CPC image object activated with this profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
EE (238) HWI_CRYPTACTIVITY_CPU _AUTH_COUNT_CNTL	Requests to change or set the enablement value of the crypto activity CPU counter facility for the CPC image object activated with this profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
EF (239) HWI_EXTENDED_CPU _AUTH_COUNT_CNTL	Requests to change or set the enablement value of the extended CPU counter facility for the CPC image object activated with this profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F0 (240) HWI_COPROCESSOR_CPU _AUTH_COUNT_CNTL	Requests to change or set the enablement value of the coprocessor group CPU counter facility for the CPC image object activated with this profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F1 (241) HWI_BASIC_CPU_SAMPLING _AUTH_CNTL	Requests to change or set the enablement value of the basic CP CPU sampling facility for the CPC image object activated with this profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
F2 (242) HWI_APROF_STORE_STATUS	Requests to change or set the store status function value. This value is only valid if HWI_APROF_LOADTYPE is set to normal. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent a <i>load activation profile</i> .
F3 (243) HWI_APROF_LOADTYPE	Requests to change or set the type of load being requested. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent a <i>load activation profile</i> .
F4 (244) HWI_PROFILE_DESCRIPTION	Requests to change or set the activation profile description. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> , reset activation profile, load activation profile or group profile.
F5 (245) HWI_PROFILE_PARTITION_ID	Requests to change or set the partition identifier for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F6 (246) HWI_OPERATING_MODE	Requests to change or set the operating mode value for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F7 (247) HWI_CLOCK_TYPE	Requests to change or set the clock type assignment (time source setting) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F8 (248) HWI_TIME_OFFSET_DAYS	Requests to change or set the time offset days (the number of days currently set as the offset from the external time source's time of day) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
F9 (249) HWI_TIME_OFFSET_HOURS	Requests to change or set the time offset hours (the number of hours currently set as the offset from the external time source's time of day) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FA (250) HWI_TIME_OFFSET_MINUTES	Requests to change or set the time offset minutes (the number of minutes currently set as the offset from the external time source's time of day) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FB (251) HWI_TIME_OFFSET_INCREASE	Requests to change or set the time offset increase or decrease value for the activation profile. The time offset, as specified in days, hours, and minutes, is increased or decreased from GMT. TRUE means that the time offset is east of GMT. FALSE means that the time offset is west of GMT. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FC (252) HWI_LICCC_VALIDATION _ENABLED	Requests to change or set whether the activation profile must conform to the current Licensed Internal Code Configuration Control (LICCC) configuration. This attribute is only available when targeting a zEnterprise or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FD (253) HWI_GLOBAL_PERFORMANCE _DATA_CONTROL	Requests to change or set whether the logical partition can be used to view the processing unit activity data for all other LPARs activated on the same CPC. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FE (254) HWI_IO_CONFIGURATION _CONTROL	Requests to change or set whether the logical partition can be used to read and write any Input/Output Configuration Data Set (IOCDs) in the configuration. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
100 (256) HWI_LOGICAL_PARTITION _ISOLATION	Requests to change or set whether reconfigurable channel paths assigned to the logical partition are reserved for its exclusive use. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
101-109 (257–264) RESERVED	Reserved for activation profile attributes.

SetTypeValue_Ptr (HWISET only - non-REXX)**SetTypeValue (HWISET only - REXX)**

Supplied parameter

- Type: Pointer (non-REXX), character or character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

Non-REXX:

SetTypeValue_Ptr specifies the address of the value to be set or changed. Some **SetType** requests allow a null value to be set. If a null value is desired, the **SetTypeValue_Ptr** value must be zero.

REXX:

SetTypeValue specifies the value to be set or changed. Some **SetType** requests allow a null value to be set. If a null value is desired, **SetTypeValue** should be set to null ("").

The particular **SetType** determines what data value must be specified. See the following chart as well as the following documentation for more information:

- *IBM z SNMP Application Programming Interfaces* (SB10-7171-06)
- *System z10 and eServer zSeries Application Programming Interfaces* (SB10-7030-09)
- *System z9 and eServer zSeries Application Programming Interfaces* (SB10-7030-08)

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
6 (6) HWI_ACCSTAT	<p>A 4-byte integer type value.</p> <p>For CPC connections, bit values can be set to:</p> <ul style="list-style-type: none"> • 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 <p>For image connections, bit values can be set to:</p> <ul style="list-style-type: none"> • HWMCA_STATUS_OPERATING • HWMCA_STATUS_NOT_OPERATING • HWMCA_STATUS_NOT_ACTIVATED • HWMCA_STATUS_EXCEPTIONS • HWMCA_STATUS_STATUS_CHECK • HWMCA_STATUS_POWERSAVE
7 (7) HWI_APROF	A 16-character activation profile name padded with trailing blanks.
27 (39) HWI_PRUNTYPE	<p>A 4-byte integer type value.</p> <p>HWMCA_DETERMINED_SYSTEM The processor running is dynamically determined by the system.</p> <p>HWMCA_DETERMINED_USER The processor running time is set to a constant value.</p>
28 (40) HWI_PRUNTIME	<p>A 4-byte integer type value.</p> <p>A value between 1 to 100 for the user defined processor running time.</p> <p>Note: This value can only be set if the processor running time type (HWI_PRUNTYPE) is set to HWMCA_DETERMINED_USER.</p>

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
29 (41) HWI_PRUNTSEW	<p>A 4-byte integer type value.</p> <p>HWMCA_TRUE Indicates that an image should lose its share of running time when it enters a wait state.</p> <p>HWMCA_FALSE Indicates that an image should not lose its share of running time when it enters a wait state.</p> <p>Note: This value can only be set if the processor running time type (HWI_PRUNTYPE) is set to HWMCA_DETERMINED_USER.</p> <p>This attribute is only available when targeting a z13 GA2 or lower CPC.</p>
70 (112) HWI_DEFCAP	<p>A 4-byte integer type value.</p> <p>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.</p>
71 (113) HWI_SGPIPW	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the relative amount of shared general purpose processor resources allocated to the CPC image object.</p> <p>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.</p> <p>Note: 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.</p>
72 (114) HWI_SGPIWCAP	<p>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.</p> <p>HWMCA_TRUE Capped</p> <p>HWMCA_FALSE Not capped</p>

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
73 (115) HWI_SGPPWMIN	<p>A 4-byte integer type value.</p> <p>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.</p> <p>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.</p> <p>Note: 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.</p>
74 (116) HWI_SGPPWMAX	<p>A 4-byte integer type value.</p> <p>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.</p> <p>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.</p> <p>Note: 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.</p>
77 (119) HWI_WLM	<p>A 4-byte integer type value.</p> <p>This indicates whether the Workload Manager is allowed to change processor weight-related attributes.</p> <ul style="list-style-type: none"> • HWMCA_TRUE • HWMCA_FALSE <p>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.</p>

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
78 (120) HWI_IFAIPW	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the relative amount of shared integrated facility for applications (IFA) processor resources allocated to the CPC image object.</p> <p>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.</p> <p>Note: 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.</p>
79 (121) HWI_IFAIPWCAP	<p>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.</p> <p>HWMCA_TRUE Capped</p> <p>HWMCA_FALSE Not capped</p>
7A (122) HWI_IFAPWMIN	<p>A 4-byte integer type value.</p> <p>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.</p> <p>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.</p> <p>Note: 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.</p>

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
7B (123) HWI_IFAPWMAX	<p>A 4-byte integer type value.</p> <p>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.</p> <p>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.</p> <p>Note: 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.</p>
7E (126) HWI_IFLIPW	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the relative amount of shared integrated facility for Linux (IFL) processor resources allocated to the CPC image object.</p> <p>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.</p> <p>Note: 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.</p>
7F (127) HWI_IFLIPWCAP	<p>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.</p> <p>HWMCA_TRUE Capped</p> <p>HWMCA_FALSE Not capped</p>
80 (128) HWI_IFLPWMIN	<p>A 4-byte integer type value.</p> <p>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.</p> <p>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.</p> <p>Note: 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.</p>

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
81 (129) HWI_IFLPWMAX	<p>A 4-byte integer type value.</p> <p>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.</p> <p>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.</p> <p>Note: 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.</p>
84 (132) HWI_ICFIPW	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the relative amount of shared internal coupling facility (ICF) processor resources allocated to the CPC image object.</p> <p>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.</p> <p>Note: 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.</p>
85 (133) HWI_ICFIPWCAP	<p>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.</p> <p>HWMCA_TRUE Capped</p> <p>HWMCA_FALSE Not capped</p>
86 (134) HWI_ICFPWMIN	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the minimum relative amount of shared internal coupling facility (ICF) processor resources allocated to the CPC image object.</p> <p>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.</p> <p>Note: 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.</p>

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
87 (135) HWI_ICFPWMAX	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the maximum relative amount of shared internal coupling facility (ICF) processor resources allocated to the CPC image object.</p> <p>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.</p> <p>Note: 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.</p>
8A (138) HWI_IIPW	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the relative amount of shared integrated information processors (IIP) resources allocated to the CPC image object.</p> <p>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).</p> <p>Note: 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).</p>
8B (139) HWI_IIPWCAP	<p>A 4-byte integer type value. This indicates whether the initial processing weight for integrated information processors (IIP) is a limit or a target.</p> <p>HWMCA_TRUE Capped</p> <p>HWMCA_FALSE Not capped</p>
8C (140) HWI_IIPWMIN	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the minimum relative amount of shared integrated information processors (IIP) resources allocated to the CPC image object.</p> <p>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).</p> <p>Note: 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).</p>

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
8D (141) HWI_IIPPWMAX	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the maximum relative amount of shared integrated information processors (IIP) resources allocated to the CPC image object.</p> <p>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).</p> <p>Note: 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).</p>
92 (146) HWI_GROUP_PROFILE_CAPACITY	A 4-byte integer value to represent the workload unit capacity for the group profile associated with an image.
95 (149) HWI_ABSCAP	<p>A 4-byte integer type value. This indicates if absolute capping is in effect for the general purpose processor type.</p> <p>HWMCA_TRUE Enabled</p> <p>HWMCA_FALSE Disabled</p>
96 (150) HWI_ABSCAPVAL	<p>A character string representing the absolute capping value to be set for general purpose processors.</p> <p>The format is generally xxx.yy where xxx is between 0 and 255 and yy is between 00 and 99.</p> <p>For more details and the most current information regarding the format of this data, see IBM Z SNMP Application Programming Interfaces (www.ibm.com/support/pages/node/6018616).</p>
97 (151) HWI_IFAABSCAP	<p>A 4-byte integer type value. This indicates if absolute capping is in effect for the AAP processor type.</p> <p>HWMCA_TRUE Enabled</p> <p>HWMCA_FALSE Disabled</p>
98 (152) HWI_IFAABSCAPVAL	<p>A character string representing the absolute capping value to be set for AAP processors.</p> <p>The format is generally xxx.yy where xxx is between 0 and 255 and yy is between 00 and 99.</p> <p>For more details and the most current information regarding the format of this data, see IBM Z SNMP Application Programming Interfaces (www.ibm.com/support/pages/node/6018616).</p>

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
99 (153) HWI_IFLABSCAP	<p>A 4-byte integer type value. This indicates if absolute capping is in effect for the IFL processors.</p> <p>HWMCA_TRUE Enabled</p> <p>HWMCA_FALSE Disabled</p>
9A (154) HWI_IFLABSCAPVAL	<p>A character string representing the absolute capping value to be set for IFL processors.</p> <p>The format is generally xxx.yy where xxx is between 0 and 255 and yy is between 00 and 99.</p> <p>For more details and the most current information regarding the format of this data, see IBM Z SNMP Application Programming Interfaces (www.ibm.com/support/pages/node/6018616).</p>
9B (155) HWI_ICFABSCAP	<p>A 4-byte integer type value. This indicates if absolute capping is in effect for the ICF processors.</p> <p>HWMCA_TRUE Enabled</p> <p>HWMCA_FALSE Disabled</p>
9C (156) HWI_ICFABSCAPVAL	<p>A character string representing the absolute capping value to be set for ICF processors.</p> <p>The format is generally xxx.yy where xxx is between 0 and 255 and yy is between 00 and 99.</p> <p>For more details and the most current information regarding the format of this data, see IBM Z SNMP Application Programming Interfaces (www.ibm.com/support/pages/node/6018616).</p>
9D (157) HWI_IIPABSCAP	<p>A 4-byte integer type value. This indicates if absolute capping is in effect for the IIP processors.</p> <p>HWMCA_TRUE Enabled</p> <p>HWMCA_FALSE Disabled</p>
9E (158) HWI_IIPABSCAPVAL	<p>A character string representing the absolute capping value to be set for IIP processors.</p> <p>The format is generally xxx.yy where xxx is between 0 and 255 and yy is between 00 and 99.</p> <p>For more details and the most current information regarding the format of this data, see IBM Z SNMP Application Programming Interfaces (www.ibm.com/support/pages/node/6018616).</p>

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
9F (159) HWI_GROUP_PROF_ABSCAP	A 4-byte integer type value. This indicates if group absolute capping is in effect for the general-purpose processor type. HWMCA_TRUE Enabled HWMCA_FALSE Disabled
A0 (160) HWI_GROUP_PROF_ABSCAPVAL	A character string representing the group absolute capping value to be set for general-purpose processors. The format is generally xxx.yy where xxx is between 0 and 255 and yy is between 00 and 99.
A1 (161) HWI_GROUP_PROF_ICFABSCAP	A 4-byte integer type value. This indicates if group absolute capping is in effect for the ICF processor type. HWMCA_TRUE Enabled HWMCA_FALSE Disabled
A2 (162) HWI_GROUP_PROF_ICFABSCAPVAL	A character string representing the group absolute capping value to be set for ICF processors. The format is generally xxx.yy where xxx is between 0 and 255 and yy is between 00 and 99.
A3 (163) HWI_GROUP_PROF_IFLABSCAP	A 4-byte integer type value. This indicates if group absolute capping is in effect for the IFL processor type. HWMCA_TRUE Enabled HWMCA_FALSE Disabled
A4 (164) HWI_GROUP_PROF_IFLABSCAPVAL	A character string representing the group absolute capping value to be set for IFL processors. The format is generally xxx.yy where xxx is between 0 and 255 and yy is between 00 and 99.
A5 (165) HWI_GROUP_PROF_IIPABSCAP	A 4-byte integer type value. This indicates if group absolute capping is in effect for the IIP processor type. HWMCA_TRUE Enabled HWMCA_FALSE Disabled

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
A6 (166) HWI_GROUP_PROF_IIPABSCAPVAL	A character string representing the group absolute capping value to be set for IIP processors. The format is generally xxx.yy where xxx is between 0 and 255 and yy is between 00 and 99.
C9 (201) HWI_IOCDS	A character string representing the IOCDS. A value of an empty string indicates that the reset activation profile will use the currently active IOCDS.
CA (202) HWI_IPL_ADDRESS	A character string representing the IPL address. Note: A value of an empty string indicates that the image activation profile uses the next IPL address set by HCD.
CB (203) HWI_IPL_PARM	A character string representing the IPL parameter. Note: A value of an empty string indicates that the image activation profile uses the next IPL parameter set by HCD.
CC (204) HWI_IPL_TYPE	A 4-byte integer type value. HWMCA_IPLTYPE_STANDARD Indicates that the image activation profile is used to perform a standard load. HWMCA_IPLTYPE_SCSI Indicates that the image activation profile is used to perform a SCSI load. HWMCA_IPLTYPE_SCSIDUMP Indicates that the image activation profile is used to perform a SCSI dump.
CD (205) HWI_WW_PORTNAME	A character string representing the worldwide port name.
CE (206) HWI_BOOT_PGM_SELECTOR	A 4-byte integer type value representing the boot program selector value.
CF (207) HWI_LU_NUM	A character string representing the logical unit number.
D0 (208) HWI_BOOTREC_BLK_ADDR	A character string representing the boot record logical block address.

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
D1 (209) HWI_OPSYS_LOADPARM	A character string representing the operating system specific load parameters.
D2 (210) HWI_GROUP_PROF_NAME	A character string that represents the name of a group capacity profile.
D3 (211) HWI_LOAD_AT_ACTIVATION	A 4-byte integer type value. This indicates whether a load should be done at the end of activation. <ul style="list-style-type: none"> • HWMCA_TRUE • HWMCA_FALSE
D4 (212) HWI_CENTRAL_STOR	A 4-byte integer type value to represent the initial amount of central storage (in megabytes) to be used for the CPC image.
D5 (213) HWI_RES_CENTRAL_STOR	A 4-byte integer type value to represent the reserved amount of central storage (in megabytes) to be used for the CPC image.
D6 (214) HWI_EXPANDED_STOR	A 4-byte integer type value to represent the initial amount of expanded storage (in megabytes) to be used for the CPC image.
D7 (215) HWI_RES_EXPANDED_STOR	A 4-byte integer type value to represent the reserved amount of expanded storage (in megabytes) to be used for the CPC image.
D8 (216) HWI_NUM_GPP	A 4-byte integer type value to represent the number of dedicated general purpose processors to be used for the CPC image.
D9 (217) HWI_NUM_RESGPP	A 4-byte integer type value to represent the number of reserved dedicated general purpose processors to be used for the CPC image.
DA (218) HWI_NUM_IFA	A 4-byte integer value to represent the number of dedicated integrated facility for applications (IFA) processors to be used for the CPC image.

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
DB (219) HWI_NUM_RESIFA	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.
DC (220) HWI_NUM_IFL	A 4-byte integer value to represent the number of dedicated integrated facility for Linux (IFL) processors to be used for the CPC image.
DD (221) HWI_NUM_RESIFL	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.
DE (222) HWI_NUM_ICF	A 4-byte integer value to represent the number of dedicated internal coupling facility (ICF) processors to be used for the CPC image.
DF (223) HWI_NUM_RESICF	A 4-byte integer value to represent the number of reserved dedicated internal coupling facility (ICF) processors to be used for the CPC image.
E0 (224) HWI_NUM_ZIIP	A 4-byte integer value to represent the number of dedicated z System Integrated Information Processors (zIIPs) to be used for the CPC image.
E1 (225) HWI_NUM_RESZIIP	A 4-byte integer value to represent the number of reserved dedicated z System Integrated Information Processors (zIIPs) to be used for the CPC image.
E2 (226) HWI_NUM_SHARED_GPP	A 4-byte integer type value to represent the number of shared general purpose processors to be used for the CPC image.
E3 (227) HWI_NUM_RES_SHARED_GPP	A 4-byte integer type value to represent the number of reserved shared general purpose processors to be used for the CPC image.
E4 (228) HWI_NUM_SHARED_IFA	A 4-byte integer value to represent the number of shared integrated facility for applications (IFA) processors to be used for the CPC image.
E5 (229) HWI_NUM_RES_SHARED_IFA	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.

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
E6 (230) HWI_NUM_SHARED_IFL	A 4-byte integer value to represent the number of shared integrated facility for Linux (IFL) processors to be used for the CPC image.
E7 (231) HWI_NUM_RES_SHARED_IFL	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.
E8 (232) HWI_NUM_SHARED_ICF	A 4-byte integer value to represent the number of shared internal coupling facility (ICF) processors to be used for the CPC image.
E9 (233) HWI_NUM_RES_SHARED_ICF	A 4-byte integer value to represent the number of reserved shared internal coupling facility (ICF) processors to be used for the CPC image.
EA (234) HWI_NUM_SHARED_ZIIP	A 4-byte integer value to represent the number of shared z System Integrated Information Processors (zIIPs) to be used for the CPC image.
EB (235) HWI_NUM_RES_SHARED_ZIIP	A 4-byte integer value to represent the number of reserved shared z System Integrated Information Processors (zIIPs) to be used for the CPC image.
EC (236) HWI_BASIC_CPU_AUTH_COUNT_CNTL	<p>A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC.</p> <p>HWMCA_TRUE The authorization control is enabled.</p> <p>HWMCA_FALSE The authorization control is disabled.</p>
ED (237) HWI_PROBSTATE_CPU_AUTH_COUNT_CNTL	<p>A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC.</p> <p>HWMCA_TRUE The authorization control is enabled.</p> <p>HWMCA_FALSE The authorization control is disabled.</p>
EE (238) HWI_CRYPTACTIVITY_CPU_AUTH_COUNT_CNTL	<p>A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC.</p> <p>HWMCA_TRUE The authorization control is enabled.</p> <p>HWMCA_FALSE The authorization control is disabled.</p>

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
EF (239) HWI_EXTENDED_CPU_AUTH_COUNT _CNTL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE The authorization control is enabled. HWMCA_FALSE The authorization control is disabled.
F0 (240) HWI_COPROCESSOR_CPU_AUTH _COUNT_CNTL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE The authorization control is enabled. HWMCA_FALSE The authorization control is disabled.
F1 (241) HWI_BASIC_CPU_SAMPLING_AUTH _CNTL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE The authorization control is enabled. HWMCA_FALSE The authorization control is disabled.
F2 (242) HWI_APROF_STORE_STATUS	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE Store status is selected. Only allowed if HWI_APROF_LOADTYPE is set to HWMCA_LOADTYPE_NORMAL. HWMCA_FALSE Store status is not selected.
F3 (243) HWI_APROF_LOADTYPE	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_LOADTYPE_NORMAL The Loadtype is set to normal. HWMCA_LOADTYPE_CLEAR The Loadtype is set to clear.
F4 (244) HWI_PROFILE_DESCRIPTION	A 50-character activation profile description. This attribute is only available when targeting a z10 or higher CPC.
F5 (245) HWI_PROFILE_PARTITION_ID	A 4-byte integer type decimal value ranging from 0 to 63. This attribute is only available when targeting a z10 or higher CPC.

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
F6 (246) HWI_OPERATING_MODE	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. <ul style="list-style-type: none"> • HWMCA_GENERAL_OPERATING_MODE • HWMCA_ESA390_OPERATING_MODE • HWMCA_ESA390TPF_OPERATING_MODE • HWMCA_CF_OPERATING_MODE • HWMCA_LINUX_OPERATING_MODE • HWMCA_FMEX_OPERATING_MODE • HWMCA_HMEX_OPERATING_MODE • HWMCA_HMAS_OPERATING_MODE • HWMCA_ZVM_OPERATING_MODE
F7 (247) HWI_CLOCK_TYPE	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. <ul style="list-style-type: none"> • HWMCA_CLOCK_TYPE_STANDARD • HWMCA_CLOCK_TYPE_LPAR
F8 (248) HWI_TIME_OFFSET_DAYS	A 4-byte integer type decimal value ranging from 0 - 999. This attribute is only available when targeting a z10 or higher CPC.
F9 (249) HWI_TIME_OFFSET_HOURS	A 4-byte integer type decimal value ranging from 0 - 23. This attribute is only available when targeting a z10 or higher CPC.
FA (250) HWI_TIME_OFFSET_MINUTES	A 4-byte integer type decimal value. Possible values are 0, 15, 30 or 45. This attribute is only available when targeting a z10 or higher CPC.
FB (251) HWI_TIME_OFFSET_INCREASE	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. <p>HWMCA_TRUE The local time zone is east of GMT.</p> <p>HWMCA_FALSE The local time zone is west of GMT.</p>
FC (252) HWI_LICCC_VALIDATION_ENABLED	A 4-byte integer type value. This attribute is only available when targeting a zEnterprise or higher CPC. <p>HWMCA_TRUE Activation profile must conform to the current LICCC configuration.</p> <p>HWMCA_FALSE Activation profile is not required to conform to the current LICCC configuration.</p>

SetType values in hexadecimal, (decimal), and equate symbol	Values to be specified
FD (253) HWI_GLOBAL_PERFORMANCE _DATA_CONTROL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE Global performance data control is enabled. HWMCA_FALSE Global performance data control is disabled.
FE (254) HWI_IO_CONFIGURATION _CONTROL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE I/O configuration control is enabled. HWMCA_FALSE I/O configuration control is disabled.
100 (256) HWI_LOGICAL_PARTITION _ISOLATION	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE Logical partition isolation control is enabled. HWMCA_FALSE Logical partition isolation control is disabled.

SetTypeValueLen (HWISET only - non-REXX)

Supplied parameter

- Type: Integer
- Length: 4 bytes

SetTypeValueLen specifies the length in bytes of the **SetTypeValue** pointed to by the **SetTypeValue_Ptr** parameter. Some **SetType** requests allow a null value to be set. If a null value is desired, the **SetTypeValueLen** value must be zero.

SetParm_Ptr (HWISET2 only - non-REXX)**SetParm (HWISET2 only - REXX)**

Supplied parameter

- Type: Pointer (non-REXX), stem variable (REXX)
- Length: 4 bytes (non-REXX)

Non-REXX:

SetParm_Ptr specifies the address of a user-defined set structure that contains a list of one or more requested attributes to be set in the following form:

- Connection token representing the attribute to be set.
- The **SetType** attribute to be set.
- The address of the value to be set.
- The length of the value.

The size of the data area pointed to by this parameter must be the size of the data structure mapping a single **SetParm** multiplied by the **NumOfAttributes** parameter. For example, if the **NumOfAttributes** is 4, the data area pointed to by this parameter must be 112 bytes (28 x 4).

The storage area that contains each attribute in the **SetParm** is shown in the following table:

Table 84. Parameters of the (SetParm) structure pointed by the SetParm_Ptr

Field Name	Field Type	Description
Set2_Ctoken	16-character ConnectToken	Set2_Ctoken specifies a ConnectToken that is either be the same value as the HWISET2 ConnectToken parameter (if setting a CPC attribute), or a ConnectToken representing an image or activation profile on the same CPC as the ConnectToken parameter.
Set2_Settype	32-bit signed integer	Set2_Settype specifies the attribute to be set. See the SetType parameter for details on the value choices which can be specified.
Set2_SetValue_Ptr	Pointer	Set2_SetValue_Ptr specifies the address of the value to be set or changed. See the SetTypeValue_Ptr parameter for details on the value ranges that can be specified for each attribute.
Set2_SetValueLen	32-bit signed integer	Set2_SetValueLen specifies the length in bytes of the SetValue pointed to by the Set2_SetValue_Ptr field above. See the SetTypeValueLen parameter for more details.

This table is mapped by the data structure Hwi_Set2_SetParm_Type in the data mappings provided for the various programming languages supported. See [“Syntax” on page 401](#) for more information.

REXX:

SetParm is a compound (stem) variable which contains one or more requested attributes to be set. The stem variable is specified as follows (where x is the user-defined SetParm stem variable and n is the n-th attribute for the request):

- x.0 specifies the number of attributes to be set. The maximum number of attributes allowed is 9 per invocation (Supplied parameter).
- x.n.SET2_CTOKEN specifies a ConnectToken that is either the same value as the HWISET2 **ConnectToken** parameter (if setting a CPC attribute), or a ConnectToken representing an image or activation profile on the same CPC as the **ConnectToken** parameter.
- x.n.SET2_SETTYPE specifies the attribute to be set. See the **SetType** parameter for details on the value choices which can be specified.
- x.n.SET2_SETVALUE specifies the value to be set. See the **SetTypeValue_Ptr** parameter for details on the value ranges that can be specified for each attribute.

NumofAttributes (HWISET2 only - non-REXX)

Supplied parameter

- Type: Integer
- Length: 4 bytes

NumofAttributes specifies the number of attributes to be set. The valid value range is 1 - 9.

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

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.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code that is returned from the Console Application API or the BCPii transport layer.
Diag_Text	Character (12)	Reserved.

See Appendix A, “BCPii communication error reason codes,” on page 819 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0007yyyy' for HWISET or X'0009yyyy' for HWISET2 for one of the following reasons:

<i>Table 85. Reasons for abend X'042', RC X'0007yyyy' for HWISET or X'0009yyyy' for HWISET2</i>	
yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See *z/OS MVS System Codes* for additional information.

Return codes

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

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWI_OK	0 HWI_OK	Meaning: Successful completion. An SMF record has been written. Action: None.

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
100 HWI_CONNECT_TOKEN_INV	256 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified connect token is not valid. This return code indicates one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • 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. <p>Action: Check for probable coding error.</p>
101 HWI_COMMUNICATION_ERROR	257 HWI_COMMUNICATION_ERROR	<p>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.</p> <p>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.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, “BCPii communication error reason codes,” on page 819.</p>

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWI_DIAGAREA_INV	258 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify the specified DiagArea is defined as a 32-byte character field.</p>
103 HWI_CONNECT_TOKEN_INACTIVE	259 HWI_CONNECT_TOKEN_INACTIVE	<p>Meaning: The specified connect token is no longer valid. The connection has been disconnected or it is in the progress of being disconnected.</p> <p>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.</p>
104 HWI_TARGET_CPC_CHANGED	260 HWI_TARGET_CPC_CHANGED	<p>Meaning: The CPC name represented by the specified token is valid but does not represent the same physical machine that was targeted by the initial HWICONN call. All connections that were established prior to the name change can no longer be used.</p> <p>Action: The application should cease using this connect token. If the application intends to target the CPC using the name represented by the specified connect token, it must first reconnect to the CPC before issuing any BCPii service call.</p>
105 HWI_CONNECT_TOKEN_TYPE_INVALID	261 HWI_CONNECT_TOKEN_TYPE_INVALID	<p>Meaning: The specified connect token does not represent a CPC connection.</p> <p>Action: Check for probable coding error. Verify that the specified connect token represents a CPC connection.</p>

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
501 HWI_SETTYPE_INV	1281 HWI_SETTYPE_INV	<p>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:</p> <ul style="list-style-type: none"> • 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. <p>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.</p> <p>See the DiagArea for further diagnostic information.</p> <ul style="list-style-type: none"> • 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	1282 HWI_SETTYPE_VALUE_INV	<p>Meaning: Program error. The requested SetTypeValue to be set or changed is not valid. The system rejects the service call.</p> <p>Action: Check for probable coding error. Validate that the value to which an attribute is being set is appropriate for that attribute.</p>

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
503 HWI_SETTYPE_VALUE_LEN_INV	1283 HWI_SETTYPE_VALUE_LEN_INV	<p>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.</p> <p>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.</p> <p>Note: If the application is setting the value to null for a request that allows a null SetTypeValue, ensure that both the SetTypeValueLen and SetTypeValue_Ptr parameters are set to zero.</p>
504 HWI_SETTYPE_VALUE_INACCESSIBLE	1284 HWI_SETTYPE_VALUE_INACCESSIBLE	<p>Meaning: Program error. The set type value data area is either partially or completely inaccessible by the application, or BCPii, or both.</p> <p>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.</p>
506 HWI_SET_ATTRIBUTE_NOT_SUPPORTED	1286 HWI_SET_ATTRIBUTE_NOT_SUPPORTED	<p>Meaning: The targeted hardware of the HWISET / HWISSET2 request does not recognize the attribute that the user is attempting to set.</p> <p>Action: Verify that the targeted hardware is at a level that supports the type of attribute being set.</p>

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
507 HWI_SET2_SETPARM_INACCESS IBLE	1287 HWI_SET2_SETPARM_INACCESS IBLE	<p>Meaning: Program error. The SetParm data area is either partially or completely inaccessible by the application, the Base Control Program internal interface (BCPii) address space, or both.</p> <p>Action: Check for probable coding error. Consider the following possibilities:</p> <ul style="list-style-type: none"> • The SetParm length could be too small. The size of the SetParm must be at least the product of the NumofAttributes parameter and the length of the data area mapping for each attribute. • The NumofAttributes value can be larger than the number of parameters actually passed.
508 HWI_SET2_NUMOFATTRIB_INV	1288 HWI_SET2_NUMOFATTRIB_INV	<p>Meaning: Program error. The NumofAttributes specified on the call is not valid. The NumofAttributes value must be in the range of 1 to 9.</p> <p>Action: Check for probable coding error. Verify that the NumofAttributes specified is greater than zero and less than or equal to 9.</p>
509 HWI_SET2_CONNECT_TOKEN_I NV	1289 HWI_SET2_CONNECT_TOKEN_I NV	<p>Meaning: Program error. The ConnectToken specified in one of the SetParms is not valid for the specified ConnectToken parameter. Each SetParm ConnectToken must be either the CPC ConnectToken on the HWISET2 call or a child of this CPC ConnectToken.</p> <p>Action: Check for probable coding error. Verify that all SetParm ConnectToken parameters are either the CPC ConnectToken specified on the HWISET2 ConnectToken parameter or are children of that CPC connection.</p>

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
50A HWI_SET2_PARTIAL_UPDATE	1290 HWI_SET2_PARTIAL_UPDATE	<p>Meaning: Program or system error. One or more of the specified attributes were not set. This only applies to hardware levels z13 GA1 and lower.</p> <p>Action: Attempt to roll back the SET request by setting all the requested attributes back to the original values, or attempt to retry the set of the unset attributes. Prior to calling the HWISET2 service, it is good practice to call the HWIQUERY service to retrieve and save the current values of the attributes to be set. BCPii applications can use these original values to compare with the values after the HWISET2 call has completed to determine which attributes have not been set and take the appropriate action. The contents of the DiagArea can also be used to learn some diagnostic information about the first failing attribute.</p>
F00 HWI_NOT_AVAILABLE	3840 HWI_NOT_AVAILABLE	<p>Meaning: HWI is not available, and the system rejects the service request.</p> <p>Action: Start HWI and try the request again.</p>
F01 HWI_AUTH_FAILURE	3801 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state.</p> <p>Action: Check the calling program for a probable coding error.</p>

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F02 HWI_NO_SAF_AUTH	3802 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV. • Define update access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau</i> for a CPC connection, activation profile connection, group profile connectin, or LPAR Capacity group connection. • Define update access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau.imagen ame</i> for an image connection. • Ensure that the referenced Facility Class Profile is RACLIST-specified.
F03 HWI_INTERRUPT_STATUS_INV	3803 HWI_INTERRUPT_STATUS_INV	<p>Meaning: The calling program is disabled. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWI_MODE_INV	3804 HWI_MODE_INV	<p>Meaning: The calling program is not in Task mode, which is the required mode. The system rejects this service request.</p> <p>Action: Check the calling program for a probable error.</p>
F05 HWI_LOCKS_HELD	3805 HWI_LOCKS_HELD	<p>Meaning: The calling program is holding one or more locks. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F06 HWI_UNSUPPORTED_RELEASE	3806 HWI_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects this service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports HWI. Then rerun the calling program.</p>
F07 HWI_UNSUPPORTED_ENVIRONMENT	3807 HWI_UNSUPPORTED_ENVIRONMENT	<p>Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine).</p> <p>Action: Issue the BCPii service from a different execution environment.</p>
FFF HWI_UNEXPECTED_ERROR	4095 HWI_UNEXPECTED_ERROR	<p>Meaning: System error. The service that was called encountered an unexpected error. The system rejects the service call.</p> <p>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 BCPii to attempt the collection of the necessary information to diagnose the error. If so, provide this dump to the IBM support team.</p>

Example

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

```

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

```

The following example uses the HWISET2 service to set two attributes at the same time (one attribute on one image and one attribute on another image):

```
SetParm_Ptr = ADDR(SetParm);
NumOfAttributes = 2;
SetParm(1).Set2_CToken = Image1CToken;
SetParm(1).Set2_SetType = HWI_DEFCAP;
SetParm(1).Set2_SetValue_Ptr = ADDR(DefCapValue1);
SetParm(1).Set2_SetValue_Len = length of DefCapValue1;
SetParm(2).Set2_CToken = Image2CToken;
SetParm(2).Set2_SetType = HWI_DEFCAP;
SetParm(2).Set2_SetValue_Ptr = ADDR(DefCapValue2);
SetParm(2).Set2_SetValue_Len = length of DefCapValue2;
CALL HWISET2 (ReturnCode, CPCConnectToken, SetParm_Ptr,
              NumofAttributes, DiagArea);
```

A REXX programming example for the HWISET service:

```
mySetType = HWI_ACCSTAT          /* AccStat attribute          */
mySetTypeValue = HWMCA_STATUS_EXCEPTIONS

address bcpii
    "hwiset RetCode myConnectToken mySetType mySetTypeValue myDiag."

If (RC <> 0) | (Retcode <> 0) Then
    Do
        Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
        If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
            Do
                Say ' Diag_index=' myDiag.DIAG_INDEX
                Say ' Diag_key=' myDiag.DIAG_KEY
                Say ' Diag_actual=' myDiag.DIAG_ACTUAL
                Say ' Diag_expected=' myDiag.DIAG_EXPECTED
                Say ' Diag_commer=' myDiag.DIAG_COMMERR
                Say ' Diag_text=' myDiag.DIAG_TEXT
            End
        End
    End
```

To code an HWISET2 invocation in REXX, use the following example as a starting point:

```
SetParm.1.SET2_SETTYPE = HWI_IIPPWMIN
SetParm.1.SET2_SETVALUE = 1 + current_IIPPWMIN_Value
SetParm.1.SET2_CTOKEN = ActProfConnectToken

SetParm.2.SET2_SETTYPE = HWI_IIPPWMAX
SetParm.2.SET2_SETVALUE = 1 + current_IIPPWMAX_Value
SetParm.2.SET2_CTOKEN = ActProfConnectToken
SetParm.0 = 2

address bcpii "hwiset2 RetCode myCPC_ConnectToken SetParm. myDiag."

/* Similar error checking as in the previous HWISET example */
```

HWIBeginEventDelivery – Begin delivery of BCPii event notifications

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

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	None
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN

Requirement	Details
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.

z/OS UNIX 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.

CALL statement	Parameters
<code>int HWIBeginEventDelivery</code>	<pre>(*DiagArea ,ConnectToken ,**DeliveryToken)</pre>

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.

Field Name	Field Type	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.

Field Name	Field Type	Description
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code from the failing operation.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See [Appendix A, “BCPii communication error reason codes,”](#) on page 819 for a partial list of the descriptive communication transport error return codes and suggested actions.

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:

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
00000000 HWIUSS_RC_OK	0 HWIUSS_RC_OK	Meaning: Successful completion. Action: None.
00001001 HWIUSS_RC_UNAVAILABLE	4097 HWIUSS_RC_UNAVAILABLE	Meaning: This error is returned for one of the following reasons, which is written to the diag_commerk field of the DiagArea: <ul style="list-style-type: none"> • CEA (Common Event Adapter) communication is unavailable. (reason x'100') • Write access to a socket is denied. (reason x'103') • Services are failing in the CEA Server. (reason x'111') Action: The request is rejected. Confirm that the CEA address space has been started and try the request again.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
00001002 HWIUSS_RC_NO_AUTH	4098 HWIUSS_RC_NO_AUTH	<p>Meaning: The program is not authorized to access CEA services.</p> <p>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. See “Setting up event notification for BCPII z/OS UNIX applications” on page 258 for further information.</p>
00001003 HWIUSS_RC_MAX_CLIENTS	4099 HWIUSS_RC_MAX_CLIENTS	<p>Meaning: The maximum number of CEA clients has been reached.</p> <p>Action: The request is rejected. Determine if other CEA clients can be stopped. If so, try this request again.</p>
00001007 HWIUSS_RC_SAF_NOTDEF_CONNECT	4101 HWIUSS_RC_SAF_NOTDEF_CONNECT	<p>Meaning: The SAF profile CEA.CONNECT is not defined.</p> <p>Action: The request is rejected. Add the CEA.CONNECT profile to the SERVAUTH class and try this request again.</p>
00001008 HWIUSS_RC_COMM_FAILURE	4102 HWIUSS_RC_COMM_FAILURE	<p>Meaning: An error occurred in z/OS UNIX socket processing.</p> <p>Action: The request is rejected. Verify that the file system is properly configured for z/OS UNIX sockets and try this request again.</p>
00001009 HWIUSS_RC_CEA_INTERNAL_ERROR	4103 HWIUSS_RC_CEA_INTERNAL_ERROR	<p>Meaning: An internal CEA processing error has occurred.</p> <p>Action: The request is rejected. Consult the DiagArea for the details about this error. If the error persists, contact the IBM Support Center.</p>
0000100A HWIUSS_RC_INPUT_PTR_IS_NULL	4106 HWIUSS_RC_INPUT_PTR_IS_NULL	<p>Meaning: A null input pointer was found.</p> <p>Action: The request is rejected. Pass a valid pointer to the API and try this request again.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
OFFFFFFF HWIUSS_RC_UNEXPECTED_ERROR	268435455 HWIUSS_RC_UNEXPECTED_ERROR	<p>Meaning: An unexpected error has occurred.</p> <p>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.</p>

Example

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

```
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 z/OS UNIX System Services 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:

Requirement	Details
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.

z/OS UNIX 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.

CALL statement	Parameters
<code>int HWIEndEventDelivery</code>	<pre>(*DiagArea ,*DeliveryToken)</pre>

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.

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

See Appendix A, “BCPii communication error reason codes,” on page 819 for a partial list of the descriptive communication transport error return codes and suggested actions.

*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:

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
00000000 HWIUSS_RC_OK	0 HWIUSS_RC_OK	Meaning: Successful completion. Action: None.
00001001 HWIUSS_RC_UNAVAILABLE	4097 HWIUSS_RC_UNAVAILABLE	Meaning: This error is returned for one of the following reasons, which is written to the diag_commerc field of the DiagArea: <ul style="list-style-type: none"> CEA (Common Event Adapter) communication is unavailable. (reason x'100') Write access to a socket is denied. (reason x'103') Services are failing in the CEA Server. (reason x'111') Action: The request is rejected. Confirm that the CEA address space has been started and try the request again.
00001004 HWIUSS_RC_BAD_DELIVERYTOKEN	4100 HWIUSS_RC_BAD_DELIVERYTOKEN	Meaning: The provided delivery token is not valid. Action: The request is rejected. This is a probable coding error.
00001008 HWIUSS_RC_COMM_FAILURE	4104 HWIUSS_RC_COMM_FAILURE	Meaning: An error occurred in z/OS UNIX socket processing. Action: The request is rejected. Verify that the file system is properly configured for z/OS UNIX sockets and try this request again.
00001009 HWIUSS_RC_CEA_INTERNAL_ERROR	4105 HWIUSS_RC_CEA_INTERNAL_ERROR	Meaning: An internal CEA processing error has occurred. Action: The request is rejected. Consult the DiagArea for the details about this error. If the error persists, contact the IBM Support Center.
0000100A HWIUSS_RC_INPUT_PTR_IS_NULL	4106 HWIUSS_RC_INPUT_PTR_IS_NULL	Meaning: A null input pointer was found. Action: The request is rejected. Pass a valid pointer to the API and try this request again.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
OFFFFFFF HWIUSS_RC_UNEXPECTED_ERROR	268435455 HWIUSS_RC_UNEXPECTED_ERROR	<p>Meaning: An unexpected error has occurred.</p> <p>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.</p>

Example

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

```
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 z/OS UNIX System Services 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:

Requirement	Details
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.

z/OS UNIX 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 qqqqqqqq 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.image* for a ConnectToken representing an image connection.

Note: BCPii requires 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.

CALL statement	Parameters
<pre>int HWIManageEvents</pre>	<pre>(*DiagArea ,*DeliveryToken ,ConnectToken ,EventAction ,EventIDs)</pre>

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.

Field Name	Field Type	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.

Field Name	Field Type	Description
Diag_CommErr	32-bit integer	The returned code from the failing operation.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, “BCPii communication error reason codes,” on page 819 for a partial list of the descriptive communication transport error return codes and suggested actions.

***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 or unregister for BCPii events](#)” on page 318 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 or unregister for BCPii events](#)” on page 318 for the exact syntax.

IBM recommends that an application should at least add the Hwi_Event_BCPIIStatus event if other events are going to be added by the application. The only way to listen for BCPii events in the z/OS UNIX System Services environment is to issue a blocking call to the HwiGetEvent service. If BCPii stops and the Hwi_Event_BCPIIStatus has not been added, the application has no way of knowing of this termination and may hang indefinitely. By at least listening to this event, an application can be aware of BCPii terminations and take the appropriate action.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0004yyyy' because of one of the following reasons:

<i>Table 86. Reasons for abend X'042', RC X'0004yyyy'</i>	
yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.

Table 86. Reasons for abend X'042', RC X'0004yyyy' (continued)

yyyy	Reason
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See [z/OS MVS System Codes](#) for additional information.

Return codes

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

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
00000000 HWIUSS_RC_OK	0 HWIUSS_RC_OK	Meaning: Successful completion. Action: None.
00001000 HWIUSS_RC_HWIEVENT_FAILURE	4096 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	4097 HWIUSS_RC_UNAVAILABLE	Meaning: This error is returned for one of the following reasons, which is written to the diag_commer field of the DiagArea: <ul style="list-style-type: none"> • CEA (Common Event Adapter) communication is unavailable. (reason x'100') • Write access to a socket is denied. (reason x'103') • Services are failing in the CEA Server. (reason x'111') Action: The request is rejected. Confirm that the CEA address space has been started and try the request again.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
00001002 HWIUSS_RC_NO_AUTH	4098 HWIUSS_RC_NO_AUTH	<p>Meaning: This error is returned for one of the following reasons, which is written to the diag_commer field of the DiagArea:</p> <ul style="list-style-type: none"> • The program is not authorized to access CEA services. (reason x'102') • The program is not authorized to monitor the requested event. (reason x'10E') <p>Action: The request is rejected. Determine whether the program needs access to CEA services. If so, grant the required access to the proper resources and try this request again. See “Setting up event notification for BCPii z/OS UNIX applications” on page 258 for further information.</p>
00001004 HWIUSS_RC_BAD_DELIVERYTOKEN	4100 HWIUSS_RC_BAD_DELIVERYTOKEN	<p>Meaning: The provided delivery token is not valid.</p> <p>Action: The request is rejected. This is a probable coding error.</p>
00001006 HWIUSS_RC_SAF_NOTDEF_EVENT	4102 HWIUSS_RC_SAF_NOTDEF_EVENT	<p>Meaning: The SAF profile CEA.SUBSCRIBE.ENF_0068* is not defined.</p> <p>Action: The request is rejected. Add the proper CEA.SUBSCRIBE.ENF_0068* profile to the SERVAUTH class and try this request again.</p>
00001008 HWIUSS_RC_COMM_FAILURE	4104 HWIUSS_RC_COMM_FAILURE	<p>Meaning: An error occurred in z/OS UNIX socket processing.</p> <p>Action: The request is rejected. Verify that the file system is properly configured for z/OS UNIX sockets and try this request again.</p>
00001009 HWIUSS_RC_CEA_INTERNAL_ERROR	4105 HWIUSS_RC_CEA_INTERNAL_ERROR	<p>Meaning: An internal CEA processing error has occurred.</p> <p>Action: The request is rejected. Consult the DiagArea for the details about this error. If the error persists, contact the IBM Support Center.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
0000100A HWIUSS_RC_INPUT_PTR_IS_NULL	4106 HWIUSS_RC_INPUT_PTR_IS_NULL	Meaning: A null input pointer was found. Action: The request is rejected. Pass a valid pointer to the API and try this request again.
OFFFFFFFFF HWIUSS_RC_UNEXPECTED_ERROR	268435455 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.

```
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)
```

HWIGetEvent – Retrieve outstanding BCPii event notifications

Call the HWIGetEvent service to allow a C application running in the z/OS UNIX System Services environment to retrieve outstanding BCPii event notifications.

Description

Environment

The requirements for the callers are:

Requirement	Details
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

Requirement**Details****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.

z/OS UNIX 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.

CALL statement	Parameters
<pre>int HWIGetEvent</pre>	<pre>(*DiagArea ,*DeliveryToken ,*Buffer ,BufferSize ,Timeout ,*BytesNeeded)</pre>

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.

Field Name	Field Type	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.

Field Name	Field Type	Description
Diag_CommErr	32-bit integer	The returned code from the failing operation.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, “BCPii communication error reason codes,” on page 819 for a partial list of the descriptive communication transport error return codes and suggested actions.

***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.

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

Note: If the Hwi_Event_BCPIIStatus event is not registered by the application and the BCPii address space goes down, this service will not be completed if HWIUSS_TIMEOUT_INFINITE was specified. If a numeric value was specified, the service will wake up but neither event data nor indicator that BCPii is not available will be returned. IBM recommends that an application specifies the Hwi_Event_BCPIIStatus event on the HwiManageEvents service call if the HwiGetEvent service is

used. When the HwiGetEvent service returns control to the application, an inspection of which event was received will allow the application to react appropriately when BCPii stops.

*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:

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
00000000 HWIUSS_RC_OK	0 HWIUSS_RC_OK	Meaning: Successful completion. Action: None.
00000001 HWIUSS_RC_PARTIAL_DATA	1 HWIUSS_RC_PARTIAL_DATA	Meaning: The provided buffer was not large enough to contain all the event data. Action: The request is successful. To receive all the event data, buffer the size of which is at least BytesNeeded must be provided.
00000002 HWIUSS_RC_EVENTS_LOST	2 HWIUSS_RC_EVENTS_LOST	Meaning: At least one event was not returned because the program has not been retrieving events timely. 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.
00000003 HWIUSS_RC_TIMEOUT	3 HWIUSS_RC_TIMEOUT	Meaning: No events have occurred in the requested time interval. Action: The request is successful.

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
00001001 HWIUSS_RC_UNAVAILABLE	4097 HWIUSS_RC_UNAVAILABLE	<p>Meaning: This error is returned for one of the following reasons, which is written to the diag_commerc field of the DiagArea:</p> <ul style="list-style-type: none"> • CEA (Common Event Adapter) communication is unavailable. (reason x'100') • Write access to a socket is denied. (reason x'103') • Services are failing in the CEA Server. (reason x'111') <p>Action: The request is rejected. Confirm that the CEA address space has been started and try the request again.</p>
00001004 HWIUSS_RC_BAD_DELIVERYTOKEN	4100 HWIUSS_RC_BAD_DELIVERYTOKEN	<p>Meaning: The provided delivery token is not valid.</p> <p>Action: The request is rejected. This is a probable coding error.</p>
00001005 HWIUSS_RC_SMALL_BUFFER	4101 HWIUSS_RC_SMALL_BUFFER	<p>Meaning: The provided buffer is not large enough to contain the event data.</p> <p>Action: The request is rejected. This is a probable coding error. Provide a larger buffer and try the request again.</p>
00001008 HWIUSS_RC_COMM_FAILURE	4104 HWIUSS_RC_COMM_FAILURE	<p>Meaning: An error occurred in z/OS UNIX socket processing.</p> <p>Action: The request is rejected. Verify that the file system is properly configured for z/OS UNIX sockets and try this request again.</p>
00001009 HWIUSS_RC_CEA_INTERNAL_ERROR	4105 HWIUSS_RC_CEA_INTERNAL_ERROR	<p>Meaning: An internal CEA processing error has occurred.</p> <p>Action: The request is rejected. Consult the DiagArea for the details about this error. If the error persists, contact the IBM Support Center.</p>

Return Code in Hexadecimal Equate Symbol	Return Code in Decimal Equate Symbol	Meaning and Action
0000100A HWIUSS_RC_INPUT_PTR_IS_NULL	4106 HWIUSS_RC_INPUT_PTR_IS_NULL	Meaning: A null input pointer was found. Action: The request is rejected. Pass a valid pointer to the API and try this request again.
FFFFFFF HWIUSS_RC_UNEXPECTED_ERROR	268435455 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.

```
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)
```

Part 9. z/OS client web enablement toolkit

The z/OS client web enablement toolkit provides a set of application programming interfaces (APIs) to enable traditional, native z/OS programs to participate in modern web services applications.

Introduction to the z/OS client web enablement toolkit

You can use web application APIs to create a client/server application using a request-response protocol that can link a client residing anywhere in the world with any web server. Many web applications have evolved to a simpler programming model based on representational state transfer (REST). Governed by a set of architectural constraints, RESTful applications can be much easier to develop, enabling the creation of elegant and secure web applications. RESTful applications typically use the ubiquitous Hypertext Transfer Protocol (HTTP) as the means of communication and either JavaScript Object Notation (JSON) or Extensible Markup Language (XML) as the format of data exchange between the client and server programs.

Applications running in traditional z/OS environments can play the client role of a RESTful web application and initiate a request to a web server residing on z/OS or any other platform that supports web applications. The z/OS client web enablement toolkit provides the following components to enable these applications to more easily participate in the client/server realm:

- A z/OS JSON parser to parse JSON text coming from any source, build new JSON, or add to existing JSON text, as described in [Chapter 20, “The z/OS JSON parser,” on page 467](#)
- A z/OS HTTP/HTTPS protocol enabler that uses interfaces similar to other industry-standard APIs, as described in [Chapter 21, “The z/OS HTTP/HTTPS protocol enabler,” on page 575](#)

While the primary focus of the toolkit is to enable traditional z/OS programs running in environments where these types of services are not as readily available (as compared to a z/OS UNIX or Java™ Virtual Machine (JVM) environment), the services can be run from virtually any environment on a z/OS system. Programs running as a batch job, as a started procedure, or running in almost any address space on a z/OS system can use the toolkit APIs in a similar manner to any standard z/OS APIs provided by the operating system. Furthermore, programs can invoke these APIs in the programming language of choice; the toolkit fully supports C/C++, COBOL, PL/I, REXX and high-level assembler languages.

Chapter 20. The z/OS JSON parser

The JSON parser portion of the z/OS client web enablement toolkit provides a generic, native z/OS JavaScript Object Notation (JSON) parser for z/OS applications.

JSON is a text-based open standard designed for human-readable data interchange. It is derived from the JavaScript scripting language for representing simple data structures and associative arrays, called objects. It is language-independent, with parsers available for many languages. JSON is described in <https://www.json.org/>. The official Internet media type for JSON is `application/json`. The JSON format is often used for serializing and transmitting structured data over a network connection. It is primarily used to transmit data between a server and web application, serving as an alternative to XML. JSON data representation is becoming more pervasive across the industry due to its simplicity and ease of use.

Both JSON and XML are recommended formats for the representation of a request/response body when programming an application following the principles of REST. The XML System Services component of z/OS (z/OS XML) provides a system-level XML parser that is integrated with the base z/OS operating system. (For more information, refer to *z/OS XML System Services User's Guide and Reference*.) It is intended for use by system components, middleware, and applications that need a simple, efficient XML parsing solution. Likewise, the z/OS JSON parser provided with the z/OS client web enablement toolkit is an equivalent system-level, general-purpose JSON parser that is integrated with the z/OS operating system and that works in any native z/OS environment.

JSON basics

The designation of JSON as a great data exchange format lies in its innate simplicity. There are only a few types of data (string, number, boolean, null, array, and object) and its data structures mirror many of the modern programming languages. Many resources are available for you to consult on the Internet to help you gain a basic understanding of the easy-to-use syntax.

An important JSON concept to understand is the idea of an object entry. Within an object, there can be one or more unordered object entries. Each entry consists of a name/value pair. The name, represented by a string enclosed in double quotation marks, identifies the value portion of the pair. The value can be any valid JSON data type.

Figure 28 on page 467 shows an example of JSON text.

```
{
  "firstName": "Steve",
  "lastName": "Jones",
  "age": 46,
  "address": {
    "streetAddress": "123 Anywhere Ave",
    "city": "Poughkeepsie",
    "state": "NY",
    "postalCode": "12601",
    "country": "USA"
  },
  "phone": [
    {
      "type": "mobile",
      "number": "914 555 5555"
    },
    {
      "type": "home",
      "number": "845 555 1234"
    }
  ]
}
```

Figure 28. Example of JSON text

In Figure 28 on page 467, "firstName", "lastName", "age", "address", and "phone" are all names of object entries in the main (root) object of the JSON text. The values assigned to the

"firstName", "lastName", and "age" object entries are all simple data types (string, string, and number, respectively). The named entries "address" and "phone", however, contain a more complex data type as their values. The "address" entry nests additional address details within another object, which contains the portions of an address. The "phone" entry contains an array made up of two array entries. Array entries differ from object entries in that arrays contain only a value (of any JSON data type). In this case, the array values are two objects, each of which group specific types of phone numbers together.

Comments in JSON text

In addition to the well recognized data types, the z/OS JSON parser also tolerates single and multi-line comments defined by the JSON5 Data Interchange Format extension to JSON (<https://spec.json5.org/#comments>). Comments can be either single line or multi-line as shown in the following examples:

```
//single line comment starts with a double-solidus and ends with a line terminator
//the supported line terminator character is a Line Feed<LF>:
//IBM-1047 -> hex 15
//UTF-8 -> hex 0A
```

Figure 29. Example of JSON single line comment

```
/* multi-line comment on a single line, uses a solidus + asterisk form */
```

Figure 30. Example of JSON multi-line comment on one line

```
/* multi-line comments start with solidus + asterisk,
   and can span arbitrarily-many lines,
   until ended with an asterisk + solidus like this */
```

Figure 31. Example of JSON multi-line comment

Comments themselves are allowed to surround and, depending on the style, can even be placed in between the name and the value. However, comments can not be embedded inside either the name or value, and nested comments are not supported.

In the following example, comments are indicated in **bold**:

```

/***** Testing All *****/
* All attributes are tested inside their      *
* condition and displayed with send/log      *
* message. All modifiable attributes are then *
* modified within their own condition def    *
* then standard actions                      *
*****/
{
  /***** Testing CompletionCode *****/
  "condition" : " String(CompletionCode) = '()' "
  , "actions" :
  [
    {
      "action" : "sendMessage" //first action
      , "message": /*CC? */ "CompletionCode: '||String(CompletionCode)"
    }
  ]
}

```

Figure 32. Example of comments on JSON data

Elements of the z/OS JSON parser

The z/OS JSON parser is organized into several types of services:

- **Initialize and terminate:** The purpose of these services is to prepare the memory space required by the z/OS JSON parser or to free it after parser services are no longer needed. The memory allocation created by the initialize service is known as a *parser instance*.

- **Options:** By default, the z/OS JSON parser will tolerate single and multi-line comments as defined by the JSON5 specification. This service allows the application to indicate if the z/OS JSON parser should expect comments in the input JSON text, which may result in better performance results for commented JSON text compared to using the auto detect default setting. The application can also use this service to indicate to the z/OS JSON parser that comments should continue to be rejected as not valid syntax.
- **Codepage:** These services allow the application to indicate to the JSON parser whether their input JSON data is encoded in EBCDIC (codepage 1047) or in UTF-8.
- **Parse:** The parse service performs the following functions:
 - Assigns a particular JSON text to a previously created parser instance
 - Checks the JSON text input for syntax errors
 - Creates an internal representation of the JSON text in the parser instance memory, allowing subsequent parse functions to execute quickly. The internal representation created does not store any information related to the comments that may have been found in the JSON text.
- **Traverse:** These services are designed for applications that might not know the content of the JSON text they are parsing. By using these services, a program can easily traverse the JSON text input, one construct at a time, to discover what data was passed to it. For example, by using a recursive programming methodology, a few simple routines can easily and efficiently traverse the entire stream of data. See [“z/OS JSON parser programming examples” on page 475](#) for examples provided by the toolkit in SYS1.SAMPLIB.
- **Search:** A program might need to find a particular name in a name/value pair that identifies a particular entry of interest. By using the search service and specifying the appropriate search scope, the handle of the matching entry is returned. The handle can then be used on subsequent services to get the information at that location. When retrieving the value portion of the entry, the application must use the HWTJ* API specific to the type of the value being retrieved. The application can use the HWTJGST API to identify the value if it is not known.
- **Create, delete and serialize:** Beyond just reading JSON text created elsewhere and parsing through it, a program might need to modify JSON text. The create service provides the ability to create new JSON text, add entries into existing JSON text, or insert JSON text from another source into the middle of existing JSON text. The delete service provides the ability to delete JSON text.

When the program is done modifying the JSON text, it can then use the serialize service to build the JSON text and place it into a buffer in preparation for sending it to some program over a network or saving it in a local data store. The resulting JSON text will be absent of any comments that may have been present in the original and of any surplus white space.

The general usage of the z/OS JSON parser services in an application follows this order:

1. Create a parser instance and obtain the parser handle (HWTJINIT).
2. Optionally set the encoding (HWTJSENC) and comment toleration (HWTJOPTS) preferences.
3. Associate JSON text with the parser instance (HWTJPARS), which create an internal representation of the data for fast access by subsequent services, or create new JSON text from scratch (HWTJCREN).
4. Traverse to discover the contents of the JSON text, or search to find a particular name in a name/value pair (HWTJSRCH), or add or insert new entries into the existing JSON text (HWTJCREN), or some combination of all of these services.
5. Build the new JSON text if any text was added (HWTJSERI).
6. Free storage used by the parser services (HWTJTERM).

Availability of the z/OS JSON parser

The z/OS JSON parser contained the z/OS client web enablement toolkit is available to virtually any address space. The toolkit is enabled as part of z/OS initialization during IPL time. A message is written to the syslog, which regards the status of the toolkit. Success or failure of toolkit initialization can be found by locating any HWT-prefixed syslog messages, which are issued during IPL.

Syntax, linkage, and programming considerations

The z/OS JSON parser is available to almost any program running in any address space. Almost all z/OS execution environments are supported as well as a wide variety of programming languages.

Programming interface files provided by the JSON parser

Table 87 on page 470 lists the programming interface files provided by the z/OS JSON parser.

Table 87. JSON parser programming interface

Programming language	Programming interface file
C / C++	Include file HWTJIC provided in SYS1.SIEAHDRV.H and under z/OS UNIX /usr/include directory as hwtjic.h
COBOL	Copybook file HWTJICOB provided in SYS1.MACLIB
PL/I	Include file HWTJIPLI provided in SYS1.MACLIB
Assembler	Include file HWTJIASM provided in SYS1.MACLIB
REXX	See “HWTCONST — Initialize predefined variables (REXX)” on page 477 on how to access all the toolkit constants in REXX

Calling formats

Table 88 on page 470 lists specific calling formats for languages that can invoke the z/OS JSON parser callable services.

Table 88. Calling formats for the z/OS JSON parser callable services

Programming language	Calling format
C / C++	<i>Parser_service_name</i> (<i>return_code</i> , <i>parm1</i> , <i>parm2</i> ,...) where the <i>Parser_service_name</i> is all lower case
COBOL	CALL <i>Parser-service-name</i> USING <i>return_code</i> , <i>parm1</i> , <i>parm2</i> ,...
PL/I	CALL <i>Parser_service_name</i> (<i>return_code</i> , <i>parm1</i> , <i>parm2</i> ,...)
Assembler	CALL <i>Parser_service_name</i> (<i>return_code</i> , <i>parm1</i> , <i>parm2</i> ,...),VLIST
REXX	ADDRESS HWTJSON " <i>Parser_service_name return_code parm1 parm2...</i> "

Linkage considerations

There are three ways for a compiled application to find the z/OS JSON parser callable services:

Linkage stub method

(Recommended) Use the linkable stub routine HWTJCSS from SYS1.CSSLIB to link edit your object code. If you attempt to run the parser on a previous release of z/OS that does not support the z/OS JSON parser, this method results in the service call receiving a return code of X' F03 ' (HWTJ_UNSUPPORTED_RELEASE).

Load method

Use the LOAD macro to find the address of the z/OS JSON parsing callable service at run time and then CALL the service. If you attempt to run the parser on a previous release of z/OS that does not support the z/OS JSON parser, this method results in the LOAD macro failing to find the requested service.

Direct linkage method

Code the linkage to the z/OS parser services directly. This can be done if the program first confirms that the level of z/OS contains the toolkit. The following example shows the assembler linkage:

```
L    R14,CVTCSTRT-CVT(R14,0)
L    R14,84(R14,0)
L    R15,4*HWT_SERV_XXXXX(R14,0)
LR   R14,R0
BR   R15
```

In the example, `XXXXX` represents the last five letters of the service you want to call. This requires that the `HWTJKASM` assembler macro be included. If you attempt to run the parser on a previous release of z/OS that does not support the z/OS JSON parser, this method results in the application receiving an abend `X'019'`.

Linkage considerations for high-level language programming

Callers must ensure that the proper linkage is made to the JSON parsing services. The supplied IDF files for the various high-level languages contain the necessary definitions that ensure that the parameter list passed to the JSON parser has the high-order bit turned on for the last parameter. For example, for C, the linkage must be specified as OS linkage, such as:

```
#pragma linkage(HWTJXXXX_CALLTYPE,OS)
```

For PL/I, the entry declaration should have the following options defined:

```
OPTIONS(LINKAGE(SYSTEM))
```

Linkage considerations for assembler language programming

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 - 14 are restored (registers 0, 1 and 15 are not restored).

General programming considerations

Codepage considerations

Input data into the JSON parser may either be in EBCDIC encoding (codepage 1047) or in UTF-8. Any JSON text received by the application in another encoding format must first be converted to one of these supported formats before it can be input to the parser instance via either `parse` (`HWTJPARS`) or `create` (`HWTJCREN`) service calls.

Recovery considerations

The z/OS JSON parser runs in the address space of the application. In addition, all the storage needed by the parser is obtained in the application's address space. Because every application has its own programming environment, it is impossible for the parser to predict the recovery environment required by the application; therefore, the parser does not provide its own recovery. It is imperative that a robust application provide recovery to catch any abnormal ends to parser execution.

When the parser is attempting to access application-provided parameters and those parameters are either inaccessible, point to an inaccessible location, or specify a length that goes beyond the available storage that is obtained by the application, an ABEND occurs and the recovery of the application (if established) receives control. To allow for easier debugging of the problem, when the parser is about to access any application-specified values, the **returnCode** parameter is pre-filled with the HWTJ_INACCESSIBLE_PARM return code and the **diagArea reasonDesc** value is pre-filled with the specifics of which parameter the z/OS JSON parser is attempting to access. If the parser abnormally ends with an SOC4 ABEND code due to an inaccessible parameter, the recovery routine can consult the **returnCode** and **diagArea** values in the callers dynamic storage at the time of the ABEND and see which parameter the parser could not process.

Lastly, if the application catches any abnormal ends (abends) during the z/OS JSON parser execution, subsequent parser calls using the same parser handle can fail with an HWTJ_PARSERHANDLE_INUSE return code. See the action description for this return code for a list of options a program can take when encountering this condition.

REXX Programming Considerations

The toolkit provides a REXX host command environment, HWTJSON, to allow REXX applications to parse and modify JSON strings, as well as search within a JSON string. REXX applications running in TSO/E, System REXX, z/OS UNIX, or ISV-provided REXX environments are supported.

- To initialize the HWTJSON host command environment in your REXX exec, it can be necessary to invoke the **hwtcalls** function at the beginning of your application: `call hwtcalls on`. After this invocation, both the ADDRESS HWTHTTP and ADDRESS HWTJSON host commands will direct API calls to the toolkit.
- To declare all toolkit constants in your REXX exec, use the HWTCONST service as documented in [“HWTCONST — Initialize predefined variables \(REXX\)”](#) on page 477.

Note: There is no REXX IDF (include file) provided by the toolkit.

- The toolkit services allocate task associated resources, which are released at task termination and the termination API calls.
- Handles are not shared among multiple tasks, which can restrict some reentrant REXX environments.
- JSON parser handles can be updated by any of the JSON parser services. The content of these variables should not be modified in any way by the application.
- Verify that all variables have proper content and are exposed if set outside of procedures.
- Variable names specified on toolkit REXX service calls are limited to 40 characters in length.
- REXX does not have unlimited variable content size. In general, a single variable cannot contain more than 16 MB of content. This limits the amount of data that can be sent and received in the JSON parser. If the data required is greater than 16 MB for any of these cases, consider to use one of the high-level languages, which are supported by the toolkit (C/C++, COBOL, PL/I or Assembler).
- The built-in REXX RC variable contains the return code from the REXX HWTJSON host command. This return code indicates the toolkit's acceptance of the supplied REXX HWTJSON host command. The return codes returned in the RC variable are generally unique to the REXX environment. In contrast, the JSON service return code, the variable supplied on the service call itself, is only completed if the RC variable has a value of HWTJ_OK (0) or HWT_REXXParmSyntaxError (1). Possible return codes returned by the toolkit in the RC variable are listed in [Table 89 on page 473](#).
- The **DiagArea** for each REXX service call is returned by using stem variables in the form: `x.HWTJ_ReasonCode`, and `x.HWTJ_ReasonDesc` where `x` is the name of the stem variable specified on the parameter list. If no **DiagArea** information is completed by the toolkit, the value of the **DiagArea** stem-variable on return is all blanks or nulls.

[Table 89 on page 473](#) lists the host command return codes for the REXX environment.

Table 89. Host return codes for REXX

Host return code	Meaning and action
0	<p>Meaning: REXX toolkit host command successful.</p> <p>Action: Consult the toolkit return code on the service call to determine the final result of the request.</p>
1 HWT_REXXParmSyntaxError	<p>Meaning: REXX toolkit host command detects the parameter format is not in the proper form to be accepted.</p> <p>Action: Check for a probable coding error.</p> <ul style="list-style-type: none"> • See the return code on the toolkit service call to determine the reason for the syntax error. • See the REXX programming considerations of the toolkit service to see the exact calling specifications. • Compare the toolkit REXX service call attempted with service call examples in the supplied toolkit REXX programming sample found in SYS1.SAMPLIB. • The DiagArea might contain additional diagnostic information.
2 HWT_REXXUnsupportedService	<p>Meaning: Program error. An unknown toolkit service name was specified on the toolkit REXX host command.</p> <p>Action: Check for a probable coding error. Specify a valid toolkit service name. For example, HWTJPARS.</p>
3 HWT_REXXInvalidNumOfParms	<p>Meaning: Program error. The number of parameters specified on the toolkit REXX host command for the service name specified does not match the number of parameters expected.</p> <p>Action: Check for a probable coding error. See the REXX programming considerations of the toolkit service to see the exact calling specifications. Compare the toolkit REXX service call attempted with service call examples in the supplied toolkit REXX programming sample found in SYS1.SAMPLIB.</p>
4 HWT_REXXStemVarRequired	<p>Meaning: Program error. The toolkit REXX service specified on the toolkit REXX host command is missing one or more required stem variables in the positional parameter list.</p> <p>Action: Check for a probable coding error. See the REXX programming considerations of the toolkit service to see the exact calling specifications. A stem variable parameter must specify a period (.) following the variable name (for example, var.). Also, compare the toolkit REXX service call attempted with service call examples found in the supplied toolkit REXX programming sample found in SYS1.SAMPLIB.</p>
5 HWT_REXXParmNameTooLong	<p>Meaning: Program error. One or more variables specified on the toolkit REXX service call on the toolkit REXX host command is greater than the toolkit maximum REXX variable length (40).</p> <p>Action: Check for a probable coding error. Reduce the variable name lengths on the toolkit REXX service call to be 40 characters or less in length</p>

Table 89. Host return codes for REXX (continued)

Host return code	Meaning and action
6 HWT_REXXInvalidHostEnv	<p>Meaning: System error. The toolkit detected an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
7 HWT_REXXNoStorageForVar	<p>Meaning: System error. Insufficient storage is detected by a SET request from the REXX variable access routine (IRXEXCOM). The system rejects the service call.</p> <p>Action: Ensure that there is sufficient storage available for the toolkit to set REXX variables. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
8 HWT_REXXirxexcom1	<p>Meaning: System error. The REXX variable access routine (IRXEXCOM) used by the toolkit detected an invalid entry condition. This error can be caused by invoking the toolkit REXX host command from a non-REXX application.</p> <p>Action: Ensure to invoke the toolkit REXX host command from a valid REXX exec. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
9 HWT_REXXirxexcom28	<p>Meaning: System error. The REXX variable access routine (IRXEXCOM) detected a language processor environment is missing. This error can be caused by invoking the toolkit from an invalid REXX environment.</p> <p>Action: Ensure that REXX applications invoke the specified toolkit service in a proper REXX environment. TSO/E, System REXX, z/OS UNIX, or ISV-provided REXX environments are supported. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
11 HWT_REXXNoStorage	<p>Meaning: System error. The toolkit could not obtain sufficient storage to satisfy the request.</p> <p>Action: Ensure there is sufficient memory available for REXX command processing. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
13 HWT_REXXInvalidVariable	<p>Meaning: Program error. The toolkit detected one of the variables passed in the parameter list is an invalid REXX variable name.</p> <p>Action: Check for a probable coding error. Verify that all variables passed in the parameter list for the specified service have valid names. See the REXX programming considerations and parameters sections for reference.</p>

Table 89. Host return codes for REXX (continued)

Host return code	Meaning and action
14 HWT_REXXDataTooLongForVar	<p>Meaning: Program error. The REXX variable cannot contain more than 16 megabytes of data.</p> <p>Action: Check for a possible coding error. If the application requires more than 16 megabytes of data, consider using another supported language.</p>
32 HWT_REXXUnexpectedError	<p>Meaning: System error. An unexpected error is detected. The system rejects the service call.</p> <p>Action: A symptom record has been written to LOGREC to record the problem. Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

z/OS JSON parser programming examples

The z/OS JSON parser provides a sample program in many of the supported programming languages to aid in the creation of applications that use the parser functions. Each sample contains examples of how to use almost all of the JSON parser services available in the toolkit. The samples are shipped in SYS1.SAMPLIB. [Table 90 on page 475](#) lists the sample files for each programming language. Additional samples can also be found on the external z/OS client web enablement toolkit github (<https://github.com/IBM/zOS-Client-Web-Enablement-Toolkit>).

Table 90. JSON parser programming sample files

Programming language	Name of sample in SYS1.SAMPLIB
C / C++	HWTJXC1, HWTJXC2
COBOL	HWTJXCB1, HWTJXCB2
PL/I	HWTJXPI1
REXX	HWTJXR1, HWTJXR2, HWTJXR3

z/OS JSON parser callable services

The z/OS JSON parser callable services are grouped under the following categories.

Initialization and termination services

Initialization and termination services deal with the creation and termination of JSON parser instances.

The z/OS JSON parser callable services in this category are:

- “HWTJINIT — Initialize a parser instance” on page 535
- “HWTJTERM — Terminate a parser instance” on page 569

Codepage services

The z/OS JSON parser callable services in this category are:

- “HWTJGENC — Get JSON encoding” on page 506
- “HWTJSENC — Set JSON encoding” on page 551

Usage notes: After an HWTJINIT to obtain a parser handle, and prior to invoking HWTJPARS (or HWTJCEN), no data encoding will be in effect for the JSON parser instance associated with the handle. This could be demonstrated with a call to HWTJGENC, which would return the value

HWTJ_ENCODING_UNKNOWN. Following a successful first HWTJPARS (or HWTJCREN), an encoding is said to be in effect; it remains so until HWTJTERM is invoked to terminate the JSON parser instance. To discourage the comingling of data with different encodings, any attempt to use HWTJSENC to change an encoding currently in effect is considered a usage error and is failed.

An application may choose to invoke HWTJSENC to assert that the text they supply is encoded as either HWTJ_ENCODING_EBCDIC or HWTJ_ENCODING_UTF8. During HWTJPARS, the JSON parser will discover the text's encoding; if the discovered type does not agree with that asserted by the application, HWTJPARS processing will fail immediately (with diagnostics indicating the encoding mismatch). This usage of HWTJSENC is optional; if the application has not asserted an encoding, HWTJPARS processing asserts the finding from its discovery, and proceeds. Following successful parse completion, the application can use HWTJGENC to learn the discovered encoding type, if necessary.

Parse service

The parse service loads a selected JSON text stream into a particular JSON parser instance.

The z/OS JSON parser callable service in this category is:

- [“HWTJPARS — Parse a JSON string” on page 544](#)

Traversal (auto-discovery) parsing services

The purpose of the traversal services is to traverse (discover) the contents of the supplied JSON text in a methodical manner. Typically, these services take a supplied parser handle and either an **entryValueHandle** or **objectValueHandle** parameter (returned by previous invocations of various JSON parser API calls) that points to a specific location within the JSON text.

Note: Throughout the documentation of the z/OS JSON parsing services, the word *entry* is used in the context of a JSON entry. When referring to an object, an entry represents a JSON name or value pair, where the name is a string enclosed in double quotation marks and the value can be specified as any valid JSON type. When referring to an array, an entry represents a value. The value can be any valid JSON type. An object or array can have one or more JSON entries. Multiple entries are separated by commas.

The z/OS JSON parser callable services in this category are:

- [“HWTJGAEN — Get array entry” on page 498](#)
- [“HWTJGBOV — Get boolean value” on page 502](#)
- [“HWTJGJST — Get JSON type” on page 510](#)
- [“HWTJGNUE — Get number of entries” on page 515](#)
- [“HWTJGNUV — Get number value \(non-REXX\)” on page 519](#)
- [“HWTJGOEN — Get object entry” on page 525](#)
- [“HWTJGVAL — Get value” on page 531](#)

Search service

The search service searches for a particular "*name*" in the JSON text.

The z/OS JSON parser callable service in this category is:

- [“HWTJSRCH — Search” on page 561](#)

JSON text creation methods

All of the other methods deal with existing JSON text that an application can process and analyze. However, there is also a need to be able to build JSON text.

Creation methods:

There are two ways to create JSON text using the z/OS web enablement toolkit:

- Create JSON text "from scratch."
- Insert additional JSON text at a particular insertion point.

The toolkit also allows for the JSON text to be added using two different methods:

- **Adding one entry at a time:** By using the provided input parameters, the create service creates syntactically valid JSON text.
- **Adding previously defined JSON text:** The create service first parses the supplied JSON text to verify that the inserted JSON text contains no syntax errors. Then, the insertion point is validated to ensure that the supplied JSON text can logically be inserted at that point. This merges two JSON text streams into one larger text stream.

Deletion method:

It is possible to delete entries from a given JSON text using the z/OS web enablement toolkit.

The new JSON text stream which reflects the modifications can be obtained using the serialize service.

The HWTJSERI service builds the JSON text associated with the specified parser instance by combining the existing JSON text (if any) and any newly added OR DELETED objects or entries.

The z/OS JSON parser callable services in this category are:

- [“HWTJCREN — Create JSON entry” on page 478](#)
- [“HWTJDEL — Delete a JSON entry” on page 490](#)
- [“HWTJSERI — Serialize \(build\) JSON text” on page 555](#)

Comment toleration related service

By default, the z/OS JSON parser will tolerate single and multi-line comments defined in the JSON5 specification.. However the application may want to alter this behavior to either fine tune the parser to always expect comments, or to reject comments as not valid syntax.

The z/OS JSON parser callable service in this category is [HWTJOPTS - set z/OS JSON parser options](#).

Usage Note: The comment toleration preference can only be altered after an HWTJINIT is issued to obtain a parser handle, and must be done prior to invoking HWTJPARS or, when creating a new JSON text, the first invocation of HWTJCREN.

HWTCONST — Initialize predefined variables (REXX)

Call the HWTCONST service to initialize predefined variables in the current REXX variable pool.

Description

This service sets the variables with names prefixed for HWTJ corresponding to the interface definition for the JSON toolkit. This service is helpful when using symbolic names in checking for specific return codes or when specifying constant values in the application. The variable **HWT_CONSTANTS** is set to a list of the interface variable names, which is useful on a procedure expose statement to make the variables visible to a procedure.

Note: This service also sets the variables for the z/OS HTTP Enabler (HWTJH-prefixed) as well. If the REXX application utilizes both the HTTP and JSON parser portions of the toolkit, it is only necessary to call HWTCONST once to initialize all the variables.

Syntax

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

REXX parameters

```
address hwtjson "hwtconst",
               "ReturnCode",
               "DiagArea."
```

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Character string

Contains the return code from the service.

DiagArea.

Returned parameter.

- **Type:** Stem variable

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [“HWTCONST — Initialize predefined variables \(REXX\)”](#) on page 477.

Table 91. Return codes for the HWTCONST service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.

HWTJCREN — Create JSON entry

Call the HWTJCREN service to create or insert new JSON text.

Description

The HWTJCREN service creates or inserts new JSON text into the specified parser instance.

The presence of JSON5-style commentary content among the arguments passed to this service is conditionally tolerated for the following values of EntryValueType only HWTJ_ARRAYVALUETYPE, HWTJ_OBJECTVALUETYPE, and HWTJ_JSONTEXTVALUETYPE.

If the EntryValueType is not in this list, or comment toleration has been explicitly disabled by HWTJOPTS, the presence of commentary will result in an error. See [EntryValueType](#) for further details.

Environment

The requirements for the caller are:

Requirement**Details****Minimum authorization:**

Supervisor state or problem state, any PSW key.

Dispatchable unit mode:

Task or SRB.

Cross memory mode:

Any PASN, any HASN, any SASN.

Requirement	Details
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 470 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJCREN service

All information for the HWTJCREN service applies for REXX requests except:

- EntryName replaces EntryNameAddr and EntryNameLen
- EntryValue replaces EntryValueAddr and EntryValueLen

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJCREN(ReturnCode, ParserHandle, ObjectHandle, EntryNameAddr, EntryNameLen, EntryValueType, EntryValueAddr, EntryValueLen, NewEntryValueHandle, DiagArea);</pre>	<pre>address hwjson "hwtjcren", "ReturnCode", "ParserHandle", "ObjectHandle", "EntryName", "EntryValueType", "EntryValue", "NewEntryValueHandle", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

ObjectHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies an object handle representing a particular JSON object (object or array object) that indicates where the JSON text is to be added (that is, defines the insertion point). The **objectHandle** value is either zero for the root object or an entry value handle (**entryValueHandle**) whose JSON type is HWTJ_OBJECT_TYPE or HWTJ_ARRAY_TYPE.

Because JSON does not allow for the ordering of object entries, a new entry will always be added after the current last entry.

If a new JSON text stream is being built, you must specify an **objectHandle** of zero. This service detects that the root object does not exist and will automatically create it. This service adds an entry to the root object.

Note: With a new JSON text stream, it is not possible for this service to create an object entry consisting of an empty name string ("": *value*), that is, an **entryNameAddr** and **entryNameLen** of zero. If you want to have an object entry with an empty name string as the first object entry, take the following actions:

1. Create the stand-alone root object ({}) by calling the HWTJCREN service and specifying an **entryNameAddr**, **entryNameValue**, **entryValueAddr**, and **entryValueLen** of zero, and an **entryValueType** of HWTJ_OBJECTVALUETYPE.
2. Create an entry within the root object by issuing a second HWTJCREN call and specifying an **entryNameAddr** and **entryNameValue** of zero, the wanted **entryValueType**, and an appropriate value.

If you specify an **objectHandle** of zero and there is existing JSON text that is already parsed within the parser instance, the new text is added as an entry to the root object.

EntryName (REXX)

Supplied parameter.

- **Type:** Character string

Specifies the REXX variable, which contains the name for the entry to be added. Set this variable to a null string if **entryname** is not applicable for the request.

Note: The supplied entry name must be encoded in either EBCDIC (codepage 1047) or UTF-8. Its encoding must be consistent with that of other data input to the JSON parser instance.

EntryNameAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of a buffer that contains the name string associated with the entry to be created. This parameter is required if the **objectHandle** parameter specifies a normal object (JSON type is HWTJ_OBJECT_TYPE). This parameter must be zero if the **objectHandle** parameter specifies an array object (JSON type is HWTJ_ARRAY_TYPE), since array entries have no name.

If the specified **objectHandle** parameter is zero (that is, adding text to the root object) and there is no existing JSON text already parsed within the parser instance, a new JSON text stream is built. In this case, **entryNameAddr** must be zero if the **entryValueType** is either HWTJ_JSONTEXTVALUETYPE or HWTJ_OBJECTVALUETYPE (create a null root object).

Note:

- In the case where a parameter is expected to be set to zero, the application is still expected to pass in a valid parameter variable, however, the value that parameter contains should be zero or blank.

See **Example 1** and **Example 2** for code snippets which illustrate how to set **EntryNameAddr** parameter to zero.

- The supplied entry name must be encoded in either EBCDIC (codepage 1047) or UTF-8. Its encoding must be consistent with that of other data input to the JSON parser instance.

EntryNameLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the "*name*" string to be added at the location specified by **entryNameAddr**. This parameter is required if the **objectHandle** parameter specifies a normal object (JSON type is HWJT_OBJECT_TYPE). This parameter must be zero if the **objectHandle** parameter specifies an array object (JSON type is HWJT_ARRAY_TYPE), since array entries have no name.

If the specified **objectHandle** parameter is zero (that is, adding text to the root object) and there is no existing JSON text already parsed within the parser instance, a new JSON text stream is built. In this case, **entryNameLen** must be zero if the **entryValueType** is either HWTJ_JSONTEXTVALUETYPE or HWTJ_OBJECTVALUETYPE (create a null root object).

EntryValueType

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a constant value that indicates the data type of the value to be added. Valid values are:

HWTJ_OBJECTVALUETYPE

The value to be added is an object. Single and multi-line JSON5 style comments are allowed in the content inside the '{}'.

HWTJ_ARRAYVALUETYPE

The value to be added is an array. Single and multi-line JSON5 style comments are allowed when the array element value is a JSON Object, for example, "Array" : [{}, {}, {}]. In the case of a JSON Object entry value, the comments are allowed in the content inside the '{}'.

HWTJ_STRINGVALUETYPE

The value to be added is a string.

HWTJ_NUMVALUETYPE

The value to be added is a number.

HWTJ_TRUEVALUETYPE

The value to be added is a boolean value of TRUE.

HWTJ_FALSEVALUETYPE

The value to be added is a boolean value of FALSE.

HWTJ_NULLVALUETYPE

The value to be added is a NULL type.

HWTJ_JSONTEXTVALUETYPE

The value to be added is another JSON text stream. Single and multi-line JSON5 style comments are allowed outside and inside of the '{}'. For example:

```
/* I am a JSON TEXT VALUE with comments */
{
  /** comment1 *****/
  "message1": "JSON is great" /** comment 2
  , "message2": "but it can be a lot sometimes" /* comment 3 */
}
/* comment 4 after the json object */
```

If **entryValueType** is HWTJ_STRINGVALUETYPE, HWTJ_NUMVALUETYPE, or HWTJ_JSONTEXTVALUETYPE, the **entryValueAddr** and **entryValueLen** parameters must specify

the actual value in the "*name*" : *value* pair. If the **entryValueType** is not one of these three values, **entryValueAddr** and **entryValueLen** must be set to zero.

Note: In the case where a parameter is expected to be set to zero, the application is still expected to pass in a valid parameter variable, however, the value that parameter contains should be zero or blank. See [Example 1](#) and [Example 2](#) for code snippets which illustrate how to set **EntryNameAddr** parameter to zero.

EntryValue (REXX)

Supplied parameter.

- **Type:** Character string

Specifies the REXX variable, which contains the value of the entry to be added. Set this variable to a null string if **entryvalue** is not applicable for the request.

EntryValueAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of a buffer that contains the value to be added. This is only valid if **entryValueType** is HWTJ_STRINGVALUETYPE, HWTJ_NUMVALUETYPE, or HWTJ_JSONTEXTVALUETYPE. For all other **entryValueType** values, specify zero.

Notes:

1. In the case where a parameter is expected to be set to zero, the application is still expected to pass in a valid parameter variable, however, the value that parameter contains should be zero or blank. See [Example 1](#) and [Example 2](#) for code snippets which illustrate how to set **EntryNameAddr** parameter to zero.
2. If the **entryValueType** is HWTJ_JSONTEXTVALUETYPE, the parser maintains a binding to the buffer specified by this parameter even after the call to the HWTJCREN service completes. The buffer must not be freed or reused until the caller has completed all parsing functions associated with this JSON text string.
3. All **entryValueType** values must be encoded in EBCDIC (codepage 1047) or UTF-8. Their encoding must be consistent with that of other data input to the JSON parser instance. If no encoding is currently in effect (the entry being created is the first data input to the JSON parser instance, and no encoding has been asserted via HWTJSENC), the parser instance will attempt to discover the encoding of the value data, defaulting to EBCDIC when necessary. User assertion of encoding prior to entry creation is recommended, to ensure that the entry value data is properly handled.

EntryValueLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the value to be added at the location specified by the **entryValueAddr** parameter. This is only valid if **entryValueType** is HWTJ_STRINGVALUETYPE, HWTJ_NUMVALUETYPE, or HWTJ_JSONTEXTVALUETYPE. For all other **entryValueType** values, specify zero.

NewEntryValueHandle

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

The handle that represents the new object entry that was created.

Note: If **entryValueType** is either HWTJ_OBJECTVALUETYPE or HWTJ_ARRAYVALUETYPE, the returned handle should be treated as an object handle (**objectHandle**) on subsequent requests, since the entry value is an object.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0001yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 92 on page 483](#).

Table 92. Return codes for the HWTJCREN service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 92. Return codes for the HWTJCREN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>

Table 92. Return codes for the HWTJCREN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
104 HWTJ_HANDLE_INV	260 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. (For example, uninitialized handle or a reference to a deleted entry.) Only pass one of the following values:</p> <ul style="list-style-type: none"> • An object handle or entry value handle on the objOrEntryHandle parameter that was returned by a prior z/OS JSON parser service call. • A value of zero for the root object.
105 HWTJ_HANDLE_TYPE_ERROR	261 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified objectHandle does not represent an object or array object (JSON type of HWTJ_OBJECT_TYPE or HWTJ_ARRAY_TYPE).</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an objectHandle value that represents an object handle or array object handle.</p>
108 HWTJ_WORKAREA_TOO_SMALL	264 HWTJ_WORKAREA_TOO_SMALL	<p>Meaning: Program error. The work area (which contains the internal representation of the entire JSON text, including the JSON constructs, which the HWTJCREN service attempted to add) is not large enough to satisfy this HWTJCREN request. The parser requires a work area that is larger than the maxParserWorkAreaSize value that was specified on the HWTJINIT service.</p> <p>Action: Check for a probable coding error. IBM recommends specifying a maxParserWorkAreaSize value of 0 (unlimited work area size available to the parser). If the application cannot specify this value, then modify the invocation of the HWTJINIT service to specify a larger maxParserWorkAreaSize value based on the recommendations given in the ReasonDesc section of the diagArea and in the description of the HWTJINIT maxParserWorkAreaSize parameter.</p>

Table 92. Return codes for the HWTJCREN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
109 HWTJ_PARSE_ERROR	265 HWTJ_PARSE_ERROR	<p>Meaning: Supplied JSON text error. The JSON text passed by the caller to the HWTJCREN service contains a syntax error. The specified entryValueAddr points to a buffer containing JSON text to be added to the parser instance (entryValueType is HWTJ_JSONTEXTVALUETYPE). However, the supplied JSON text has one or more syntax errors.</p> <p>Note: This error can also be generated if the specified JSONTextLen value is greater than the actual length of the JSON text and there are non-null characters after the JSON text.</p> <p>Action: Check the diagArea for a complete explanation of the error. The reasonCode portion of the diagArea pinpoints the reason for the parse failure, while the ReasonDesc portion points to the exact location in the supplied JSON text where the parser detected a JSON syntax error.</p>
10B HWTJ_CANNOT_OBTAIN_WORKAREA	267 HWTJ_CANNOT_OBTAIN_WORKAREA	<p>Meaning: System error. The Storage Obtain service could not obtain the work area storage as required by the z/OS JSON parser during the HWTJCREN service call.</p> <p>Action: Consult the diagArea for the Storage Obtain failure return code and additional information found in the ReasonDesc section. Check to see if there is sufficient memory available in order for the parser to obtain the necessary amount of work area. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
201 HWTJ_JCREN_ENTRYNAMEADDR_INV	513 HWTJ_JCREN_ENTRYNAMEADDR_INV	<p>Meaning: Program error. The caller specified a value of zero for the address of the entry name buffer when the specified objectHandle was of type HWTJ_OBJECT_TYPE or specified a nonzero value when the specified objectHandle was of type HWTJ_ARRAY_TYPE.</p> <p>Action: Check for a probable coding error. Specify the actual address of the buffer containing the entry name to be added if the objectHandle is of type HWTJ_OBJECT_TYPE. Specify zero if the objectHandle is of type HWTJ_ARRAY_TYPE.</p> <p>Note: Specifying a bad entry name buffer address other than zero can result in the parser terminating with an X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>

Table 92. Return codes for the HWTJCREN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
202 HWTJ_JCREN_ENTRYNAMELEN_INV	514 HWTJ_JCREN_ENTRYNAMELEN_INV	<p>Meaning: Program error. The caller specified a value of zero as the length of the entry name buffer when the specified objectHandle was of type HWTJ_OBJECT_TYPE or specified a nonzero value when the specified objectHandle was of type HWTJ_ARRAY_TYPE.</p> <p>Action: Check for a probable coding error. Specify the actual length of the entry name buffer if the objectHandle is of type HWTJ_OBJECT_TYPE. Specify zero if the objectHandle is of type HWTJ_ARRAY_TYPE.</p> <p>Note: Specifying a bad entry name buffer length other than zero can result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
203 HWTJ_JCREN_ENTRYVALUEADDR_INV	515 HWTJ_JCREN_ENTRYVALUEADDR_INV	<p>Meaning: Program error. The caller specified one of the following values:</p> <ul style="list-style-type: none"> A value of zero for the address of the entry value buffer when the specified entryValueType was HWTJ_STRINGVALUETYPE, HWTJ_NUMVALUETYPE, HWTJ_JSONTEXTVALUETYPE A nonzero value for the address of the entry value buffer when the specified entryValueType was HWTJ_OBJECTVALUETYPE, HWTJ_ARRAYVALUETYPE, HWTJ_TRUEVALUETYPE, HWTJ_FALSEVALUETYPE, or HWTJ_NULLVALUETYPE <p>Action: Check for a probable coding error. Specify the correct value based on the selected entryValueType. The parser enforces a zero value for this parameter whenever a value is specified that is incompatible with the entry value type the caller is trying to create.</p> <p>Note: Specifying a bad entry value buffer address other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>

Table 92. Return codes for the HWTJCREN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
204 HWTJ_JCREN_ENTRYVALUELEN_INV	516 HWTJ_JCREN_ENTRYVALUELEN_INV	<p>Meaning: Program error. The caller specified one of the following values:</p> <ul style="list-style-type: none"> A value of zero for the length of the entry value buffer when the specified entryValueType was HWTJ_STRINGVALUETYPE, HWTJ_NUMVALUETYPE, HWTJ_JSONTEXTVALUETYPE A nonzero value for the length of the entry value buffer when the specified entryValueType was HWTJ_OBJECTVALUETYPE, HWTJ_ARRAYVALUETYPE, HWTJ_TRUEVALUETYPE, HWTJ_FALSEVALUETYPE, or HWTJ_NULLVALUETYPE <p>Action: Check for a probable coding error. Specify the correct value based on the selected entryValueType.</p> <p>Note: Specifying a bad entry value buffer length other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
205 HWTJ_JCREN_ENTRYVALUETYPE_INV	517 HWTJ_JCREN_ENTRYVALUETYPE_INV	<p>Meaning: Program error. The caller specified an invalid entryValueType parameter.</p> <p>Action: Check for a probable coding error. Change the entryValueType to one of the valid values.</p>
206 HWTJ_JCREN_ENTRYNAME_INV	518 HWTJ_JCREN_ENTRYNAME_INV	<p>Meaning: Program error. The caller specified an entry name, which contains a syntax error. (This is a valid return code for all types of the objectHandle parameter except HWTJ_ARRAY_TYPE.)</p> <p>Action: Check for a probable coding error. Consult <i>Introducing JSON</i> (json.org) or other JSON specifications to determine the correct syntax of a string or number value in a name or value pair.</p> <p>Note: The z/OS JSON parser adds the beginning and ending double quotation marks to all entry names. Any quotation marks contained in the entry name is flagged with this error.</p>

Table 92. Return codes for the HWTJCREN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
207 HWTJ_JCREN_ENTRYVALUE_INV	519 HWTJ_JCREN_ENTRYVALUE_INV	<p>Meaning: Program error. The caller specified an entry value, which contains a syntax error. (This is a valid return code for entryValueType parameter values HWTJ_STRINGVALUETYPE and HWTJ_NUMVALUETYPE.)</p> <p>Action: Check for a probable coding error. Consult Introducing JSON (json.org) or other JSON specifications to determine the correct syntax of a string or number value in a name or value pair.</p> <p>Note: The z/OS JSON parser adds the beginning and ending double quotation marks to all entry values with a type of HWTJ_STRINGVALUETYPE. Any quotation marks contained in the entry value buffer is flagged with this error.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Examples

Example 1: C code adding an object into an array

Part of C code that illustrates how to add an object into an array. See SYS1.SAMPLIB(HWTJXC1) for full sample.

```

/* Declare a handle to hold the resulting employee object. */
HWTJ_HANDLE_TYPE new_employee_handle;

/* HWTJCREN requires a name address of 0 in this case. */
char *entryNameAddr = 0;

/* This call to HWTJCREN inserts the JSON data for the "Hank Hacker"
 * employee entry directly into the employee array. The

```

```

* employee-array-handle parameter specifies the "insertion point" -- the
* object/array that will contain this entry. The entryValueType
* parameter specifies the format of the incoming data. In this case,
* HWTJ_JSONTEXTVALUETYPE is specified because the data to be inserted is
* valid JSON text. Note that the original JSON string is not modified by
* HWTJCREN. The serialize service (HWTJSERI) can be used to obtain a new
* JSON text string which will include the new "Hank Hacker" array element
*/
hwtjcrcn(&return_code,
        parser_instance,
        employee_array, /* handle to the insertion point (input) */
        (char *)&entryNameAddr, /* name of the object (input) */
        0, /* length of the name (input) */
        HWTJ_JSONTEXTVALUETYPE, /* type of data to be inserted (input) */
        (char *)&new_employee_json, /* JSON text string address (input) */
        strlen(new_employee_json), /* JSON text string length (input) */
        &new_employee_handle, /* handle to the new entry (output) */
        &diag_area);

```

Example 2: REXX code adding an object into an array

Part of REXX code that illustrates how to add an object into an array. See SYS1.SAMPLIB(HWTJXR1) for full sample.

```

/*****
/* Insert the input json body "all at once" as a new array element. Since it
/* is not a name:value pair (rather, an update to the value of the array), we
/* supply an empty newEntryName (in contrast with the next insert, to come). */
*****/
newEntryName = ''
newEntryValue = newEmployeeJsonText

ReturnCode = -1
DiagArea. = ''
address hwtjson "hwtjcrcn ",
               "ReturnCode ",
               "parserHandle ",
               "employeeArrayHandle ",
               "newEntryName ",
               "HWTJ_JSONTEXTVALUETYPE ",
               "newEntryValue ",
               "handleOut ",
               "DiagArea."

```

HWTJDEL — Delete a JSON entry

Call the HWTJDEL service to delete content from a JSON text.

Description

The HWTJDEL service deletes a name:value pair from a JSON object or a value from a JSON array. HWTJDEL accepts as input a handle representing the JSON value to be deleted, and a handle representing the containing JSON object or JSON array. The value handle can represent any of the following JSON types:

- *number* or *string*
- Literals *true*, *false*, or *null*
- JSON object or JSON array

There are several ways to obtain the handles required to delete a given JSON value, depending on the type of value being deleted. See [“Examples” on page 494](#) at the end of this service for pseudo-code examples.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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 470](#) for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJDEL service

All information for the HWTJDEL service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJDEL(ReturnCode, ParserHandle, ObjectHandle, EntryValueHandle, DiagArea);</pre>	<pre>address hwjson "hwtjdel", "ReturnCode", "ParserHandle", "ObjectHandle", "EntryValueHandle", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

ObjectHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies an object handle representing a particular JSON object (object or array object) that contains the object or array entry to be deleted. The **objectHandle** value is either zero for the root object or an entry value handle (**entryValueHandle**) whose JSON type is HWTJ_OBJECT_TYPE or HWTJ_ARRAY_TYPE.

EntryValueHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a handle representing a particular entry to be deleted. The entry value handle must represent an entry contained within the JSON object or JSON array represented by **ObjectHandle**. If **ObjectHandle** represents a JSON object, **EntryValueHandle** must represent a member of the object. If **ObjectHandle** represents a JSON array, **EntryValueHandle** must represent an element of the array.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'000Eyyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

Table 93. Return codes for the HWTJDEL service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 93. Return codes for the HWTJDEL service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	260 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle or entryValueHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. (For example, uninitialized handle or a reference to a deleted entry.)</p>

Table 93. Return codes for the HWTJDEL service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
105 HWTJ_HANDLE_TYPE_ERROR	261 HWTJ_HANDLE_TYPE_ERROR	Meaning: Program error. The specified objectHandle does not represent an object or array object (JSON type of HWTJ_OBJECT_TYPE or HWTJ_ARRAY_TYPE). Action: Check for a probable coding error. Specify a handle that represents a JSON object or JSON array.
901 HWTJ_JDEL_ENTRY_NOTE_FOUND	2305 HWTJ_JDEL_ENTRY_NOTE_FOUND	Meaning: Program error. The specified entryValueHandle does not represent an entry contained within the object specified by objectHandle . Action: Check for a probable coding error.
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	Meaning: Program error. The calling program is disabled. The system rejects the service request. Action: Check the calling program for a probable coding error.
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	Meaning: The operating system level does not support this service. The system rejects the service request. Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Examples

The following pseudo-code examples demonstrate several uses of HWTJDEL.

Example 1:

Given the following JSON text:

```
{
  "foo": {
    "mood": "happy",
    "color": "red",
    "bling": "baz"
  },
  "bar": [ "bag", 3, true ]
}
```

To delete the "mood" entry from the "foo" object, first call HWTJSRCH to obtain a handle to the "foo" object.

```
objectName = "foo"
rootObject = 0      /* search the root object */
startingHandle = 0  /* start with the first member */

HWTJSRCH (
    returnCode,
    parserHandle,
    HWTJ_SEARCHTYPE_GLOBAL,
    objectName,
    length(objectName),
    rootObject,
    startingHandle,
    targetObjectHandle,
    DiagArea
);
```

The output handle (**targetObjectHandle**) can then be used on a subsequent call to HWTJSRCH to find the "mood" entry.

```
entryName = "mood"

HWTJSRCH (
    returnCode,
    parserHandle,
    HWTJ_SEARCHTYPE_OBJECT,
    entryName,
    length(entryName),
    targetObjectHandle,
    startingHandle,
    targetEntryHandle,
    diagArea
);
```

Having obtained the required **targetObjectHandle** and **targetEntryHandle**, a call to HWTJDEL would look similar to this:

```
HWTJDEL (
    returnCode,
    parserHandle,
    targetObjectHandle,
    targetEntryHandle,
    diagArea
);
```

A subsequent call to HWTJSERI would return a JSON text string reflecting the delete operation.

```
{
  "foo":{
    "color":"red",
    "bling":"baz"
  },
  "bar":[ "bag", 3, true]
}
```

Example 2:

Deleting an element from a JSON array would require a slightly different sequence of operations.

Given the following JSON text:

```
{
  "bar":[ "bag", 3, true, {"a": "somewhere"} ],
  "bling": "blam",
}
```

```
    "pi": 3.14159
}
```

To delete the fourth entry (the object with a single entry named "a") from the "bar" array , first call HWTJSRCH to obtain a handle to the "bar" array.

```
arrayName = "bar"
rootObject = 0 /* Search the root */
startingHandle = 0 /* Start at the first member */
targetArrayHandle = 0

HWTJSRCH (
    returnCode,
    parserHandle,
    HWTJ_SEARCHTYPE_GLOBAL,
    arrayName,
    length(arrayName),
    rootObject,
    startHandle,
    targetArrayHandle,
    DiagArea
);
```

The output handle (**targetArrayHandle**) can then be used on a subsequent call to HWTJGAEN to retrieve a handle to the fourth array value.

```
entryIndex = 3 /* Note that array entries are zero-indexed */

HWTJGAEN (
    returnCode,
    parserHandle,
    targetArrayHandle,
    entryIndex,
    targetEntryHandle,
    diagArea
);
```

Having obtained the required **targetArrayHandle** and **targetEntryHandle**, a call to HWTJDEL would look similar to this:

```
HWTJDEL (
    returnCode,
    parserHandle,
    targetArrayHandle,
    targetEntryHandle,
    diagArea
);
```

A subsequent call to the HWTJSERI service would return a JSON text string reflecting the delete operation.

```
{
    "bar": [ "bag", 3, true ],
    "bling": "blam",
    "pi": 3.14159
}
```

HWTJESCT — Encode or decode escape sequences (REXX)

Call the HWTJESCT service to encode or decode escape sequences in a JSON text stream.

Description

This service is a simple utility program, which transforms non-conforming JSON text (not properly escaped) into conforming JSON text (encode). This service can also be used to non-escape conforming JSON text into text, which can be easily read or displayed (decode).

Note: This service currently supports only EBCDIC (codepage 1047) data input.

Syntax

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

REXX parameters
<pre>address hwtjson "hwtjesct", "ReturnCode", "RequestType", "EnOrDeSource", "EnOrDeTarget", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Character string

Contains the return code from the service.

RequestType

Supplied parameter.

- **Type:** Character string

A REXX variable that contains the request type. The request type can be set by either the **HWTJ_ENCODE** or the **HWTJ_DECODE** string.

EnOrDeSource

Supplied parameter.

- **Type:** Character string

A REXX variable that identifies the string to encode or decode.

EnOrDeTarget

Returned parameter.

- **Type:** Character string

A REXX variable that the service sets to the encoded or decoded JSON text.

DiagArea.

Returned parameter.

- **Type:** Stem variable

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [“HWTJESCT — Encode or decode escape sequences \(REXX\)”](#) on page 497.

Table 94. Return codes for the HWTJESCT service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.

HWTJGAEN — Get array entry

Call the HWTJGAEN service to obtain a handle for an array entry.

Description

The HWTJGAEN service returns the handle of a particular array entry (**entryValueHandle**) as specified by the index of that array entry (**arrayEntryIndex**). The returned handle is used on subsequent calls to service calls to inquire about the value of this particular array entry.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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 470 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGAEN service

All information for the HWTJGAEN service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJGAEN(ReturnCode, ParserHandle, ObjectHandle, ArrayEntryIndex, EntryValueHandle, DiagArea);</pre>	<pre>address hwtjson "hwtjgaen", "ReturnCode", "ParserHandle", "ObjectHandle", "ArrayEntryIndex", "EntryValueHandle", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

ObjectHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a particular JSON array object. The **objectHandle** must have a JSON type of HWTJ_ARRAY_TYPE.

ArrayEntryIndex

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies the index of the n^{th} entry of the array object specified by **objectHandle**.

Note: This is a zero-origin index, meaning that the first entry in the array has an index of zero; the n^{th} entry has an index of $(n - 1)$.

EntryValueHandle

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

The handle representing the particular entry value selected in the JSON array object.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that may contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0002yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 95 on page 500](#).

Table 95. Return codes for the HWTJGAEN service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 95. Return codes for the HWTJGAEN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one will be allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service call abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	260 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. (For example, uninitialized handle or a reference to a deleted entry.) Only pass an objectHandle parameter that was returned by a prior z/OS JSON parser service call.</p>

Table 95. Return codes for the HWTJGAEN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
105 HWTJ_HANDLE_TYPE_ERROR	261 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified objectHandle does not represent an array object (JSON type of HWTJ_ARRAY_TYPE).</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an objectHandle value that represents an array handle.</p>
107 HWTJ_INDEX_OUT_OF_BOUNDS	262 HWTJ_INDEX_OUT_OF_BOUNDS	<p>Meaning: Program error. The index value specified by the arrayEntryIndex parameter is greater than the number of entries in the array.</p> <p>Action: Check for a probable coding error.</p> <ul style="list-style-type: none"> • Issue the HWTJGNUE (Get number of entries) service to determine the upper bound for the number of array entries. • Remember that the number of entries returned is the actual count of entries, but accessing a particular entry uses zero-origin indexing. To iterate through all the elements, the application must start at the first entry (index 0) up to the number of entries, minus 1.
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJGBOV – Get boolean value

Call the HWTJGBOV service to obtain the boolean value of an entry.

Description

The HWTJGBOV service returns the boolean value of the entry, which is associated with a specified entry handle (**entryValueHandle**).

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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 470 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGBOV service

All information for the HWTJGBOV service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJGBOV(ReturnCode, ParserHandle, EntryValueHandle, BooleanValue DiagArea);</pre>	<pre>address hwtjson "hwtjgbov", "ReturnCode", "ParserHandle", "EntryValueHandle", "BooleanValue", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

EntryValueHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a handle representing a particular entry in a JSON object that has a JSON type of boolean.

BooleanValue

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 1 byte (non-REXX)

A value contains one of the following values: HWTJ_true or HWTJ_false. Constants are provided in the associated interface definition file (IDF).

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0003yyyy' for one of the following reasons:

yyyy**Reason****0000**

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 96 on page 504](#).

Table 96. Return codes for the HWTJGBOV service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 96. Return codes for the HWTJGBOV service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	260 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. (For example, uninitialized handle or a reference to a deleted entry.) Only pass an objectHandle parameter that was returned by a prior z/OS JSON parser service call.</p>

Table 96. Return codes for the HWTJGBOV service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
105 HWTJ_HANDLE_TYPE_ERROR	261 HWTJ_HANDLE_TYPE_ERROR	Meaning: Program error. The specified entryValueHandle does not represent an entry value with a JSON data type of HWTJ_BOOLEAN_TYPE . Action: Check for a probable coding error. Correct the mismatched handle and specify an entryValueHandle value that represents a boolean entry value handle.
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	Meaning: Program error. The calling program is disabled. The system rejects the service request. Action: Check the calling program for a probable coding error.
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	Meaning: The operating system level does not support this service. The system rejects the service request. Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTJGENC – Get JSON encoding

Call the HWTJGENC service to determine which encoding type is in effect for a JSON parser instance.

Description

This service returns one of the values { **HWTJ_ENCODING_UNKNOWN**, **HWTJ_ENCODING_EBCDIC**, **HWTJ_ENCODING_UTF8** } which reflects what the user may have asserted to the JSON parser (using **HWTJSENC**), or what the JSON parser instance has discovered from its processing of input data.

Environment

The requirements for the caller are:

Requirement

Minimum authorization:

Dispatchable unit mode:

Cross memory mode:

AMODE:

Details

Supervisor state or problem state, any PSW key.

Task or SRB.

Any PASN, any HASN, any SASN.

31 bit.

Requirement	Details
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 470](#) for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGENC service.

All information for the HWTJGENC service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJGENC(ReturnCode, ParserHandle, Encoding, DiagArea);</pre>	<pre>address hwtjson "hwtjgenc", "ReturnCode", "ParserHandle", "Encoding", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

Encoding

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the value of any encoding type which may be in effect for the JSON parser instance represented by the input parser handle. Constants for the possible values are defined in the supplied interface definition files (IDFs).

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

Abend codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0010yyyy' for one of the following reasons:

yyyy**Reason****0000**

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [“HWTJESCT — Encode or decode escape sequences \(REXX\)”](#) on page 497.

Table 97. Return codes for the HWTJGENC service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 97. Return codes for the HWTJGENC service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 97. Return codes for the HWTJGENC service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	Meaning: The operating system level does not support this service. The system rejects the service request. Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTJGJST – Get JSON type

Call the HWTJGJST service to obtain the JSON data type of an entry or object.

Description

The HWTJGJST service returns the type of JSON data which is associated with the specified entry object or entry handle (**objOrEntryValueHandle**). This value can then be used by subsequent parse methods to take specific action based on the type of the data.

Example: If the **objOrEntryValueHandle** represents an object (JSONType = HWTJ_OBJECT_TYPE), the HWTJGNUE (Get number of entries) parse method can be used to determine the number of entries in the object. If the **objOrEntryValueHandle** represents a string (JSONType = HWTJ_STRING_TYPE), the HWTJGVAL (Get value) parse method can be used to get the actual string text.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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 470 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGJST service

All information for the HWTJGJST service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJGJST(ReturnCode, ParserHandle, ObjOrEntryValueHandle, JSONType, DiagArea);</pre>	<pre>address hwtjson "hwtjgkst", "ReturnCode", "ParserHandle", "ObjOrEntryValueHandle", "JSONType", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

ObjOrEntryValueHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a handle representing either a JSON object or a JSON entry value. A JSON entry is a particular JSON name or value pair.

JSONType

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

A value representing the type of data of the item represented by the **objOrEntryValueHandle** parameter. Constants for the type values are defined in the supplied interface definition files (IDFs). For REXX, consult one of the other high-level language IDFs for a list of possible values.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0004yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 98 on page 512](#).

Table 98. Return codes for the HWTJGJST service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 98. Return codes for the HWTJGJST service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>

Table 98. Return codes for the HWTJGJST service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
104 HWTJ_HANDLE_INV	260 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objOrEntryValueHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. (For example, uninitialized handle or a reference to a deleted entry.) Pass either of the following values in the objOrEntryValueHandle parameter:</p> <ul style="list-style-type: none"> • An object value handle or entry value handle that was returned by a prior z/OS JSON parser service call • A zero for the root object
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTCONST — Initialize predefined variables (REXX)

Call the HWTCONST service to initialize predefined variables in the current REXX variable pool.

Description

This service sets the variables with names prefixed for HWTJ corresponding to the interface definition for the JSON toolkit. This service is helpful when using symbolic names in checking for specific return codes or when specifying constant values in the application. The variable **HWT_CONSTANTS** is set to a list of the interface variable names, which is useful on a procedure expose statement to make the variables visible to a procedure.

Note: This service also sets the variables for the z/OS HTTP Enabler (HWTH-prefixed) as well. If the REXX application utilizes both the HTTP and JSON parser portions of the toolkit, it is only necessary to call HWTCONST once to initialize all the variables.

Syntax

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

REXX parameters
<pre>address hwtjson "hwtconst", "ReturnCode", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Character string

Contains the return code from the service.

DiagArea.

Returned parameter.

- **Type:** Stem variable

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [“HWTCONST — Initialize predefined variables \(REXX\)”](#) on page 477.

Table 99. Return codes for the HWTCONST service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.

HWTJGNUE — Get number of entries

Call the HWTJGNUE service to obtain the number of entries (name or value pairs) associated with an object.

Description

The HWTJGNUE service returns the number of entries (name or value pairs) associated with the object represented by the specified object handle (**objectHandle**). The returned value is typically used on subsequent calls as a loop control to traverse all of the object elements.

Environment

The requirements for the caller are:

Requirement

Minimum authorization:

Dispatchable unit mode:

Cross memory mode:

Details

Supervisor state or problem state, any PSW key.

Task or SRB.

Any PASN, any HASN, any SASN.

Requirement	Details
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 470 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGNUE service

All information for the HWTJGNUE service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJGNUE(ReturnCode, ParserHandle, ObjectHandle, NumOfEntries, DiagArea);</pre>	<pre>address hwtjson "hwtjgnue", "ReturnCode", "ParserHandle", "ObjectHandle", "NumOfEntries", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

- ReturnCode**
Returned parameter.
- **Type:** Integer (non-REXX), character representation of an integer (REXX)
 - **Length:** 4 bytes (non-REXX)
- Contains the return code from the service.
- ParserHandle**
Supplied parameter.
- **Type:** Character string
 - **Length:** 12 bytes (non-REXX)
- Specifies the JSON parser instance to be used.
- ObjectHandle**
Supplied parameter.
- **Type:** Integer (non-REXX), character representation of an integer (REXX)
 - **Length:** 4 bytes (non-REXX)

Specifies a handle representing a particular JSON object (object or array object). The **objectHandle** value can be 0 (zero) for the root object or an entry value handle (**entryValueHandle**) whose JSON type is HWTJ_OBJECT_TYPE or HWTJ_ARRAY_TYPE.

NumOfEntries

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

A value represents the number of entries found in the object or array represented by the **objectHandle** parameter.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0005yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 100 on page 517](#).

Table 100. Return codes for the HWTJGNUE service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 100. Return codes for the HWTJGNUE service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>

Table 100. Return codes for the HWTJGNUM service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
104 HWTJ_HANDLE_INV	260 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. (For example, uninitialized handle or a reference to a deleted entry.) Pass either of the following values in the objectHandle parameter:</p> <ul style="list-style-type: none"> • An object handle or entry value handle that was returned by a prior z/OS JSON parser service call • A zero for the root object
105 HWTJ_HANDLE_TYPE_ERROR	261 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified objectHandle does not represent an entry value with a JSON data type of either HWTJ_OBJECT_TYPE or HWTJ_ARRAY_TYPE.</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an objectHandle value that represents either an object handle or an array handle.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJGNUM – Get number value (non-REXX)

Call the HWTJGNUM service to obtain the binary representation of a JSON number.

Description

The HWTJGNUM service returns the binary representation of a JSON number. The service returns a number in integer or floating-point format, based on the actual character-based number value represented by the specified entry value handle (**entryValueHandle**). This service may only be called

when the JSON type for the **entryValueHandle** is HWTJ_NUMBER_TYPE. Consider using HWTJGVAL if no data conversion of a number is required.

If the value contains a decimal point or the scientific notation indicator (e or E), the service converts the value to a floating-point number. Conversely, if the value contains neither a decimal point nor the scientific notation indicator, the service converts the value to an integer number.

Note: Users of the z/OS web enablement toolkit that employ COBOL as their primary language, are currently unable to use the HWTJGNUV callable service to retrieve a compatible binary value for JSON number values that contain a decimal point, or that specify exponential (E-notation) forms. The HWTJGNUV service is designed to return a 4- or 8-byte binary form of a real (floating point) JSON number in the IEEE754 floating point format, and this standard format is not compatible with the Hexadecimal Floating Point used by COBOL on z.

If your JSON body is encoded in IBM-1047, you may be able to use the COBOL function NUMVAL against the string value of the JSON number. This value may be obtained using HWTJGVAL for the numeric-type field or array element. If your JSON body is encoded in UTF-8, the value returned from HWTJGVAL may require code-page conversion to IBM-1047 (or other EBCDIC code page) for COBOL's NUMVAL function to be useful.

Underflow conditions: Converting a floating-point number may result in a number that is too small to be arithmetically represented in the buffer length as specified by the **valueBufferLen** (precision) parameter. This is known as an underflow condition, and this service will return a value of zero in this case. If this is not an acceptable outcome for the application and the **valueBufferLen** value can be increased, the application should increase the **valueBufferLen** value and reissue the request.

As with any floating-point number handling, use caution when performing floating-point conversions on certain types of data, such as currency values, as unexpected results may occur.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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 470](#) for details about how to call the z/OS JSON parser services in the various supported programming languages.

Syntax

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

Parameters
<pre>CALL HWTJGNUV(ReturnCode, ParserHandle, EntryValueHandle, ValueBufferAddr, ValueBufferLen, ValueDescriptor, DiagArea);</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- **Type:** Integer
- **Length:** 4 bytes

Contains the return code from the service.

ParserHandle

Supplied parameter

- **Type:** Character string
- **Length:** 12 bytes

Specifies the JSON parser instance to be used.

EntryValueHandle

Supplied parameter

- **Type:** Integer
- **Length:** 4 bytes

Specifies a handle representing an entry in a JSON object that has a JSON type of NUMBER.

ValueBufferAddr

Supplied parameter

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of an 8-byte buffer where this service will store the converted number. After the service completes, the buffer contains the numeric binary representation of the number represented by the specified **entryValueHandle** parameter. The **valueDescriptor** parameter will indicate the format of the binary data pointed to by this parameter. The value could be a positive or negative 8-byte integer or an 8-byte floating-point number in IEEE binary floating-point format.

ValueBufferLen

Supplied parameter

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length (precision) of the buffer as specified by the **valueBufferAddr** parameter. The length specified here will determine the precision of the number returned by the service. The valid values are:

4

The service will return either a 4-byte signed integer or a 4-byte IEEE floating-point number.

8

The service will return either an 8-byte signed integer or an 8-byte IEEE floating-point number.

ValueDescriptor

Returned parameter

- **Type:** Integer
- **Length:** 4 bytes

A constant that indicates the format of the number being returned. Possible values include integer and floating point.

DiagArea

Returned parameter

- **Type:** Character string
- **Length:** 132 bytes

A storage area provided by the caller that may contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'000Dyyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 101 on page 522](#).

Table 101. Return codes for the HWTJGNUM service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 101. Return codes for the HWTJGNUV service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one will be allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service call abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	260 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the entryValueHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. (For example, uninitialized handle or a reference to a deleted entry.) Only pass an entryValueHandle parameter that was returned by a prior z/OS JSON parser service call.</p>

Table 101. Return codes for the HWTJGNUM service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
105 HWTJ_HANDLE_TYPE_ERROR	261 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified entryValueHandle does not represent an entry value with a JSON data type of HWTJ_NUMBER_TYPE.</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an entryValueHandle value that represents a number entry value handle.</p>
801 HWTJ_JGNUM_VALBUFFADDR_INV	2049 HWTJ_JGNUM_VALBUFFADDR_INV	<p>Meaning: Program error. The caller specified a value of zero as the address of the valueBufferAddr parameter.</p> <p>Action: Check for a probable coding error. Specify the actual address of the buffer where the returned converted binary representation of the JSON number is to be written.</p> <p>Note: Specifying a bad buffer address other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
802 HWTJ_JGNUM_VALBUFFLEN_INV	2050 HWTJ_JGNUM_VALBUFFLEN_INV	<p>Meaning: Program error. The caller specified an invalid value for the length of the output buffer.</p> <p>Action: Check for a probable coding error. Validate that the specified valueBufferLen value is one of the allowed values according to the description of the valueBufferLen parameter.</p> <p>Note: Specifying a bad buffer length other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
803 HWTJ_JGNUM_NUM_OUT_OF_RANGE	2051 HWTJ_JGNUM_NUM_OUT_OF_RANGE	<p>Meaning: The number cannot be converted. The attempt to convert the number to a binary representation results in an overflow condition (positive or negative).</p> <p>Action: If a valueBufferLen of 4 was specified, consider increasing the valueBufferLen to 8 and retrying the service. If 8 was already specified, the number is too large for the HWTJGNUM service to convert. Consider issuing HWTJGVAL to obtain the character representation of the number.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 101. Return codes for the HWTJGNUV service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	Meaning: The operating system level does not support this service. The system rejects the service request. Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTJGOEN – Get object entry

Call the HWTJGOEN service to obtain the address of the name portion and the entry value handle for an object entry.

Description

The HWTJGOEN service returns two items which are associated with the specified object handle (**objectHandle**) and object entry index (**objectEntryIndex**):

1. The address of the name portion of this entry (within the previously supplied JSON text)
2. The **entryValueHandle** of this entry, which can be used on subsequent service calls to inquire about the value of this particular object entry

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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 470 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGOEN service

All information for the HWTJGOEN service applies for REXX requests except:

- EntryName replaces EntryNameBufferAddr and EntryNameBufferLen
- ActualNameLenReturned is not used

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJGOEN(ReturnCode, ParserHandle, ObjectHandle, ObjectEntryIndex, EntryNameBufferAddr, EntryNameBufferLen, EntryValueHandle, ActualNameLenReturned, DiagArea);</pre>	<pre>address hwtjson "hwtjgoen", "ReturnCode", "ParserHandle", "ObjectHandle", "ObjectEntryIndex", "EntryName", "EntryValueHandle", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

ObjectHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a handle which represents a particular JSON object. The **objectHandle** value can be 0 (zero) for the root object or an entry value handle (**entryValueHandle**) whose JSON type is HWTJ_OBJECT_TYPE.

ObjectEntryIndex

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies the index of the n^{th} entry of the object specified by **objectHandle**.

Note: This is a zero-origin index, meaning that the first entry in the object has an index of zero; the n^{th} entry has an index of $(n - 1)$.

EntryName (REXX)

Returned parameter.

- **Type:** Character string

Contains the returned EntryName of the JSON object entry.

EntryNameBufferAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of a buffer provided to hold the returned entry name of the JSON object entry.

Note: The returned entry name data will be in the encoding currently in effect for the JSON parser instance.

EntryNameBufferLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the buffer provided as specified by the **entryNameBufferAddr** parameter.

EntryValueHandle

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

A handle representing the particular entry value selected in the JSON object.

ActualNameLenReturned (non-REXX)

Returned parameter.

- **Type:** Integer
- **Length:** 4 bytes

The actual length of the returned entry name (in bytes). If the size of the provided buffer is not large enough to contain the entry name (`returnCode = HWTJ_BUFFER_TOO_SMALL`), this returned value can be used to reissue the service with the proper buffer size.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0006yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 102 on page 528](#).

Table 102. Return codes for the HWTJGOEN service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	Meaning: Program error. Two possible reasons can result in this return code: <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. Action: Check for a probable coding error. <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.

Table 102. Return codes for the HWTJGOEN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	260 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. (For example, uninitialized handle or a reference to a deleted entry.) Pass either of the following values in the objectHandle parameter:</p> <ul style="list-style-type: none"> An object handle or entry value handle that was returned by a prior z/OS JSON parser service call A zero for the root object
105 HWTJ_HANDLE_TYPE_ERROR	261 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified objectHandle does not represent an object with a JSON data type of HWTJ_OBJECT_TYPE.</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an objectHandle value that represents an object (non-array object) handle.</p>
106 HWTJ_BUFFER_TOO_SMALL	262 HWTJ_BUFFER_TOO_SMALL	<p>Meaning: Program error. The buffer provided for the entry name is not large enough to contain the entire name. The value of the entryNameBufferLen parameter is too small.</p> <p>Action: Check for a probable coding error. Examine the actualNameLenReturned parameter to determine the required size for the entry name. Then, increase the size of the buffer specified by the entryNameBufferAddr and entryNameBufferLen parameters and reissue the service call.</p>

Table 102. Return codes for the HWTJGOEN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
107 HWTJ_INDEX_OUT_OF_BOUNDS	263 HWTJ_INDEX_OUT_OF_BOUNDS	<p>Meaning: Program error. The index value specified by the objectEntryIndex parameter is greater than the number of entries in the object.</p> <p>Action: Check for a probable coding error.</p> <ul style="list-style-type: none"> • Issue the HWTJGNUE (Get number of entries) service to determine the upper bound for the number of object entries. • Remember that the number of entries returned is the actual count of entries, but accessing a particular entry uses zero-origin indexing. To iterate through all the elements, the application must start at the first entry (index 0) up to the number of entries minus 1.
301 HWTJ_JGOEN_BUFFERADDR_INV	769 HWTJ_JGOEN_BUFFERADDR_INV	<p>Meaning: Program error. The caller specified a value of zero for EntryNameBufferAddr parameter.</p> <p>Action: Check for a probable coding error. Specify the actual address of the buffer to which the JSON object entry name is to be written.</p>
302 HWTJ_JGOEN_BUFFERLEN_INV	770 HWTJ_JGOEN_BUFFERLEN_INV	<p>Meaning: Program error. The value specified for the EntryNameBufferLen parameter is zero or less.</p> <p>Action: Check for a probable coding error. Correct EntryNameBufferLen to reflect the length of the buffer provided for the JSON object entry name (EntryNameBufferAddr).</p>
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJGVAL — Get value

Call the HWTJGVAL service to obtain the location of a string or number within the JSON text.

Description

The HWTJGVAL service returns the exact location in the JSON source text of a string or number associated with the specified entry value handle (**entryValueHandle**). This service might be called whenever the JSON type for the **entryValueHandle** is either a string or numeric type.

Note: If the JSON type for the **entryValueHandle** is numeric and the binary representation (rather than the character representation) of the number is preferred, consider using the HWTJGNUV (get numeric value) service.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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 470](#) for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGVAL service

All information for the HWTJGVAL service applies for REXX requests except:

- Value replaces ValueLocationAddr and ValueLen

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJGVAL (ReturnCode, ParserHandle, EntryValueHandle, ValueLocationAddr, ValueLen, DiagArea);</pre>	<pre>address hwtjson "hwtjgval", "ReturnCode", "ParserHandle", "EntryValueHandle", "Value", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

EntryValueHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a handle representing an entry in a JSON object that has a JSON type of STRING or NUMBER.

Value (REXX)

Returned parameter.

- **Type:** Character string

The name of a REXX variable that the service sets to the entry value represented by the **EntryValueHandle**.

Note: The encoding type of the JSON text value returned by HWTJGVAL will be that which is currently in effect for the JSON parser instance.

ValueLocationAddr (non-REXX)

Returned parameter.

- **Type:** Pointer
- **Length:** 4 bytes

The address of the location in the source JSON text where the value represented by **entryValueHandle** resides.

ValueLen (non-REXX)

Returned parameter.

- **Type:** Integer
- **Length:** 4 bytes

The length (in bytes) of the wanted value referenced by the **valueLocationAddr** parameter.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0007yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 103 on page 533](#).

Table 103. Return codes for the HWTJGVAL service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 103. Return codes for the HWTJGVAL service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	260 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the entryValueHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. (For example, uninitialized handle or a reference to a deleted entry.) Only pass an entryValueHandle parameter that was returned by a prior z/OS JSON parser service call.</p>

Table 103. Return codes for the HWTJGVAL service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
105 HWTJ_HANDLE_TYPE_ERROR	261 HWTJ_HANDLE_TYPE_ERROR	Meaning: Program error. The specified entryValueHandle does not represent an entry value with a JSON data type of HWTJ_STRING_TYPE or HWTJ_NUMBER_TYPE. Action: Check for a probable coding error. Correct the mismatched handle and specify an entryValueHandle value that represents either a string or number entry value handle.
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	Meaning: Program error. The calling program is disabled. The system rejects the service request. Action: Check the calling program for a probable coding error.
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	Meaning: The operating system level does not support this service. The system rejects the service request. Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTJINIT — Initialize a parser instance

Call the HWTJINIT service to prepare to parse a JSON text stream.

Description

This service must be invoked before any other z/OS JSON parsing service contained in the toolkit. The service prepares the memory space required by the z/OS JSON parser in the callers address space. The memory allocation created by HWTJINIT is known as a *parser instance*. If you need your program to parse more than one JSON data stream simultaneously, you can create more than one parser instance by invoking HWTJINIT as many times as necessary (once for each data stream). This allows the parser to track multiple JSON streams at the same time. The caller distinguishes one JSON stream from another by a unique JSON parser handle returned from each HWTJINIT call.

Environment

The requirements for the caller are:

Requirement

Minimum authorization:

Details

Supervisor state or problem state, any PSW key.

Requirement	Details
Dispatchable unit mode:	Task or SRB.
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 470 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJINIT service

All information for the HWTJINIT service applies for REXX requests except:

- MaxParserWorkAreaSize is not used

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJINIT(ReturnCode, MaxParserWorkAreaSize, ParserHandle, DiagArea);</pre>	<pre>address hwtjson "hwtjinit", "ReturnCode", "ParserHandle", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

MaxParserWorkAreaSize (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the maximum amount of storage, in bytes, that the parser can consume during parser functions. This allows the application to optionally set an upper threshold for the amount of storage the parser can consume in addition to the storage explicitly allocated by the callers application.

- A value of zero (recommended) specifies no limit to the work area size; the parser can use as much storage as necessary to successfully parse or create JSON text.
- A nonzero value does not necessarily cause the parser to obtain the specified amount of storage when the JSON instance is initialized, but allows the parser to consume up to the specified amount of memory in the users memory area. If the specified value is less than the minimum amount required for the parser to operate, a return code of HWTJ_WARNING is returned and the **diagArea** parameter contains amount of storage obtained, which also serves as the maximum area the parser is able to use. The **diagArea** parameter also contains the amount of overhead storage required by the parser, which can be useful when calculating an appropriate value to specify. Consider the following formula as a rough estimate to determine the value to specify:

```
maxParserWorkAreaSize = (number of name/value pairs in the JSON text × 30) +
                        (total length, in bytes, of all "names", string values and
                         number values added to the JSON text via the HWTJCREN service) +
                        overhead storage
```

ParserHandle

Returned parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

A value generated by the parser that represents a handle to be used on all subsequent JSON parser services for this parser instance. This instance contains all of the data structures and storage areas required for the JSON parsing services to run efficiently. The REXX variable is updated by this service.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that may contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0008yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 104 on page 537](#).

Table 104. Return codes for the HWTJINIT service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.

Table 104. Return codes for the HWTJINIT service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
4 HWTJ_WARNING	4 HWTJ_WARNING	<p>Meaning: Warning. The maxParserWorkAreaSize value specified by the caller is smaller than the minimum required by the z/OS JSON parser.</p> <p>Action: The system creates the minimum size workarea (larger than the user-specified value) and informs the caller of the actual size in the reasonDesc field of the diagArea. Modify future invocations of the HWTJINIT service to specify a larger maxParserWorkAreaSize value based on the recommendation provided in the reasonDesc field.</p>
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>
10B HWTJ_CANNOT_OBTAIN_WORKAREA	267 HWTJ_CANNOT_OBTAIN_WORKAREA	<p>Meaning: System error. The Storage Obtain service could not obtain the work area storage as required by the z/OS JSON parser during the HWTJINIT service call.</p> <p>Action: Consult the diagArea for the Storage Obtain failure return code and additional information found in the ReasonDesc section. Check to see if there is sufficient memory available in order for the parser to obtain the necessary amount of work area. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 104. Return codes for the HWTJINIT service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	Meaning: The operating system level does not support this service. The system rejects the service request. Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTJOPTS – Set parser options

Call the HWTJOPTS service to set options related to the JSON text that will be encountered by the z/OS JSON parser.

Description

This service allows the application to set various behavior preferences for the z/OS JSON parser. The preference currently supported is associated with toleration of single and multi-line comments defined by the JSON5 Data Interchange Format extension to JSON (<https://spec.json5.org/#comments>). By default, the z/OS JSON parser will tolerate the comments, however the application can choose to specify **HWTJ_TOLERATE_CMT_OFF**, to indicate the parser should comply with the JSON specification and classify any comments encountered as not a valid syntax. Alternatively, the application can choose to specify **HWTJ_TOLERATE_CMT_ON** to indicate to the z/OS JSON parser that it should prime its parsing algorithm to expect comments in the JSON text. See the description of the various option types for further information regarding possible performance implications.

This service, if invoked, needs to be invoked prior to the first successful "parse". The first parse is accomplished by either using HWTJPARS or, if authoring new JSON text, with the invocation of HWTJCREN.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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 470 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJOPTS service

All information for the HWTJOPTS service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJOPTS(ReturnCode, ParserHandle, OptionType, DiagArea);</pre>	<pre>address hwtjson "hwtjopts", "ReturnCode", "ParserHandle", "OptionType", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Returned parameter.

- **Type:** Character string
- **Length:**12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

OptionType

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a constant value that indicates the option which is to take effect for the JSON parser instance represented by the input parser handle.

If not set, by default the z/OS JSON parser will tolerate single and multi-line comments defined by the JSON5 Data Interchange Format extension to JSON.

Regardless of the comment related option chosen, the application will not be able to retrieve or access any comments that maybe encountered in the JSON text. Any JSON output produced using HWTJSERI will be void of any comments that may have been present in the original content.

Examples of supported JSON5 single and multi-line comments:

```
//single line comment starts with a double-solidus and ends with a line terminator  
//the supported line terminator character is a Line Feed<LF>:  
//IBM-1047 -> hex 15  
//UTF-8 -> hex 0A
```

Figure 33. Example of JSON single line comment


```
/* multi-line comment on a single line, uses a solidus + asterisk form */
```

Figure 34. Example of JSON multi-line comment on one line

```
/* multi-line comments start with solidus + asterisk,
   and can span arbitrarily-many lines,
   until ended with an asterisk + solidus like this */
```

Figure 35. Example of JSON multi-line comment

```
/******
 * This is another multi-line comment *
 * that creates a flower box shape. *
 *****/
```

Figure 36. Example of JSON multi-line comment

In the following example, comments are indicated in **bold**:

```
/***** Testing All *****/
* All attributes are tested inside their *
* condition and displayed with send/log *
* message. All modifiable attributes are then *
* modified within their own condition def *
* then standard actions *
*****/
{
  /****** Testing CompletionCode *****/
  "condition" : " String(CompletionCode) = '()' "
  , "actions" :
    [
      {
        "action" : "sendMessage" //first action
        , "message": /*CC? */ " 'CompletionCode: '||String(CompletionCode)"
      }
    ]
}
```

Figure 37. Example of comments on JSON data

This default setting is recommended when the input is either always uncommented JSON or a combination of commented and uncommented JSON. This setting and HWTJ_TOLERATE_CMT_OFF will deliver optimum performance for uncommented JSON text. In general, parsing performance for commented JSON text will always be slower compared to uncommented JSON, however it can be optimized by including a comment as early in your data as possible. For example:

```
line 1 /* This JSON contains a policy definition */ <- first comment
line 2 {
line 3     "action" : "sendMessage" //first action
line 4     , "message": " 'CompletionCode: '||String(CompletionCode)"
line 5 }
```

Figure 38. Commented JSON Example 1

```
line 1 {
line 2     "action" : "sendMessage" //first action <- first comment
line 3     , "message": " 'CompletionCode: '||String(CompletionCode)"
line 4 }
```

Figure 39. Commented JSON Example 2

```
line 1 {
line 2     "action" : "sendMessage"
line 3     , "message": " 'CompletionCode: '||String(CompletionCode)"
line 4 } /* The above was an action */ <- first comment
```

Figure 40. Commented JSON Example 3

Of the three JSON examples shown, [Figure 38 on page 541](#) will have the best performance because the parser will detect it should tolerate comments when it reads the very first line, on par with setting HWTJ_TOLERATE_CMT_ON. The [Figure 40 on page 541](#) will have the worst performance because the parser will not be aware it needs to tolerate comments until the very end of the text and will need to re-parse the whole text body, resulting in two full parses.

Valid values for this parameter are:

HWTJ_TOLERATE_CMT_OFF

The z/OS JSON parser should comply with the JSON specification and classify any JSON5 single and multi-line style comments encountered as invalid syntax.

HWTJ_TOLERATE_CMT_ON

The z/OS JSON parser should be primed to expect to encounter single and multi-line comments during parse, HWTJPARS API, and creation, HWTJCREN API, of JSON text. Parsing performance for commented JSON text is slower compared to uncommented JSON. This option is only recommended for content that will contain comments. Uncommented JSON text will also experience a performance penalty when the parser is primed to parse commented JSON text.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0008yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in .

Table 105. Return codes for the HWTJOPTS service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 105. Return codes for the HWTJOPTS service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>
B01 HWTJ_JOPTS_OPTION_INV	2817 HWTJ_JOPTS_OPTION_INV	<p>Meaning: Program error. The application did not pass a valid optionType parameter.</p> <p>Action: Check for a probable coding error. Change the optionType passed in to one of the valid value.</p>

Table 105. Return codes for the HWTJOPTS service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
B02 HWTJ_JOPTS_CMNT_NOT_ALLOWED	2818 HWTJ_JOPTS_CMNT_NOT_ALLOWED	Meaning: One or more of the options in effect may not be changed. Action: Check for a probable usage error. If a comment toleration option is already in effect for the parser instance represented by the parse handle, HWTJOPTS may not be invoked to alter it.
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	Meaning: Program error. The calling program is disabled. The system rejects the service request. Action: Check the calling program for a probable coding error.
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	Meaning: The operating system level does not support this service. The system rejects the service request. Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTJPARS – Parse a JSON string

Call this service to assign JSON text to a parser instance, validate its syntax, and create an internal representation of it in the parser's memory.

Description

The HWTJPARS service performs the following functions:

1. It assigns the JSON text specified by the caller to a previously created parser instance. All prior JSON text associated with this JSON parser instance, if any, is disassociated with the instance.
2. It checks the JSON text input to ensure that the text conforms to the JSON syntax standards. See [Introducing JSON \(json.org\)](#) for a description of what comprises valid JSON text and JSON5 standard comment sections for details of supported single and multi-line comment syntax. The parsers comment toleration behavior depends on the value specified in HWTJOPTS. Any errors or deviations from the published syntax standards are flagged as a parse error, and the caller is notified of both the kind of syntax error and the location in the JSON text input buffer where the reported error exists. The JSON text input may either be in EBCDIC encoding (codepage 1047) or in UTF-8. Users may, optionally, assert one of these encodings prior to invoking HWTJPARS, via the HWTJSENC service. In the absence of any such assertion, the HWTJPARS service will attempt to discover the encoding type. Either thru assertion or discovery, the encoding type in effect may be surfaced at any point subsequent to HWTJPARS via the HWTJGENC service.

3. It creates an internal representation of the JSON text in the parser instance memory, allowing subsequent parse functions to be executed quickly. (The parser creates this internal representation of the JSON text in a DOM-like tree structure.)

Note: Reissuing the HWTJPARS service using the same parser handle as a prior HWTJPARS invocation will cause the parser instance to be reinitialized. All data saved in the internal representation of the JSON text, which is used by the parser for easy traversal by the other methods, must be regenerated. In addition, the new HWTJPARS invocation causes all previous handles returned by the parser for objects or entries to be invalidated. The encoding type and any setting specified via HWTJOPTS that was in effect for the prior JSON text will be applied when HWTJPARS is reissued with an existing parse handle. If the subsequent JSON text to be parsed has a different encoding type or requires a different comment toleration setting than what was specified using HWTJOPTS, use the HWTJINIT service to create a new JSON parse instance and invoke HWTJPARS and other services with the new handle.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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 470 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJPARS service

All information for the HWTJPARS service applies for REXX requests except:

- JSONText replaces JSONTextAddr and JSONTextLen

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJPARS(ReturnCode, ParserHandle, JSONTextAddr, JSONTextLen, DiagArea);</pre>	<pre>address hwtjson "hwtjpars", "ReturnCode", "ParserHandle", "JSONText", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance associated with the supplied JSON text indicated by the **JSONTextAddr** parameter.

JSONText (REXX)

Supplied parameter.

- **Type:** Character string

Specifies the REXX variable that contains the JSON string to be parsed.

JSONTextAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of the actual buffer storage location of the JSON text to be parsed and associated with a previously initialized parser instance.

Notes:

1. The parser maintains a binding to the buffer specified by this parameter even after the service call to the HWTJPARS service completes. This buffer must not be freed or reused until the caller has completed all parsing functions associated with this JSON text string.
2. The JSON text supplied in the buffer may either be in EBCDIC (codepage 1047) or in UTF-8.

JSONTextLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the JSON text buffer pointed to by the **JSONTextAddr** parameter.

Note: The parser does not validate whether the specified length matches the length of the actual JSON text. The parser continues to parse the input up to the specified length. If the JSON text is shorter than the specified length, the parser generates a parse error if any non-null characters are encountered after the end of the JSON text. If the JSON text is longer than the specified length, the parser generates the appropriate parser error.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0009yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 106 on page 547](#).

Table 106. Return codes for the HWTJPARS service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 106. Return codes for the HWTJPARS service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter, which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>

Table 106. Return codes for the HWTJPARS service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
108 HWTJ_WORKAREA_TOO_SMALL	260 HWTJ_WORKAREA_TOO_SMALL	<p>Meaning: Program error. The amount of JSON text to parse requires a larger work area than the size specified by the maxParserWorkAreaSize parameter on the HWTJINIT service.</p> <p>Action: Check for a probable coding error. IBM recommends specifying a maxParserWorkAreaSize value of 0 (unlimited work area size available to the parser). If the application cannot specify this value, then modify the invocation of the HWTJINIT service to specify a larger maxParserWorkAreaSize value based on the recommendations given in the ReasonDesc section of the diagArea and in the description of the HWTJINIT maxParserWorkAreaSize parameter.</p>
109 HWTJ_PARSE_ERROR	261 HWTJ_PARSE_ERROR	<p>Meaning: Supplied JSON text error. The JSON text passed by the caller to the HWTJPARS service contains a syntax or encoding error.</p> <p>Note: This error might also be generated if the specified JSONTextLen value is greater than the actual length of the JSON text and there are non-null (white space) characters after the JSON text.</p> <p>Action: Check the diagArea for a complete explanation of the error. The reasonCode portion of the diagArea pinpoints the reason for the parse failure, while the ReasonDesc portion points to the exact location in the supplied JSON text in cases when the parser detects a JSON syntax error.</p>
10B HWTJ_CANNOT_OBTAIN_WORKAREA	267 HWTJ_CANNOT_OBTAIN_WORKAREA	<p>Meaning: System error. The Storage Obtain service could not obtain the work area storage as required by the z/OS JSON parser during the HWTJPARS service call.</p> <p>Action: Consult the diagArea for the Storage Obtain failure return code and additional information found in the ReasonDesc section. Check to see if there is sufficient memory available in order for the parser to obtain the necessary amount of work area. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Table 106. Return codes for the HWTJPARS service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
401 HWTJ_JPARS_JSONTEXTADDR_INV	1025 HWTJ_JPARS_JSONTEXTADDR_INV	<p>Meaning: Program error. The caller specified a value of zero for the address of the JSON text buffer.</p> <p>Action: Check for a probable coding error. Specify the actual address of the JSON text buffer.</p> <p>Note: Specifying a bad JSON buffer address other than zero might result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
402 HWTJ_JPARS_JSONTEXTLEN_INV	1026 HWTJ_JPARS_JSONTEXTLEN_INV	<p>Meaning: Program error. The caller specified a value of zero for the length of the JSON text buffer.</p> <p>Action: Check for a probable coding error. Specify the actual length of the JSON text buffer.</p> <p>Note: Specifying a bad JSON buffer length other than zero might result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
403 HWTJ_JPARS_WORKAREA_ERROR	1027 HWTJ_JPARS_WORKAREA_ERROR	<p>Meaning: System error. The z/OS JSON parser will sometimes need to issue system services to increase or decrease the amount of work area storage in order to properly build an internal representation of the JSON text data. When an error occurs while obtaining or releasing this storage, the parser cannot proceed.</p> <p>Action: Consult the diagArea for the return code of the failing Storage Obtain or Release service and additional information in the ReasonDesc section. Check to see if there is sufficient memory available in order for the application to obtain the necessary amount of work area storage. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 106. Return codes for the HWTJPARS service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	Meaning: The operating system level does not support this service. The system rejects the service request. Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTJSENC – Set JSON encoding

Call the HWTJSENC service to assert an encoding type for JSON data which will be used by a JSON parser instance.

Description

This service asserts the encoding type of the JSON text data which is to be input to an HWTJPARS or HWTJCREN invocation. It is optional in parsing scenarios, as the HWTJPARS service will attempt to discover the encoding type of the input text data. Parse will fail if the asserted encoding type does not match the discovered one, so asserting a value can be thought of as a safeguard (instructing HWTJPARS to corroborate the JSON text data's expected encoding). In scenarios of building JSON text anew using HWTJCREN, HWTJSENC should be invoked prior to any HWTJCREN usage, to ensure the intended outcome.

The commingling of data with differing encodings is not supported, and the JSON parser instance attempts to prevent it by disallowing HWTJSENC invocations once an encoding type is in effect. An encoding type takes effect for a given JSON parser instance following a first successful HWTJPARS or HWTJCREN invocation, and remains in effect for the JSON parser instance until such time as the HWTJTERM service is invoked to terminate the instance.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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 470 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJSENC service.

All information for the HWTJSENC service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJSENC(ReturnCode, ParserHandle, Encoding, DiagArea);</pre>	<pre>address hwtjson "hwtjsenc", "ReturnCode", "ParserHandle", "Encoding", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

Encoding

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the value of any encoding type which is to take effect for the JSON parser instance represented by the input parser handle. Constants for the possible values are defined in the supplied interface definition files (IDFs).

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

Abend codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'000Fyyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [“HWTJESCT — Encode or decode escape sequences \(REXX\)”](#) on page 497.

Table 107. Return codes for the HWTJSENC service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 107. Return codes for the HWTJSENC service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>
A01 HWTJ_JSENC_TYPE_INV	2561 HWTJ_JSENC_TYPE_INV	<p>Meaning: Program error. The specified encoding is not one of the supported types.</p> <p>Action: Check for a probable coding error. See the supplied interface definition files for possible values.</p>

Table 107. Return codes for the HWTJSENC service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
A02 HWTJ_JSENC_NOT_ALLOWED	2562 HWTJ_JSENC_NOT_ALLOWED	Meaning: The encoding type in effect may not be changed. Action: Check for a probable usage error. If an encoding is already in effect for the parser instance represented by the parse handle, HWTJSENC need not be invoked.
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	Meaning: Program error. The calling program is disabled. The system rejects the service request. Action: Check the calling program for a probable coding error.
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	Meaning: The operating system level does not support this service. The system rejects the service request. Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTJSERI – Serialize (build) JSON text

Call the HWTJSERI service to build the JSON text associated with a parser instance.

Description

The HWTJSERI service builds the JSON text associated with the specified parser instance by combining the existing JSON text (if any) and any newly added objects or entries. The resulting content will exclude any single and multi-line JSON5 style comments that may have been present in the original JSON text provided to the parser instance.

The serialize service can be invoked any time that the application needs to have the complete JSON text representation associated with the parser instance. For example, the application might need to generate the JSON text in the following cases:

- The application needs to send the modified JSON text back to the partner application in the client/server web application model.
- The application needs to store the JSON text into a local data store for later use.
- The application has already created JSON text (using the HWTJCREN service) and needs to perform a search (HWTJSRCH service) with a **searchType** of HWTJ_SEARCHTYPE_GLOBAL to search through all existing and new JSON text. (A search with a **searchType** of HWTJ_SEARCHTYPE_GLOBAL cannot search through entries created by the HWTJCREN service.) In this case, a serialize request would be issued and the JSON text output would be used as input to the parse service (HWTJPARS). The newly parsed JSON text could then be searched globally (albeit with new handles).

- The application received commented JSON text as input and requires a version of the JSON text with all comments removed.

The encoding type of the JSON text output produced by HWTJSERI will be that which is currently in effect for the JSON parser instance.

The serialization service will provide unformatted JSON. If the JSON text is in IBM-1047 encoding, the application can take advantage of the pretty print utility to format the text. The utility is shipped as HWTJSPRT in SYS1.SAMPLIB data set and as jsonprint in the z/OS Unix samples directory.

Example

Consider the following sample invocation in z/OS UNIX where parserOutput.json contains the serialized output returned by HWTJSERI:

```
/* REXX */
call syscalls 'ON'
inputFile = 'parserOutput.json'
outputFile = 'pretty.json'

call bpxwunix '/samples/jsonprint' inputFile '>' outputFile

return
```

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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 470 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJSERI service

All information for the HWTJSERI service applies for REXX requests except:

- NewJSONText replaces NewJSONTextBufferAddr and NewJSONTextBufferLen
- ActualJSONTextLen is not used

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJSERI(ReturnCode, ParserHandle, NewJSONTextBufferAddr, NewJSONTextBufferLen, ActualJSONTextLen, DiagArea);</pre>	<pre>address hwtjson "hwtjseri", "ReturnCode", "ParserHandle", "NewJSONText", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

NewJSONText (REXX)

Returned parameter.

- **Type:** Character string

REXX variable that contains the JSON text returned which is associated with the specified `parserHandle`.

NewJSONTextBufferAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of a buffer where the JSON text associated with the specified `parserHandle` is to be written.

NewJSONTextBufferLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the output buffer pointed to by the `newJSONTextBufferAddr` parameter.

ActualJSONTextLen (non-REXX)

Returned parameter.

- **Type:** Integer
- **Length:** 4 bytes

The actual length of the returned JSON text (in bytes). If the size of the buffer specified by the `newJSONTextBufferLen` parameter is not large enough to contain the entire JSON text, a return code of `HWTJ_BUFFER_TOO_SMALL` is returned. The application can then reissue the `HWTJSERI` request after the proper buffer size has been obtained.

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that may contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'000Ayyyy' for one of the following reasons:

yyyy**Reason****0000**

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 108 on page 558](#).

Table 108. Return codes for the HWTJSERI service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 108. Return codes for the HWTJSERI service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>

Table 108. Return codes for the HWTJSERI service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
106 HWTJ_BUFFER_TOO_SMALL	260 HWTJ_BUFFER_TOO_SMALL	<p>Meaning: Program error. The buffer provided for the new JSON text is not large enough to contain the generated text. The value specified for the newJSONTextBufferLen parameter is too small.</p> <p>Action: Check for a probable coding error. Examine the value returned in the actualJSONTextLen parameter to determine the necessary size for the JSON text buffer. Increase the size of the JSON text buffer, specify the larger size on the newJSONTextBufferLen parameter, and reissue the HWTJSERI request.</p>
10A HWTJ_ROOT_OBJECT_MISSING	266 HWTJ_ROOT_OBJECT_MISSING	<p>Meaning: Program error. There is no JSON text to serialize.</p> <p>Action: Check for a probable coding error. Invoke the parse service (HWTJPARS) or the create service (HWTJCEN) to associate JSON text with the specified parser instance before invoking the HWTJSERI service to regenerate the JSON text.</p>
501 HWTJ_JSERI_NEWJTXTBUFFADDR_INV	1281 HWTJ_JSERI_NEWJTXTBUFFADDR_INV	<p>Meaning: Program error. The caller specified a value of zero for the address of the newJSONTextBufferAddr parameter.</p> <p>Action: Check for a probable coding error. Specify the actual address of the buffer to which the JSON text output is to be written.</p> <p>Note: Specifying a bad address for the JSON text output buffer that is other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
502 HWTJ_JSERI_NEWJTXTBUFFLEN_INV	1282 HWTJ_JSERI_NEWJTXTBUFFLEN_INV	<p>Meaning: Program error. The caller specified a value of zero for the length of the new JSON text output buffer.</p> <p>Action: Check for a probable coding error. Specify the actual length of the JSON text output buffer.</p> <p>Note: Specifying a bad length for the JSON text output buffer that is other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 108. Return codes for the HWTJSERI service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	Meaning: The operating system level does not support this service. The system rejects the service request. Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTJSRCH – Search

Call the HWTJSRCH service to search for a particular name string.

JSON text is organized by name or value pairs in the form "*string*":*value*, where "*string*" is a descriptor of the value, and where *value* can be a string in double quotation marks, a number, a boolean value (true, false), a null value, an object, or an array. These structures can be nested.

It is often useful to quickly search for a particular name string in the JSON text. If a string is found, the traversal methods may then be useful to traverse items contained within the found object entry or in subordinate objects.

Description

The HWTJSRCH service searches for a particular "*name*" string within the entire JSON text or within a specific object. The starting point for the search can be either from the beginning of the text or from a specified handle start point.

Notes:

1. JSON array entries are simply a sequence of comma-separated values of any data type. As array entries do not have a "*name*" string, they cannot be searched by the HWTJSRCH service; however, the name of the array object itself can be searched, and nested objects with "*name*" strings within the specified array.
2. A global search can be used for searching through JSON text read by the parse service (HWTJPARS). Any text added to the string via the create service (HWTJCREN) cannot be searched globally, but might be searched with an object-scoped search.
3. The search string provided to HWTJSRCH should be in the same encoding as that which is currently in effect for the JSON parser instance (the same encoding as that of the data input(s) to the JSON parser).
4. The depth of the search can be restricted (see HWTJ_SEARCHTYPE_SHALLOW, and [“Examples” on page 569](#), below).

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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 470 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJSRCH service

All information for the HWTJSRCH service applies for REXX requests except:

- SearchString replaces SearchStringAddr and SearchStringLength

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTJSRCH(ReturnCode, ParserHandle, SearchType, SearchStringAddr, SearchStringLength, ObjectHandle, StartingHandle, SearchResultHandle, DiagArea);</pre>	<pre>address hwtjson "hwtjsrch", "ReturnCode", "ParserHandle", "SearchType", "SearchString", "ObjectHandle", "StartingHandle", "SearchResultHandle", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

- ReturnCode**
Returned parameter.
- **Type:** Integer (non-REXX), character representation of an integer (REXX)
 - **Length:** 4 bytes (non-REXX)
- Contains the return code from the service.
- ParserHandle**
Supplied parameter.
- **Type:** Character string

- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

SearchType

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies the scope of the search. Valid values are:

HWTJ_SEARCHTYPE_GLOBAL

Search the JSON text, starting at (but not including) the entry represented by the **startingHandle** parameter, for the first "*name*" that exactly matches the SearchString for REXX or the string pointed to by the SearchStringAddr parameter for non-REXX. The search is object-ignorant, meaning that there is no scoping of the search to be within the object specified by the **startingHandle** parameter. The search ends when either the search string is found or the end of the JSON text is reached, whichever occurs first.

HWTJ_SEARCHTYPE_OBJECT

Search the JSON text, starting at (but not including) the entry represented by the **startingHandle** parameter, for the first "*name*" within the object specified by the **objectHandle** parameter that exactly matches the SearchString for REXX or the string pointed to by the SearchStringAddr parameter for non-REXX. The search ends when either the search string is found or the end of the object is reached, whichever occurs first.

HWTJ_SEARCHTYPE_SHALLOW

Like HWTJ_SEARCHTYPE_OBJECT, but with limited depth of search, a shallow search will not consider content within any nested object(s) of that object which defines the scope of the search. Arrays may not be shallow searched, and starting handles should not reference content within nested objects.

SearchString (REXX)

Supplied parameter.

- **Type:** Character string

Specifies the REXX variable that contains the search string name to be searched.

SearchStringAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of the location of the name string that the parser should attempt to locate within the JSON text.

Note: To search for an empty name string (" "), specify a **searchStringAddr** value of zero. (In this case, the value of **searchStringLen** must also be zero).

SearchStringLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length (in bytes) of the search string pointed to by the **searchStringAddr** parameter.

Note: To search for an empty name string (" "), specify a **searchStringLen** value of zero. (In this case, the value of **searchStringAddr** must also be zero).

ObjectHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)

- **Length:** 4 bytes (non-REXX)

Specifies a handle representing a particular JSON object (object or array object) at which to start the search. An **objectHandle** is either a value of zero for the root object, or an **entryValueHandle** whose JSON type is HWJT_OBJECT_TYPE or HWJT_ARRAY_TYPE. This parameter must be specified for a **searchType** of HWJT_SEARCHTYPE_OBJECT and must be set to zero for a searchType of HWJT_SEARCHTYPE_GLOBAL.

StartingHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies any handle value returned by a prior parser service call, or a value of zero.

If the **searchType** is HWJT_SEARCHTYPE_GLOBAL, the search starts at (but not including, unless the value is zero) the specified handle within the JSON text.

If the **searchType** is HWJT_SEARCHTYPE_OBJECT, the search starts at (but not including, unless the value is zero) the specified handle, if the **startingHandle** is either within the object specified by **objectHandle** or the **startingHandle** is zero (start searching at the beginning of the object).

If multiple instances of the same name string occur within the search scope, the **searchResultHandle** that is returned on one invocation of the search service can be used as the **startingHandle** for the next invocation.

SearchResultHandle

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

The returned handle that represents the next instance of the search string found after the specified starting point.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that may contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

If the string is not found, a "**not found**" return code is returned. If there are multiple strings with the same name string value, only the first is returned. The caller can issue another search request to find the next instance of this name string.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'000Byyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 109 on page 565](#).

Table 109. Return codes for the HWTJSRCH service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	Meaning: Program error. Two possible reasons can result in this return code: <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. Action: Check for a probable coding error. <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle might be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.

Table 109. Return codes for the HWTJSRCH service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	260 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle parameter is not valid, or a nonzero value objectHandle value was specified for a searchType of HWTJ_SEARCHTYPE_GLOBAL.</p> <p>Action: Check for a probable coding error (for example, uninitialized handle or a reference to a deleted entry).</p> <p>If the searchType is HWTJ_SEARCHTYPE_OBJECT, then pass either:</p> <ul style="list-style-type: none"> • An object handle or entry value handle on the objOrEntryHandle parameter that was returned on a prior z/OS JSON parser service call • A value of zero for the root object <p>If the searchType is HWTJ_SEARCHTYPE_GLOBAL, specify a value of zero for the objectHandle parameter.</p>
105 HWTJ_HANDLE_TYPE_ERROR	261 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified objectHandle does not represent an object or array object (JSON data type of HWTJ_OBJECT_TYPE or HWTJ_ARRAY_TYPE).</p> <p>Action: Check for coding or usage error. For object searches, correct the mismatched handle and specify an objectHandle value that represents an object or array. For shallow searches, correct the mismatched handle and specify an objectHandle value (for example, shallow search may not be performed on an array).</p>

Table 109. Return codes for the HWTJSRCH service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
10A HWTJ_ROOT_OBJECT_MISSING	266 HWTJ_ROOT_OBJECT_MISSING	<p>Meaning: Program error. There is no JSON text to search.</p> <p>Action: Check for a probable coding error. Invoke the parse service (HWTJPARS) to associate JSON text with the specified parser instance before invoking the search service (HWTJSRCH).</p>
601 HWTJ_JSRCH_SEARCHTYPE_INV	1537 HWTJ_JSRCH_SEARCHTYPE_INV	<p>Meaning: Program error. The caller specified an invalid searchType.</p> <p>Action: Check for a probable coding error. The caller should change the searchType value to one of the valid values.</p>
602 HWTJ_JSRCH_SRCHSTRADDR_INV	1538 HWTJ_JSRCH_SRCHSTRADDR_INV	<p>Meaning: Program error. The caller specified a value of zero for the address of the search string buffer.</p> <p>Action: Check for a probable coding error. Specify the actual address of the buffer containing the search string.</p> <p>Note: Specifying a bad search string buffer address other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
603 HWTJ_JSRCH_SRCHSTRLEN_INV	1539 HWTJ_JSRCH_SRCHSTRLEN_INV	<p>Meaning: Program error. The caller specified a value of zero for the length of the search string buffer.</p> <p>Action: Check for a probable coding error. Specify the actual length of the search string buffer.</p> <p>Note: Specifying a bad search string buffer length other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
604 HWTJ_JSRCH_SRCHSTR_NOT_FOUND	1540 HWTJ_JSRCH_SRCHSTR_NOT_FOUND	<p>Meaning: The name string was not found in the search scope specified by the caller. If the searchType was HWTJ_SEARCHTYPE_GLOBAL, the name string was not found anywhere from the startingHandle to the end of the JSON text. If the searchType was HWTJ_SEARCHTYPE_OBJECT, the name string was not found anywhere in the object specified by objectHandle (from the startingHandle to the end of the object).</p> <p>Action: Check for a probable coding error. If the string was supposed to be found, verify that the searchType, startingHandle, and objectHandle (if applicable) are specified correctly. If all of these values are correct and the name string still cannot be found, verify that the JSON text being parsed actually contains the name string that the caller specified.</p>

Table 109. Return codes for the HWTJSRCH service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
605 HWTJ_JSRCH_STARTINGHANDLE_INV	1541 HWTJ_JSRCH_STARTINGHANDLE_INV	<p>Meaning: Program error. The specified startingHandle value is not valid for one of the following reasons:</p> <ul style="list-style-type: none"> • The caller specified a bad handle value that was not returned by a prior z/OS JSON parser service call. • If the searchType is HWTJ_SEARCHTYPE_OBJECT, the startingHandle value is either not zero or not the value of a valid handle within the object specified by the objectHandle parameter. • If the searchType is HWTJ_SEARCHTYPE_SHALLOW, the startingHandle value is either not zero or not the value of a direct child of the object specified by the objectHandle parameter. <p>Action: Check for a coding or usage error. Validate that the startingHandle parameter contains either a zero or a valid handle. If the searchType is HWTJ_SEARCHTYPE_OBJECT, verify that the startingHandle is within the object specified by the objectHandle parameter. If the type is HWTJ_SEARCHTYPE_SHALLOW, verify that the startingHandle designates one of the direct children of the object specified by the objectHandle parameter (see Example 1: Shallow Search of “Examples” on page 569 below).</p>
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Examples

Example 1: Shallow Search

Consider the following JSON text example to clarify the terms descendants and direct children, and the distinction between them:

```
{
  "a": "A1",
  "b": {
    "c": "C1",
    "d": {
      "c": "C2",
      "e": "E1",
      "f": "F1"
    },
    "e": "E2"
  },
  "c": "C3"
}
```

The descendants of the root object are: a (with value A1), b (object), c (with value C1), d (object), c (with value C2), e (with value E1), f (with value F1), e (with value E2), and c (with value C3).

The direct children of the root object are: a (with value A1), b (object), and c (with value C3).

A search of type HWTJ_SEARCHTYPE_OBJECT for "a", under the root object, yields the handle of object entry a (with value A1), as would a search of type HWTJ_SEARCHTYPE_SHALLOW.

A search of type HWTJ_SEARCHTYPE_OBJECT for "c", under the root object, yields the handle of object entry c (with value C1), whereas a search of type HWTJ_SEARCHTYPE_SHALLOW for "c", under the root object, yields the handle of c (with value C3).

A search of type HWTJ_SEARCHTYPE_OBJECT for "d", "e", or "f", under the root object, yields the handle of d (object), e (with value E1), or f (with value F1), respectively, whereas a search of type HWTJ_SEARCHTYPE_SHALLOW for d, e, or f, under the root object, yields a HWTJ_JSRCH_SRCHSTR_NOT_FOUND return code from HWTJSRCH in each case.

HWTJTERM – Terminate a parser instance

Call the HWTJTERM service to clean up resources obtained by a previous HWTJINIT invocation.

Description

When the services of a z/OS JSON parser instance are no longer needed, this service cleans up the resources in use by that parser instance. If this service is not invoked, the memory space allocated by the initialization and other parser services is allocated and ineligible for use by the application, and remains allocated until the job step task of the address space terminates.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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.

Requirement**Details****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 470](#) for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJTERM service

All information for the HWTJTERM service applies for REXX requests except:

- ForceOption is not used

Syntax

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

Non-REXX parameters	REXX parameters
CALL HWTJTERM(ReturnCode, ParserHandle, ForceOption, DiagArea);	address hwtjson "hwtjterm", "ReturnCode", "ParserHandle", "DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be cleaned up. The **parserHandle** value was returned on a previous HWTJINIT service call for the particular JSON parser instance.

ForceOption (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Controls the behavior of the HWTJTERM service. Sometimes a parser instance can be stuck in an in-use state and cannot be terminated successfully. The in-use state can occur if a prior z/OS JSON parser service call resulted in an ABEND condition. This option allows the caller to force the parser instance to terminate. The valid values are:

HWTJ_NOFORCE

(*Recommended*) Terminates the parser instance and invalidates its parser handle only if the parser instance is not in an in-use state.

HWTJ_FORCE

Unconditionally terminates the parser instance and invalidates its parser handle, regardless of the in-use status of the parser instance.

Attention: Only use the HWTJ_FORCE option under the following conditions:

- No other threads in the address space are using this parser instance
- Multiple attempts to terminate the parser instance have resulted in a return code of HWTJ_PARSERHANDLE_INUSE.

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'000Cyyyy' for one of the following reasons:

yyyy**Reason****0000**

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 110 on page 571](#).

Table 110. Return codes for the HWTJTERM service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTJ_OK	0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	257 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The specified parserHandle parameter is not a valid parser handle (that is, one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 110. Return codes for the HWTJTERM service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	258 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	259 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 471 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 471 for details about actions to consider for this return code.</p>

Table 110. Return codes for the HWTJTERM service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
701 HWTJ_JTERM_CANNOT_FREE_WORKA	1793 HWTJ_JTERM_CANNOT_FREE_WORKA	<p>Meaning: System error. The Storage Release service could not release the work area storage as requested by the z/OS JSON parser.</p> <p>Action: Consult the diagArea for the Storage Release failure return code and additional information found in the ReasonDesc section. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
702 HWTJ_JTERM_FORCEOPTION_INV	1794 HWTJ_JTERM_FORCEOPTION_INV	<p>Meaning: Program error. The caller specified an invalid forceOption value.</p> <p>Action: Check for a probable coding error. The caller should change the forceOption value to one of the valid values, as described in “Parameters” on page 570.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	3841 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	3842 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	3843 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	4095 HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Chapter 21. The z/OS HTTP/HTTPS protocol enabler

The HTTP/HTTPS protocol enabler portion of the z/OS client web enablement toolkit allows z/OS applications running in traditional environments to more easily participate as a web services client application.

Developers of new and existing z/OS applications can "webify" their applications using an industry standard API model of communicating with a web server. The enabler is a lightweight API that implements portions of the Hypertext Transfer Protocol 1.1 (HTTP/1.1), as specified by [RFC 7230](https://tools.ietf.org/html/rfc7230) (tools.ietf.org/html/rfc7230), [RFC 7231](https://tools.ietf.org/html/rfc7231) (tools.ietf.org/html/rfc7231), [RFC 6265](https://tools.ietf.org/html/rfc6265) (tools.ietf.org/html/rfc6265), and others. The toolkit also provides support for HTTP Secure (HTTPS), which layers HTTP over the Secure Sockets Layer (SSL) / Transport Layer Security (TLS) protocols to provide secure communications of standard HTTP requests over an open communications network.

HTTP/HTTPS enabler basics

The HTTP/HTTPS enabler portion of the toolkit encompasses two major aspects of a web services application:

- The *connection* to a server
- The *request* made to that server along with the response it returns

Connections

A toolkit connection is simply the network socket (pipeline) created between a client and server application for the purpose of exchanging information. A connection employs TCP sockets connecting an application to a remote IP address and port from an optionally-specified local IP address and port. This connection can either be established directly from a local IP address to the remote IP address, or can utilize a proxy server (an intermediary gateway that serves as a method of providing enhanced security and privacy controls). You can choose a standard socket connection or a secure connection that utilizes the SSL/TLS protocol. (See [“Security considerations”](#) on page 580.)

A connection must be established prior to any request being sent to the server. Once established, a connection can be dropped due to a variety of factors. The toolkit will attempt to automatically reconnect to resume communications if it detects that the connection has been dropped.

An application may explicitly disconnect a connection. A successful disconnection causes all SSL and socket resources to be released.

Requests

In toolkit terms, a request is simply an HTTP request sent over a previously established connection. The toolkit provides some flexibility in the creation and usage of requests. Requests are not tightly coupled to a particular connection. Instead, a created request can be used with one or more connections. In addition, multiple requests can be created and kept in memory, each of which uses the same connection.

Requests can be specified with numerous options to customize the HTTP processing to the application's preferences. A request method is selected along with the URI path name of the resource to be targeted by the HTTP method. There are additional options to control the operation of the HTTP request. The following options are supported:

- Basic HTTP client authentication
- URI redirection (forwarding of HTTP requests from one URI to another)
- HTTP cookie management (a mechanism for the toolkit's cookie engine to maintain stateful information sent by the server on behalf of the client, which then can be sent back to the server on subsequent requests)
- Use of specific HTTP request headers

After a request is configured, it can be sent to the server by temporarily coupling itself with an existing connection on the send request call. Two configurable callback routines (user exits) can be set up to handle the response coming back from the server:

- A routine to receive control for each response header that is received
- A routine to receive control when a response body is present

The toolkit provides services that allow easy access to sockets, System SSL, and HTTP/HTTPS functionality. To accomplish this, the toolkit makes available to applications a set of services with a similar interface as the easy-to-use, open-source libcurl programming interface. Applications do not have to deal with most of the intricacies of socket, SSL, or HTTP programming. The toolkit takes care of many of these nuances, allowing application programs to have a higher level of abstraction.

Elements of the z/OS HTTP/HTTPS enabler

The z/OS HTTP enabler is organized into five types of services:

- **Initialize, reset, and terminate:** The purpose of these services is to prepare the memory space required by the z/OS HTTP enabler, to reset the space in preparation for reuse by a subsequent connection or request, or to free the space after these services are no longer needed. The memory allocation created by the initialize service is known as a *connection* or *request* instance, depending on the type of instance being created.
- **Set options:** This service prepares a connection or request instance with the desired configuration options. A user invokes this service multiple times, once for each option to be set. When all of the options have been set for the connection or request instance, the connect or send request service can be called to perform the respective operation, as specified by the various options that have been set.
- **Connect and disconnect:** These services are for connection instances and are used to establish or to disconnect a socket connection (pipeline) between the client and the server. If SSL/TLS use has been configured, these services will handle all SSL interactions to bring up or tear down the SSL connection. Once a connection has been created, it can then be used by the send request service to flow an HTTP request to the server connected.
- **Send request:** This service couples a request with a connection. A previously initialized and configured HTTP request is sent over a previously initialized and configured connection. The HTTP response from the server can be handled through the use of the HTTP response header and HTTP response body callback (exit) routines.
- **Set link list service:** This is a utility service that creates a linked list of data objects of the same type. This linked list data type is required by any option which expects a variable number of items to be associated with that attribute (as described in the following example). A toolkit application will first create the link list (the HWTH_SLST_NEW function) to add the first item to the linked list. Additional items can be added (the HWTH_SLST_APPEND function) or the entire linked list can be deleted (the HWTH_SLST_FREE function).

Example: An application using the toolkit has the option to directly send particular HTTP headers (HWTH_OPT_HTTPHEADERS) to the server. Because it is reasonable that the application would want to specify more than one HTTP header to send, this option requires the headers to be in SLST format. The application will first create the linked list (SLST) specifying the first HTTP request header to be added to the request. If more request headers are needed, it will append those to the end of the previously created SLST. When all of the request headers are added, it will issue the HWTHSET API to set this option to the newly created SLST. The result is that the HWTH_OPT_HTTPHEADERS option is set to all of the headers specified by the linked list.

The general usage of the z/OS HTTP enabler services in an application follows this general order:

1. Create a connection instance, which returns a connection handle. (HWTHINIT)
2. Set the necessary connection options, such as URI of the server, SSL options, and so on, associated with this connection instance. (HWTHSET)
3. Connect to the server. (HWTHCONN)

4. Create a request instance, which returns a request handle. (HWTHTINIT)
5. Set the necessary request options, such as the request type, server resource, HTTP response callback exits, and so on, associated with this request instance. (HWTHTSET)
6. Send the previously defined request over the previously defined connection. If configured by the application, HTTP response routines are called during this service call to process the response data. (HWTHTRQST)
7. Disconnect the previous connection. (HWTHTDISC)
8. Free the work area associated with the request. (HWTHTTERM)
9. Free the work area associated with the connection. (HWTHTTERM)

Availability of the z/OS HTTP/HTTPS enabler

The z/OS HTTP enabler contained the z/OS client web enablement toolkit is available to almost any address space. The toolkit is enabled as part of z/OS initialization during IPL time. A message is written to the syslog regarding the status of the toolkit. Success or failure of toolkit initialization can be found by finding any HWT- prefixed syslog messages issued during IPL.

Syntax, linkage, and programming considerations

The z/OS HTTP enabler is available to many programs running in various address spaces. Many z/OS execution environments are supported, as well as various programming languages.

Programming interface files provided by the HTTP enabler

Table 111 on page 577 lists the programming interface files provided by the z/OS HTTP enabler.

Table 111. HTTP enabler

Programming language	Programming interface file
C / C++	Include file HWTHTIC provided in SYS1.SIEAHDRV.H and under z/OS UNIX /usr/include directory as hwthic.h
COBOL	Copybook file HWTHTICOB provided in SYS1.MACLIB
PL/I	Include file HWTHTIPLI provided in SYS1.MACLIB
Assembler	Include file HWTHTIASM provided in SYS1.MACLIB
REXX	See “HWTCONST — Initialize predefined variables (REXX)” on page 595 on how to access all the toolkit constants in REXX

Calling formats

Table 112 on page 577 lists specific calling formats for languages that can invoke the z/OS HTTP enabler callable services.

Table 112. Calling formats for the z/OS HTTP enabler callable services

Programming language	Calling format
C / C++	<i>HTTPenabler_service_name (return_code, parm1, parm2, ...)</i>
COBOL	<i>CALL HTTPenabler-service-name USING return_code, parm1, parm2, ...</i>
PL/I	<i>CALL HTTPenabler_service_name (return_code, parm1, parm2, ...)</i>
Assembler	<i>CALL HTTPenabler_service_name (return_code, parm1, parm2, ...),VLIST</i>
REXX	<i>ADDRESS HWTHTTP "HTTPenabler_service_name return_code parm1 parm2..."</i>

Linkage considerations

There are two ways for a compiled application to find the z/OS HTTP enabler callable services:

Linkage stub method

(Recommended) Use the linkable stub routine HWTHCSS from SYS1.CSSLIB to link edits your object code. If you attempt to run the z/OS web enablement toolkit on a previous release of z/OS that does not support the z/OS HTTP enabler, this method results in the service call receiving a return code of X'F06' (HWTH_UNSUPPORTED_RELEASE).

Direct linkage method

Code the linkage to the z/OS HTTP enabler services directly. This can be done if the program first confirms that the level of z/OS contains the toolkit. The following example shows the assembler linkage:

```
L R14,CVTCST-CVT(R14,0)
L R14,88(R14,0)
L R15,4*HWT_SERV_XXXXX(R14,0)
BASR R14,R15
```

In the example, XXXXX represents the last five letters of the service you want to call. This requires that the HWTHKASM assembler macro be included. If you attempt to run the HTTP enabler on a previous release of z/OS that does not support the HTTP enabler, this method results in the application receiving an abend X'019' with a reason code 00000000.

Linkage considerations for high-level language programming

Callers must ensure that the proper linkage is made to the HTTP enabler services. The supplied IDF files for the various high-level languages contain the necessary definitions that ensure that the parameter list passed to the HTTP enabler has the high-order bit turned on for the last parameter. For example, for C, the linkage must be specified as OS linkage, such as:

```
#pragma linkage(HWTHXXXX_CALLTYPE,OS)
```

For PL/I, the entry declaration should have the following options defined:

```
OPTIONS(LINKAGE(SYSTEM))
```

Linkage considerations for assembler language programming

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 - 14 are restored (registers 0, 1 and 15 are not restored).

Compilation consideration

The z/OS HTTP enabler provides one or more sample programs in most of the supported languages to aid in the creation of applications that use the toolkit functions. Refer to the sample JCLs in the prolog

section of the sample of your language choice for the recommended compiler and linking options. See [“z/OS HTTP enabler programming examples” on page 589](#) for details about the sample programs.

Code page consideration

All input data into the HTTP enabler, except for body data, is assumed to be in EBCDIC encoding (code page 1047) before making any z/OS HTTP enabler service call. The toolkit translates any required data to meet any RFC code page standards when sending or receiving HTTP header data. Data specified in the request body or received in the response body is treated as-is unless the application requests the toolkit to translate from EBCDIC to ASCII or vice versa (see the documentation for `HWTH_OPT_TRANSLATE_REQBODY` and `HWTH_OPT_TRANSLATE_RESPBODY` later). Other than these options, the toolkit makes no attempt to translate this data into any format. It is the responsibility of the application to translate the data in either body into the correct encoding.

Environmental considerations

The following environmental considerations apply:

OMVS segment required: The toolkit uses TCP/IP sockets and Cryptographic Services System SSL services. Because both TCP/IP sockets and the SSL services require a z/OS UNIX (POSIX) environment, the HTTP enabler runs with a Language Environment POSIX(ON) environment. A POSIX(ON) environment requires the user ID associated with the address space using the HTTP enabler services to have an OMVS segment defined and associated with it. See the appropriate security product documentation, as applicable to your installation, for instructions on how to define an OMVS segment to a user. Failure to properly define the OMVS segment for the user invoking the HTTP enabler will likely result in an `HWTH_ENVIRONMENTAL_ERROR` return code.

z/OS UNIX limit of processes with a POSIX(ON) environment and its effect on concurrent connections: z/OS UNIX limits the number of concurrent POSIX(ON) environments that can be defined to a single address space. Since each HTTP enabler connection attempts to initialize a new POSIX(ON) environment (because of the environmental requirements listed earlier), each connection gets implicitly dubbed by Language Environment. (*Dub* means to make a z/OS address space known to z/OS UNIX System Services. Once dubbed, an address space is considered to be a process.) If multiple connections are wanted from the same application (address space), a consideration of the dubbing configuration for the address space in which the HTTP enabler is running may be necessary.

The dubbing behavior that z/OS UNIX takes is customizable by using the z/OS UNIX set dub default service (`BPX1SDD` or `BPX4SDD`). If the dub default for the address space is set to `DUBPROCESS`, this allows concurrent subtasks to each have their own POSIX(ON) environment and, thereby, allow multiple connections from within the same address space, provided that each subtask has at most one connection.

z/OS Language Environment Heap runtime option considerations: The toolkit obeys the current Language Environment Runtime HEAP option for storage management. Applications can customize the HEAP storage option for their execution environment as needed. IBM does not recommend using a primary heap size that is smaller than 32K. Applications which make multiple requests per connection, or long running applications where multiple connections and requests are being made, may benefit from a larger PRIMARY HEAP size. These applications should either:

1. set their PRIMARY HEAP size to allow plenty of room for the largest anticipated messages, or
2. specify the FREE option on HEAP, so that Language Environment will release unused heap extent storage once there are no longer references to it.

For details, see [z/OS Language Environment Customization](#).

z/OS Language Environment Runtime Environment REUSE (RTEREUS) option consideration: COBOL applications running in Information Management System (IMS) should not use the RTEREUS(ON) runtime option. For details, see RTEREUS runtime option in [z/OS Language Environment Customization](#).

Security considerations

There are several aspects of security to consider at both the connection level and the request level.

Connection security: Considerations for connection security include:

TCP/IP stack and security product control

A toolkit application connecting to a server is limited by security profiles and definitions that are already in effect on the system where the application resides. Powerful controls, such as z/OS Communications Server NetAccess, in conjunction with security profiles defined using the SERVAUTH class, can be used to control network access authority, TCP/IP stack access authority, port access authority, and more. These security definitions can be configured to be as granular as required by an installation. See *z/OS Communications Server: IP Configuration Guide* and *z/OS Communications Server: IP Configuration Reference* for more information.

SSL/TLS connections

Before connecting to a HTTP server, the toolkit consults values of the HWTH_OPT_URI, HWTH_OPT_USE_SSL and HWTH_OPT_PORT user-specified options to decide whether the toolkit application expects a HTTP or HTTPS connection. Whenever the value of HWTH_OPT_URI begins with an explicit scheme prefix (either 'http://' or 'https://'), the scheme is honored; otherwise, the toolkit attempts an HTTPS connection when either HWTH_OPT_USE_SSL option is set to HWTH_SSL_USE or HWTH_OPT_PORT is set to 443 (the default port for HTTPS). See “HTTP/HTTPS enabler options and values” on page 643 for more information.

There are two ways that a toolkit application can acquire a SSL/TLS connection: Application Transparent Transport Layer Security (AT-TLS) initiated or Application initiated.

Application Transparent Transport Layer Security (AT-TLS) initiated

Connections can be secured by an AT-TLS policy without requiring applications to initiate connection security. The processing that enables this security depends on whether you use a proxy (**HWTH_OPT_PROXY**) to connect to your HTTPS destination (**HWTH_OPT_URI**).

HTTPS direct (no proxy)

When directly configuring toolkit connections to an HTTPS server (**HWTH_OPT_URI**), the AT-TLS administrator has two options for providing transparent security:

1. Map the connection to an enabled policy
2. Map to an 'application-controlling' policy

Each type of policy must specify an AT-TLS keyring that is suitable for use with the HTTPS server. There is an important difference between the two policy types:

An 'application-controlling' policy defers the TLS handshake allowing the toolkit the flexibility to use its own toolkit options (**HWTH_OPT_SSLKEY** and others), or the AT-TLS keyring (from the mapped policy), for the TLS handshake.

An enabled policy attempts the handshake immediately, using the AT-TLS keyring. This is usually the desired behavior, unless the toolkit application chose to specify its own keyring options (**HWTH_OPT_SSLKEY** and others), in which case the connection fails with a communications error.

HTTPS via proxy

When configuring toolkit connections that use a proxy (**HWTH_OPT_PROXY**) to reach an HTTPS destination server (**HWTH_OPT_URI**), the AT-TLS administrator must map the proxy connection to an 'application-controlled' policy. This policy must include an AT-TLS keyring that is suitable for use with any HTTPS destination (**HWTH_OPT_URI**) to which the proxy will connect.

Unlike the direct HTTPS connection case, it is not possible to develop AT-TLS rules that select a policy based on the destination (**HWTH_OPT_URI** and/or **HWTH_OPT_PORT**). However, the other policy selection criteria (user name, job name, local IP, and local port) are still useful for associating a proxy connection with an appropriate keyring.

For more information, see “Using the toolkit with AT-TLS” on page 590.

Application initiated

If requested by the application, the toolkit creates an SSL/TLS connection instead of a standard socket connection. These protocols provide data privacy and integrity, including server and client authentication, that is based on public key certificates. The toolkit uses z/OS System SSL services (part of the z/OS Cryptographic Services base component) to facilitate these SSL/TLS connections. (The term *SSL* is used throughout this publication to describe both the SSL and TLS protocols.) The certificate store configuration required by System SSL must be set up before making an SSL connection. The toolkit supports certificates stored in a key database file, a RACF key ring, or as a z/OS PKCS #11 token.

Note: The previous two security approaches are mutually exclusive. Applications must only use one of the approaches for their connections. When both are detected, the toolkit will fail the HWTHCONN request with a return code of HWTH_COMMUNICATION_ERROR.

Proxy security

The toolkit implicitly supports basic client authentication to an authenticating proxy, that packages a user ID and password into a format specified by RFC 1945 (tools.ietf.org/html/rfc1945). The **HWTH_OPT_PROXYAUTH**, **HWTH_OPT_PROXYAUTH_USERNAME** and **HWTH_OPT_PROXYAUTH_PASSWORD** settable options can be used to have the toolkit automatically create the necessary HTTP header. See “Using an authenticating proxy server” on page 582.

Request security: While an HTTP request flows over a secure SSL/TLS connection, the payload of the HTTP request is private between the client and the server. HTTP provides additional protocols to allow a user to authenticate with the server. In many cases, these additional authentication methods flow over an SSL connection exclusively to make sure that the security credentials are flowed from the client to the server in a private manner.

The toolkit implicitly supports *basic client authentication*, which packages a user ID and password in a format specified by RFC 1945 (tools.ietf.org/html/rfc1945). The **HWTH_OPT_HTTPAUTH**, **HWTH_OPT_USERNAME**, and **HWTH_OPT_PASSWORD** settable options can be used to have the toolkit automatically create the necessary HTTP header.

Additional authentication schemes can be used explicitly by providing the specific HTTP headers and data. Authentication cookies can sometimes be used as a method by web servers to know whether the user is logged in or not, and the account with which they are logged in.

Using a proxy server

Client applications using the z/OS web enablement toolkit (toolkit) in a controlled network environment are often required to communicate with the outside world via a proxy server. Preparing a toolkit application to use an HTTP proxy server is designed to be as simple as setting the **HWTH_OPT_PROXY** and **HWTH_OPT_PROXYPORT** options on the initialized connection handle. However, your HTTP proxy server may require Basic authentication. In this case, the **HWTH_OPT_PROXYAUTH**, **HWTH_OPT_PROXYAUTH_USERNAME**, and **HWTH_OPT_PROXYAUTH_PASSWORD** additional toolkit options are required. See “Using an authenticating proxy server” on page 582.

Connecting to HTTPS via a proxy: Client applications connecting outside an internal network will likely be required to communicate using the secure form of HTTP, known as HTTPS. The toolkit provides two methods to establish HTTPS security by either using toolkit options or by relying on Application-Transparent Transport Level Security (AT-TLS).

The choice of these two methods depends on the value of the settable toolkit option **HWTH_OPT_USE_SSL**. A setting of **HWTH_SSL_USE** instructs the toolkit to use its own options to perform the TLS handshake to secure the end-to-end HTTPS connection. A setting of **HWTH_SSL_NONE** instructs the toolkit to depend on AT-TLS for that handshake.

A proxy connection that maps to an AT-TLS policy type of 'application-controlled' allows either of these approaches to be used successfully through the same proxy server.

Using an authenticating proxy server

The toolkit supports the type of proxy that requires its own Basic authentication, commonly known as an "authenticating proxy." This type of proxy typically resides inside an enterprise firewall and requires each client request to include a Proxy-Authorization message header. Without this special request header, the proxy server returns a client error status code of 407 Proxy-Authentication-Required.

If the header is supplied with the request message, and the contents of the header satisfy the authenticating proxy, the request proceeds as expected. However, if the attached header does not satisfy the proxy, or something else is wrong upstream from the proxy, the request may fail with another client error status code of 407 or with another 4xx (client) or 5xx (server) error.

Note: The formal specification of the 407 Proxy Authentication Required client error can be found in [RFC 7235 \(tools.ietf.org/html/rfc7235\)](https://tools.ietf.org/html/rfc7235). This specification allows a client application to retry using a modified authentication header after a 407 client error occurs. The toolkit does not attempt to automate this dynamic retry process. However, the toolkit provides enough diagnostic information to reliably identify a 407 client error so that a more dynamic response is possible.

See the following suggested approaches about using an authenticating proxy with the toolkit and the important differences between the HTTP and HTTPS destination servers.

How to avoid the 407 error with the toolkit

To avoid the 407 error, the toolkit user needs to correctly set the **HWTH_OPT_PROXYAUTH**, **HWTH_OPT_PROXYAUTH_USERNAME**, **HWTH_OPT_PROXYAUTH_PASSWORD** options on their connection. For more information about options, see "HTTP/HTTPS enabler options and values" on page 643. Once these options are set, a call to either the **HWTHCONN** or **HWTHRQST** callable service will build and attach a corresponding header to the message when required. For an HTTPS server connection, the **HWTHCONN** service attaches the new header once to open a tunnel through the proxy. For an HTTP server connection, the **HWTHRQST** service attaches the header every time the request is made.

What to do if your specified options do not work

If the toolkit user specifies the options but still gets a client error, such as 407, 404, or 401, the diagnosis depends on whether the server type is HTTP or HTTPS. Failing HTTP requests will return from **HWTHRQST** after calling the header and body exits. Failing HTTPS connection attempts will return the **HWTH_COMMUNICATION_ERROR** from **HWTHCONN**, with up to 128 bytes of descriptive information in the diagnostics area. For persistent 407 errors to HTTPS, it may be useful to connect to a plain HTTP destination through the same proxy, so that the complete error description in the response body is available to your response body exit.

How HTTP and HTTPS are handled differently

As previously described, a 407 client error raised by an authenticating proxy is seen by a toolkit application at different times, and depends on whether the connection's destination URI is an HTTP or HTTPS server.

For an HTTPS destination server, **HWTHCONN** uses the **PROXYAUTH** options to open a tunnel to the destination server. For an HTTP destination server, every call to **HWTHRQST** uses the **PROXYAUTH** options, for the life of the connection.

What it means to use a tunnel through a proxy

When a toolkit application is connecting to an HTTPS server through a proxy, the **HWTHCONN** service immediately asks the proxy server to open a direct connection to the **HWTH_OPT_URI** destination server.

Note: If the proxy requires authentication, it is required during connect processing so that any 407 client error to an HTTPS server will occur at this point.

When the proxy server agrees to open a direct connection, it also agrees to ignore all traffic over the connection, effectively routing it through an internal tunnel.

What a handshake is

A handshake is a sequence of operations that establishes a secure connection so that all data passing between two endpoints is encrypted. When you first open a tunnel, it is not yet secure. The first data to flow through a new tunnel must be a special message, called a Client-Hello, that initiates the TLS handshake with the server. If the security negotiations of the handshake succeed, the end-to-end

HTTPS connection is immediately ready for secure traffic. Otherwise, the connection is closed and the tunnel collapses before any traffic can flow.

What to do if the authenticating proxy refuses to open a tunnel

If the proxy server refuses to open the tunnel for any reason, including a 407 Proxy Authentication Required client error, **HWTHCONN** capture the details and fails immediately with a communications error. Therefore, when you are using an authenticating proxy to reach an HTTPS destination, it is always a call to the **HWTHCONN** service that causes a 407 client error.

Why HTTP through a proxy does not do a handshake

HTTP connections do not require a handshake because HTTP is not a secure protocol. When the **HWTHCONN** service is called to connect to HTTP through a proxy it has no immediate need to connect to the destination server, so it only connects to the proxy. Connecting to the proxy does not require proxy authentication, nor a valid destination URI, therefore **HWTHCONN** can not report a 407 Proxy-Authentication Required client error. Only an actual attempt to use the proxy, such as occurs with each **HWTHRQST**, can encounter a 407 client error. Therefore, for HTTP destinations, any 407 client error will only result from a call to the **HWTHRQST** service.

Using the toolkit with AT-TLS

For more details on using the toolkit with AT-TLS see [“AT-TLS usage overview” on page 590](#).

Large data body considerations

Under some circumstances, applications require a large data body to be sent on a single request or received on a corresponding response. In these cases, you can supply a streaming send exit that can be used to provide the request body as an ordered sequence of contiguous pieces of data whose number, size, and location are completely at the discretion of the exit. Similarly and independently, applications that expect a very large response body can supply a streaming receive exit to accept the response body as an ordered sequence of contiguous pieces of data of unpredictable number and size.

Although exact limits on request or response body size depend on the particular characteristics of the invocation environment, most general purpose HTTP requests or responses involve data bodies whose sizes are compatible with the limits of the non-streaming methodology of sending and receiving data. For those applications whose requests invariably involve very large and or non-batchable data bodies, streaming send and receive support may address their needs.

Problem determination considerations

Problem determination in a client/server web services application can be challenging. Here are a few debugging options that can aid in the debugging of your application:

Diagnostic area (diagArea): Each HTTP toolkit API requires the application to specify the **diagArea** parameter, a diagnostic output area that contains additional information that can be useful when a service fails with a nonzero return code or even in certain cases when the return code is zero. The **diagArea** includes three pieces of information: **HWTH_Service**, **HWTH_ReasonCode**, and **HWTH_ReasonDesc**.

HWTH_Service

A 4-byte constant value that indicates which internal service invoked by the toolkit detected the possible error. To determine which service, consult the first two bytes of this field and locate that value in the defined service constants provided in the IDFs (include files).

Note: The last two bytes are for IBM and can be provided to IBM Support, if necessary.

HWTH_ReasonCode

A specific error code returned by the internal service identified by **HWTH_Service**. This can be useful in determining the reason why a particular service failed.

HWTH_ReasonDesc

A 128-character field that contains a more detailed description of the problem. In many cases, this information is sufficient to determine the cause of the problem. In some cases, additional information

is also provided here that you can provide to IBM Support if you cannot determine the problem after using the other problem determination techniques described here.

Verbose option: The HTTP enabler provides the option to get more information related to communications and data exchanged between the application and the web server by enabling the **HWTH_OPT_VERBOSE** option (setting this option to **HWTH_VERBOSE_ON** or **HWTH_VERBOSE_UNREDACTED** value using the **HWTHSET** service). Typically, you would use the verbose option during application debugging, then turn it off when the application is in production. You can also select the destination of these trace messages.

If you want to direct these messages to the application's standard output, no further action is necessary. The verbose option may have limitations in some environments and may produce excessive output on your application's standard output. If you want to direct the messages to a particular data set or file, you can use the **HWTHSET** service to set the **HWTH_OPT_VERBOSE_OUTPUT** option. For more information about the **HWTH_OPT_VERBOSE_OUTPUT** option, see [“HTTP/HTTPS enabler options and values”](#) on page 643. In many cases, these messages can help you quickly determine the root cause of a web application problem.

SSL/TLS Security: When an application sets **HWTH_OPT_USE_SSL** option to **HWTH_SSL_USE**, the application chooses to control the client side of HTTPS security. In this case, a number of related HTTP enabler options come into play:

- Some settings are required (**HWTH_OPT_SSLKEY**, **HWTH_OPT_SSLKEYTYPE**, etc.).
- Other settings override System SSL defaults to satisfy server requirements (**HWTH_OPT_SSLVERSION**, **HWTH_OPT_SSLCIPHERSPECS**, etc.).
- One setting triggers System SSL to capture detailed diagnostic output (**HWTH_OPT_SSLTRACE**).

Because System SSL is exclusively responsible for z/OS connection security, all users (including the HTTP enabler) are subject to any change in its behavior. Usually, when a connection begins failing, after working for a long time, it is due to some change in the server's requirements. For instance, a server may choose to begin requiring a stricter security level, or some stronger cipher specification. In those cases, the required application changes involve setting an option or two, fixing the client application across all z/OS releases.

In rare circumstances, an origin server might disallow a client connection from a new z/OS release, while earlier z/OS releases continue to connect without error. If this occurs, and your application does not already override the System SSL default cipherspecs using **HWTH_OPT_SSLCIPHERSPECS**, setting that option to a suitable 4-character cipherspec may provide a backward-compatible circumvention.

SOCKAPI interface: z/OS Communication Server provides a **SOCKAPI** CTRACE option, provided by TCP/IP, which can be used by application programmers to debug problems in their applications. The **SOCKAPI** option captures trace information that is related to the socket API calls that an application might issue. The **SOCKAPI** option is intended for use by TCP/IP support and provides information for debugging problems in the TCP/IP socket layer, z/OS UNIX System Services, or the TCP/IP stack. See [z/OS Communications Server: IP Diagnosis Guide](#) for more information about this problem determination methodology.

Recovery considerations

The z/OS HTTP enabler runs in the address space of the application. In addition, all the storage needed by the HTTP enabler is obtained in the application's address space. Because every application has its own programming environment, it is impossible for the HTTP enabler to predict the recovery environment required by the application.

The HTTP enabler provides a simple ESTAE recovery mechanism that provides a minimal recovery scheme for many environments from which the toolkit could be used. Programs that run under an FRR cannot avail themselves of the HTTP enabler recovery. Furthermore, the recovery provided by the HTTP enabler does not catch all errors. Therefore, it is imperative that a robust application provides its own recovery to catch any abnormal ends during toolkit execution.

When the HTTP enabler attempts to access application-provided parameters and those parameters are either inaccessible, point to an inaccessible location, or specify a length that goes beyond the available storage obtained by the application, an abend occurs. In many cases, the recovery of the toolkit catches the error and returns normally to the application with the **returnCode** parameter set to **HWTH_INACCESSIBLE_PARM** and the **reasonDesc** value in the **diagArea** set to the name of the bad parameter. There are some cases where parameter checking must occur outside of the HTTP enabler recovery environment. In these cases, the **returnCode** parameter is set to the error return code. If the toolkit abnormally ends, the application's recovery can consult the **returnCode** and **diagArea** values in the callers dynamic storage at the time of the abend and determine which parameter the toolkit could not process.

Note: Language Environment callers can see a Language Environment-specific abend code other than OC4. Under certain circumstances, the Language Environment message, CEE3501S The module *xxxxxxx* was not found, can appear in standard output, where *xxxxxxx* is the application program's Language Environment condition handler. The calling Language Environment program still receives control with the failing toolkit return code and **diagArea** information.

Redirection considerations

The HTTP enabler supports automatic redirection (URL/URI forwarding) of requests from one location to another based on the HTTP specifications, as documented in the various RFC publications.

Cross-domain and non-cross-domain redirection: Redirecting a request can be as simple as reissuing an HTTP request to a resource on the same domain but with a different path (non-cross-domain redirection), or can be more involved by establishing a new connection to another domain and then reissuing the HTTP request to that new domain location (cross-domain redirection). The application can choose to allow or disallow cross-domain redirection by using the **HWTH_OPT_XDOMAIN_REDIRECTS** set option.

To be considered as a non-cross-domain redirect, the HTTP enabler requires an exact match between an original URI's **authority** and its redirected URI's authority. For example, a redirect from `http://example.com` to `http://www.example.com` is treated as a cross-domain redirect and a successful redirection would require the **HWTH_OPT_XDOMAIN_REDIRECTS** option to have been set to **HWTH_XDOMAIN_REDIRS_ALLOWED**.

Redirection protocol change: A particular HTTP scheme (protocol) can sometimes be requested by a server when it notifies the client of the new redirect location. The application can choose what protocol changes are allowed by using the **HWTH_OPT_REDIRECT_PROTOCOLS** set option.

Number of redirect attempts: The HTTP standard allows for the (potentially unlimited) "nesting" of redirects. An application can limit the depth of redirect processing that the toolkit may attempt by setting the **HWTH_OPT_MAX_REDIRECTS** option to a value. If no value is specified, the toolkit defaults to a maximum of five attempts on a given request.

General redirect behavior—some technical details: There are a number of valid redirect status response codes that can be sent from the server back to the client. The response to these status codes is dictated, usually, by the various RFCs that deal with HTTP redirect processing. The toolkit automatically processes redirects (to unburden the application), whenever it is safe to do so. If the original request used an unsafe HTTP method (POST, PUT, DELETE, PATCH), then toolkit does not automatically process the HTTP request in order to protect the application. [Table 113 on page 586](#) describes how the toolkit handles each of the HTTP redirection status response codes (3xx), in accordance with current industry standards.

Table 113. Toolkit handling of HTTP redirection status response codes

Received HTTP status response codes	Toolkit behavior
<ul style="list-style-type: none"> • 300 Multiple Choices • 307 Temporary Redirect 	<p>The toolkit attempts to redirect the request to the location specified in the Location response header if and only if the application specifies a HWTH_OPT_REQUESTMETHOD value of HWTH_HTTP_REQUEST_GET or HWTH_HTTP_REQUEST_HEAD, provided the other redirect options set by the application allow the toolkit to redirect the request.</p> <p>For other methods or if the Location header is not specified, the request is not redirected. In this case, the response header callback routine is driven for each response header along with the status response code. The application can choose to issue a subsequent request.</p>
<ul style="list-style-type: none"> • 301 Moved Permanently • 302 Found • 303 See Other 	<p>The toolkit automatically redirects the request, provided the other options set by the application allow the toolkit to redirect the request. If the application specifies a HWTH_OPT_REQUESTMETHOD value of HWTH_HTTP_REQUEST_GET or HWTH_HTTP_REQUEST_HEAD, the redirect is identical to the initial request, except to the new target. However, if the HWTH_OPT_REQUESTMETHOD value is anything other than HWTH_HTTP_REQUEST_GET and HWTH_HTTP_REQUEST_HEAD, the toolkit follows the industry de facto client behavior and downgrades the request to HWTH_HTTP_REQUEST_GET. Under no circumstances will an unsafe HWTH_OPT_REQUESTMETHOD value of HWTH_HTTP_REQUEST_POST, HWTH_HTTP_REQUEST_PUT, or HWTH_HTTP_REQUEST_DELETE be forwarded with the request method unchanged.</p>

REXX Programming Considerations

The toolkit provides a REXX host command environment, HWTHTTP, to allow REXX applications to easily direct their requests to the HTTP enabler using an easy-to-use, made-for-REXX interface. REXX applications running in TSO/E, System REXX, z/OS UNIX, or ISV-provided REXX environments are supported.

- To initialize the HWTHTTP host command environment in your REXX exec, it may be necessary to invoke the **hwtcalls** function at the beginning of your application: `call hwtcalls on`. After this invocation, both the `ADDRESS HWTHTTP` and `ADDRESS HWTJSON` host commands will direct API calls to the toolkit.
- To declare all toolkit constants in your REXX exec, use the HWTCONST service as documented in [“HWTCONST — Initialize predefined variables \(REXX\)”](#) on page 595. There is no REXX IDF (include file) provided by the toolkit.
- The toolkit services allocate task associated resources, which are released at task termination and the termination API calls.
- Handles are not shared among multiple tasks, which can restrict some reentrant REXX environments.
- HTTP handles can be updated by any of the HTTP services. The content of these variables should not be modified in any way by the application.
- Verify that all variables have proper content and are exposed if set outside of procedures.
- Variable names specified on toolkit REXX service calls are limited to 40 characters in length.
- REXX does not have unlimited variable content size. In general, a single variable cannot contain more than 16 MB of content. This limits the amount of data that can be sent and received in the HTTP request

and response bodies. If the data required is greater than 16 MB for any of these cases, consider to use one of the high-level languages, which are supported by the toolkit (C/C++, COBOL, PL/I or Assembler).

- Programs running in any REXX environment that is also a z/OS UNIX process should code `CALL SYSCALLS 'SIGOFF'` in the REXX exec before invoking any HTTP toolkit service (HWTH* services) to turn off MVS signaling. Failure to do this in the REXX exec can result in an `HWTH_ENVIRONMENTAL_ERROR (X'F05')` return code from the service.
- The built-in REXX RC variable contains the return code from the REXX HWTHTTP host command. This return code indicates the acceptance of the supplied REXX HWTHTTP host command. The return codes returned in the RC variable are generally unique to the REXX environment. In contrast, the HTTP service return code, the variable supplied on the service call itself, is only completed if the RC variable has a value of `HWTH_OK (0)` or `HWT_REXXParmSyntaxError (1)`.
- The **DiagArea** for each REXX service call is returned by using stem variables in the form: `x.HWTH_service`, `x.HWTH_reasonCode`, and `x.HWTH_reasonDesc`, where `x` is the name of the stem variable that is specified on the parameter list. If no **DiagArea** information is completed by the toolkit, the value of the **DiagArea** stem-variable on return is blank or null.

Table 114 on page 587 lists the host command return codes for the REXX environment.

Table 114. Host return codes for REXX	
Host return code	Meaning and action
0	<p>Meaning: REXX toolkit host command successful.</p> <p>Action: Consult the toolkit return code on the service call to determine the final result of the request.</p>
1 HWT_REXXParmSyntaxError	<p>Meaning: REXX toolkit host command detects the parameter format is not in the proper form to be accepted.</p> <p>Action: Check for a probable coding error.</p> <ul style="list-style-type: none"> • See the return code on the toolkit service call to determine the reason for the syntax error. • See the REXX programming considerations of the toolkit service to see the exact calling specifications. • Compare the toolkit REXX service call attempted with service call examples in the supplied toolkit REXX programming sample found in SYS1.SAMPLIB. • The DiagArea might contain additional diagnostic information.
2 HWT_REXXUnsupportedService	<p>Meaning: Program error. An unknown toolkit service name was specified on the toolkit REXX host command.</p> <p>Action: Check for a probable coding error. Specify a valid toolkit service name. For example, <code>HWTHCONN</code>.</p>
3 HWT_REXXInvalidNumOfParms	<p>Meaning: Program error. The number of parameters specified on the toolkit REXX host command for the service name specified does not match the number of parameters expected.</p> <p>Action: Check for a probable coding error. See the REXX programming considerations of the toolkit service to see the exact calling specifications. Compare the toolkit REXX service call attempted with service call examples in the supplied toolkit REXX programming sample found in SYS1.SAMPLIB.</p>

Table 114. Host return codes for REXX (continued)

Host return code	Meaning and action
4 HWT_REXXStemVarRequired	<p>Meaning: Program error. The toolkit REXX service specified on the toolkit REXX host command is missing one or more required stem variables in the positional parameter list.</p> <p>Action: Check for a probable coding error. See the REXX programming considerations of the toolkit service to see the exact calling specifications. A stem variable parameter must specify a period (.) following the variable name (for example, var .). Also, compare the toolkit REXX service call attempted with service call examples found in the supplied toolkit REXX programming sample found in SYS1.SAMPLIB.</p>
5 HWT_REXXParmNameTooLong	<p>Meaning: Program error. One or more variables specified on the toolkit REXX service call on the toolkit REXX host command is greater than the toolkit maximum REXX variable length (40).</p> <p>Action: Check for a probable coding error. Reduce the variable name lengths on the toolkit REXX service call to be 40 characters or less in length</p>
6 HWT_REXXInvalidHostEnv	<p>Meaning: System error. The toolkit detected an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
7 HWT_REXXNoStorageForVar	<p>Meaning: System error. Insufficient storage is detected by a SET request from the REXX variable access routine (IRXEXCOM). The system rejects the service call.</p> <p>Action: Ensure that there is sufficient storage available for the toolkit to set REXX variables. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
8 HWT_REXXirxexcom1	<p>Meaning: System error. The REXX variable access routine (IRXEXCOM) used by the toolkit detected an invalid entry condition. This error can be caused by invoking the toolkit REXX host command from a non-REXX application.</p> <p>Action: Ensure to invoke the toolkit REXX host command from a valid REXX exec. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
9 HWT_REXXirxexcom28	<p>Meaning: System error. The REXX variable access routine (IRXEXCOM) detected a language processor environment is missing. This error can be caused by invoking the toolkit from an invalid REXX environment.</p> <p>Action: Ensure that REXX applications invoke the specified toolkit service in a proper REXX environment. TSO/E, System REXX, z/OS UNIX, or ISV-provided REXX environments are supported. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Table 114. Host return codes for REXX (continued)	
Host return code	Meaning and action
11 HWT_REXXNoStorage	<p>Meaning: System error. The toolkit could not obtain sufficient storage to satisfy the request.</p> <p>Action: Ensure there is sufficient memory available for REXX command processing. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
13 HWT_REXXInvalidVariable	<p>Meaning: Program error. The toolkit detected one of the variables passed in the parameter list is an invalid REXX variable name.</p> <p>Action: Check for a probable coding error. Verify that all variables passed in the parameter list for the specified service have valid names. See the REXX programming considerations and parameters sections for reference.</p>
14 HWT_REXXDataTooLongForVar	<p>Meaning: Program error. The REXX variable cannot contain more than 16 megabytes of data.</p> <p>Action: Check for a possible coding error. If the application requires more than 16 megabytes of data, consider using another supported language.</p>
32 HWT_REXXUnexpectedError	<p>Meaning: System error. An unexpected error is detected. The system rejects the service call.</p> <p>Action: A symptom record has been written to LOGREC to record the problem. Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Using the REXX APIs in a z/OS UNIX Environment

Programs running in any REXX environment that accesses z/OS UNIX from their address space must turn off MVS signaling by coding `CALL SYSCALLS "SIGOFF"` in the REXX exec before invoking any HTTP toolkit service (HWTH* services). Failure to do this in your REXX exec can result in an `HWTH_ENVIRONMENTAL_ERROR (X'F05')` return code from the service.

z/OS HTTP enabler programming examples

The z/OS HTTP enabler provides a sample program in all supported programming languages to aid in the creation of applications that use the toolkit functions. Each sample contains examples of how to use almost all of the HTTP enabler services available in the toolkit. The samples are shipped in `SYS1.SAMPLIB`. [Table 115 on page 589](#) lists the sample files for each programming language.

Table 115. z/OS HTTP enabler programming sample files	
Programming language	Name of sample in SYS1.SAMPLIB
C / C++	HWTHXC1, HWTHXC2
COBOL	HWTHXCB1, HWTHXCB2
PL / I	HWTHXPI1
REXX	HWTHXR1

AT-TLS usage overview

Using the toolkit with AT-TLS

This describes the basic understanding of what AT-TLS does and how it works. It will help you understand how the toolkit exploits its capability to transparently secure HTTPS communications.

AT-TLS explained

The AT-TLS policy agent (PAGENT) is software that resides directly in the TCPIP stack, observing every IP socket connection made using that stack. The PAGENT decides how to intervene in the processing of a given connection attempt by comparing several possible attributes of that connection attempt to a set of rules. These rules determine which policy the PAGENT will apply, if any, to that connection attempt. For a complete description of these rules, see TTLSRule of the AT-TLS policy statements section in [z/OS Communications Server: IP Configuration Reference](#).

Each of these rules can specify several attributes of a connection that it intends to map, including:

- inbound vs. outbound
- local IP address
- local IP port
- destination IP address
- destination IP port
- user identity
- job name

Note: Toolkit connections are exclusively outbound.

The address and port selectors support named groups, port ranges, and the name selectors support a trailing asterisk wildcard. Each rule also specifies an internal priority that is used when rules overlap.

Collections of these rules form a map between an arbitrary connection attempt and, at most, one AT-TLS policy defined to handle that attempt. If no AT-TLS rules match a connection attempt, then no AT-TLS policy will apply. When two or more rules match the same connection attempt, then the rule with the highest internal priority applies. If the priorities are the same on overlapping rules, then the first, alphabetical ordered rule applies.

By carefully crafting these rules, a network administrator is able to control all the network security that is required by a wide range of diverse connection types. This includes the outbound connections to HTTPS servers made by the toolkit client.

How to enable the toolkit to use AT-TLS

When you accept the default setting of **HWTH_OPT_USE_SSL (HWTH_SSL_NONE)**, your toolkit application delegates all HTTPS security responsibilities to AT-TLS. This includes both direct connections and proxy (tunnel) connections.

Note: By definition, an HTTP proxy connection is not secured by TLS. To control access to an HTTP proxy, a network administrator may utilize an authenticating proxy that accepts Basic authentication. For more information, see [“Using an authenticating proxy server” on page 582](#).

AT-TLS example scenarios for network administrators

This is a high level overview for network administrators who want to configure AT-TLS to work correctly with the toolkit. A few example scenarios are described, ranging from no security to full security.

Unsecured connections to an internal address

A network administrator may determine that some outbound connections should never be secured by AT-TLS. To accomplish this goal, they might choose not to map the connections to any policy at all, but this is risky. Some new rule, intended to map connections being made from a new and different application, could inadvertently map their connection to an incompatible policy. It is advisable to define one or more rules to AT-TLS that affirmatively map insecure connections to an AT-TLS disabled

policy. A disabled policy guarantees that AT-TLS will never secure the connection. See [“AT-TLS policy types”](#) on page 591.

Always secured direct HTTPS connections sharing a common AT-TLS policy

A network administrator may want all direct HTTPS (port 443) outbound connections, such as those made by the toolkit, to be secured transparently by AT-TLS. In that case, the administrator may define a general rule that selects outbound connections based, in part, on the destination port being 443, or other agreed-to port number, that also specifies an AT-TLS enabling policy. An enabling policy will try to transparently establish security immediately using its own keyring. See [“AT-TLS policy types”](#) on page 591.

This approach works very well when all users can share a common keyring, such as a virtual CERTAUTH or SITE keyring, without having to provide a personal certificate for mutual authentication.

Always secured direct HTTPS connections with some server specific policies

A network administrator may want to apply specific AT-TLS policies to specific HTTPS server connections. In that case, the administrator might define several rules where each rule would select a different subset of possible HTTPS connections based, in part, on both port 443 and the server specific IP address(es). Each such rule would then map its subset of possible connections to a server specific AT-TLS enabling policy. See [“AT-TLS policy types”](#) on page 591.

Note: If the selections made by some general rule overlap with the selections made by a server specific rule, the decision comes down to the internal priority of each rule; the higher priority wins.

Application secured connections

A network administrator may also determine that an outbound connection should provide the client application with the *option* for AT-TLS to secure the connection at some point after the connection is made. To do this, the administrator may choose to define a rule that maps the connections to an AT-TLS 'application-controlling' policy and then supply that policy with a keyring that is suitable to secure any expected HTTPS connections that it will map to. One use case for this type of policy is the configuration of a proxy connection that may be used to connect to an HTTPS server. If using a proxy this way, a keyring must be specified that is appropriate for all the HTTPS servers that will be contacted via that proxy.

Always use a proxy

A network administrator may want all outbound HTTPS connections, including those made via a proxy server, to be secured by AT-TLS. This is generally possible with the latest toolkit updates, but some server specific policy mapping options, presented in the previous scenarios, are not possible when a proxy is involved. This difference is due to AT-TLS only applying its rules to the proxy connection and never observing the proxy connecting elsewhere. It is strictly the proxy's responsibility to control the connection that it makes on a client's behalf.

A toolkit proxy connection is a perfect use case for an AT-TLS 'application-controlling' policy. When the toolkit connects to an HTTP proxy and that proxy connection is mapped to an AT-TLS 'application-controlling' policy, the toolkit has options for securing any HTTPS tunnel connection it might open via that proxy. The toolkit can either request that AT-TLS secure the tunnel connection, or the toolkit can secure the connection with its own options (**HWTH_OPT_SSLKEY** and others, if specified). On the other hand, if the proxy is used to communicate with an HTTP server, where any attempt to secure the connection would be a protocol error, no keyring or handshake is involved.

Network administrators who hope to control the security of HTTPS tunnels through their proxy can map the proxy connection to an AT-TLS 'application-controlling' policy and supply the policy with a keyring that is suitable for any HTTPS server that the proxy is intended to serve up to the client.

AT-TLS policy types

Whenever AT-TLS is active, its policies might affect any outbound connection that the toolkit makes. To inter-operate with AT-TLS the toolkit must be able to detect and assess these policies when they are applied. The toolkit uses a special system call (**ioctl()**) to detect whether an AT-TLS policy was applied to a connection. The toolkit uses this query on every direct connection it makes, looking for the possible presence of an AT-TLS policy.

Table 116. AT-TLS policy types

Policy status	Meaning	Note
OFF	AT-TLS is not active on this connection's TCPIP stack.	“1” on page 592
NO_POLICY	AT-TLS is active, but did not map this connection to a policy.	“1” on page 592
DISABLED	AT-TLS mapped this connection to a policy incapable of securing it.	“1” on page 592
ENABLED	AT-TLS mapped this connection to a policy that already secured it.	“2” on page 592
APPLCNTRL	AT-TLS mapped this connection to a policy that did not yet secure it, but that should be capable of securing it when asked.	“3” on page 592

Table notes:

1. Any HTTPS security for the connection must be provided by the toolkit using **HWTH_SSL_USE** and toolkit SSL options (**HWTH_OPT_SSLKEY** and others). No transparent AT-TLS security is possible with this policy status and any connections that require AT-TLS security will necessarily fail with a communications error. If the user is connecting to an HTTP server, no security is required.
2. HTTPS security for this connection has already been established by AT-TLS and the details of the security have been written to the verbose trace output, if that option was selected. If the toolkit user specified **HWTH_OPT_USE_SSL** as **HWTH_SSL_USE**, then the connection will be closed and the security conflict will be reported as a communications error.
3. No security has been established for this connection. If the user is connecting to an HTTPS server, the means of securing the HTTPS connection depends on the value of the **HWTH_OPT_USE_SSL** toolkit option. A setting of **HWTH_SSL_NONE** instructs the toolkit to request security from AT-TLS. A setting of **HWTH_SSL_USE** instructs the toolkit to use its own options (**HWTH_OPT_SSLKEY** and others) to secure the connection. If the user is connecting to an HTTP server, no security is required.

How to tell if an AT-TLS policy was applied

If verbose tracing is enabled (see **HWTH_OPT_VERBOSE** of “Options for connections” on page 643), you can see whether or not the connection was mapped to an AT-TLS policy. If the connection was mapped, the trace includes the name of the TTLSRule that selected the connection and the type of policy the connection maps to. Also, if the policy is ENABLED, meaning the connection is already secure, the trace will include additional details of the current security of the connection, including TLS level, negotiated cipherspec, and FIPS140 status.

Important: AT-TLS policy configuration errors may first appear to be toolkit problems. If AT-TLS maps a toolkit connection to a faulty or inadequate AT-TLS policy, (especially a faulty ENABLED policy), the query operation intended to detect AT-TLS may fail.

In these cases, the toolkit will be able to report the apparent symptom, including the **errno** and **errno2** results from the system call, directly in the diagnostic area's reason description and in the verbose trace, if enabled. The toolkit will not be able to report the name of the offending AT-TLS TTLSRule. The **errno2** value should be decoded with the **bpxmtext** utility to get a more detailed description of the failure.

In any cases where AT-TLS behavior is implicated in a problem, or suspected, it is highly advisable to employ the sophisticated diagnostic and trace capabilities built directly into TCPIP and AT-TLS to aid with the problem analysis.

Server identity

Verify the server identity in the case of a secure connection.

In the case of a secure connection (HTTPS, for example, HTTP over TLS), the toolkit will verify that the URI provided for the target host matches the Server Identity that was presented in the server's certificate. This verification is always done in the case of a secure connection, regardless of whether the connection is being secured by AT-TLS or System SSL (HWTH_OPT_USE_SSL = HWTH_SSL_USE). This is in compliance with Sections 4.3.4 ([https Certificate Verification](#)) and 4.3.5 (IP-ID Reference Identity) of RFC 9110, and Section 3.1 (Server Identity) of RFC 2818.

All supported URI formats will be verified:

- DNS hostname - www.example.com
- IP address - IPv4, 192.0.2.0
- IP address - IPv6, [2001:db8:3333:4444::8888]

This verification also applies to any permitted cross-domain redirects during an HWTHRQST API invocation.

If a server supplies a legacy certificate which does not contain a subjectAltName extension, or the certificate does not accurately reflect the expected server's identity, either the server certificate will need to be re-issued with the update, or the application may be modified to take advantage of the HWTH_OPT_CERT_CHECK option. For legacy certificates, HWTH_CERT_CHECK_SAN_CN_DNS may be used to allow CN checking for DNS names; otherwise HWTH_CERT_CHECK_WARN may be used to convert the ERROR return code to a WARNING return code. However, even in the latter case, the application will need to be updated to accept an RC=4 (WARNING) with corresponding DiagArea reason code value in the range of '19'x to '1B'x from both HWTHCONN and HWTHRQST.

Sample HWTHCONN RC and DiagArea content due to this type of failure:

- RC = 262 (106 hex)
- DiagArea.HWTH_service = 380004x
- DiagArea.HWTH_reasonCode = `19`x
- DiagArea.HWTH_reasonDesc = "checkServerCert: Certificate not valid for DNS name"

HWTH_OPT_CERT_CHECK may be reset throughout the usage of a connection. When the option is set following an HWTHCONN, it will only apply to future cross-domain redirects made from the connection. It will not apply retroactively to the existing connection, so requests made will continue to be affected by the original setting when the connection was made. For example:

- An application may set HWTH_OPT_CERT_CHECK to HWTH_CERT_CHECK_SAN_CN_DNS when initially connecting to a legacy server to allow the connection and any subsequent requests to succeed with RC=0 for that server, and then change it to HWTH_CERT_CHECK_SAN_ONLY to require any cross-domain redirects made to other legacy servers to fail.
- An application may set HWTH_OPT_CERT_CHECK to HWTH_CERT_CHECK_WARN to allow HWTHCONN to convert a failure for that server to a warning, and then change it to HWTH_CERT_CHECK_SAN_ONLY to require cross-domain redirects made to other servers to fail if the server identity does not match. In this case, HWTHRQST will continue to return RC=4 for requests made on that connection, except when a cross-domain redirect fails to match its server identity and HWTHRQST fails with RC=262/'106'x.

Verification rules

The following table explains the rules for successful verification:

Table 117. Verification rules

Verification rules

RFC 9110 verification rules:

For host names:

- dnsName entries that are found within the subjectAltName extension of the certificate must be used for verification against the target identity
- the use of a wildcard character '*' is allowed within the certificate - see [DNS name wildcard rules](#)
- matching is case-insensitive, for example, WWW.Example.Com would match www.example.com

For IPv4 and IPv6 addresses:

- ipAddress entries that are found within the subjectAltName extension of the certificate must be used for verification against the target identity

For more information regarding DNS name verification, see [Section 6.4 of RFC 6125](#).

RFC 2818 and RFC 6125 verification rules:

For host names:

- dnsName entries that are found within the subjectAltName extension of the certificate must be used for verification against the target identity
- if no dnsName entries are found in the subjectAltName or the subjectAltName extension does not exist, the CN (Common Name) field in the Subject field of the certificate must be used
- the use of a wildcard character '*' is allowed within the certificate - see [DNS name wildcard rules](#)
- multiple CN fields within the Subject field of the certificate are not allowed for verification purposes
- matching is case-insensitive, for example, WWW.Example.Com would match www.example.com

For IPv4 and IPv6 addresses:

- ipAddress entries that are found within the subjectAltName extension of the certificate must be used for verification against the target identity

DNS name wildcard rules:

- The wildcard character may only match contents of a single label, for example, *.example.com may match a.example.com but not a.b.example.com
- The wildcard character may only match contents of the left-most label, so *.a.example.com is allowed, but a.*.example.com is not
- The label containing the wildcard must be followed by at least two labels, so *.example.com is allowed, but *.com is not; however, example.com is allowed because no wildcard is used
- The wildcard character may match part or none of a label:
 - b*.example.com matches b.example.com and bar.example.com but not abc.example.com
 - *oo.example.com matches oo.example.com and foo.example.com but not book.example.com
 - b*z.example.com matches bz.example.com and biz.example.com but not bar.example.com or fuzz.example.com
- Only one wildcard character is allowed within the label, so a*c.example.com is allowed but *b*.example.com is not

z/OS HTTP/HTTPS callable services

The z/OS HTTP enabler callable services are grouped under the following categories.

Initialization, reset, and termination services

Initialization, reset, and termination services deal with the creation and termination of HTTP enabler connection instances and request instances.

The HTTP enabler callable services in this category are:

- [“HWTHINIT — Initialize an HTTP connection or request” on page 609](#)
- [“HWTHRSET — Reset an HTTP connection or request” on page 619](#)
- [“HWTHTERM — Terminate an HTTP connection or request” on page 638](#)

Set options service

The set options service sets the options that are needed for a connection instance or request instance. The options are set one at a time. Thus, an application may call this service multiple times to set the connection options and to set the request options.

The HTTP enabler callable service in this category is:

- [“HWTHTSET — Set HTTP connection or request options” on page 624](#)

Connect and disconnect services

The connect service attempts to connect to an HTTP server using all of the attributes associated with a connection handle, as previously set by the set options service. The disconnect service attempts to disconnect a connection previously created by the connect service.

The HTTP enabler callable services in this category are:

- [“HWTHTCONN — Connect to an HTTP server” on page 596](#)
- [“HWTHTDISC — Disconnect from an HTTP server” on page 602](#)

Send request service

The send request service sends an HTTP request to an HTTP server using a connection that was created by the connect service, and processes the response from the server.

The HTTP enabler callable service in this category is:

- [“HWTHTRQST — Send a request to an HTTP server” on page 613](#)

Set link list service

The set link list service creates, appends, or frees a linked list, which is used to allow certain HTTP enabler option values to be represented by more than one data item.

The HTTP enabler callable service in this category is:

- [“HWTHTSLST — Linked list append service” on page 631](#)

HWTCONST — Initialize predefined variables (REXX)

Call the HWTCONST service to initialize predefined variables in the current REXX variable pool.

Description

This service sets the variables with names prefixed for HWTH corresponding to the interface definition for the HTTP toolkit. This service is helpful when using symbolic names in checking for specific return codes or when specifying constant values in the application. The variable **HWT_CONSTANTS** is set to a list of the interface variable names, which is useful on a procedure expose statement to make the variables visible to a procedure.

Note: This service also sets the variables for the z/OS JSON parser (HWTJ-prefixed) as well. If the REXX application utilizes both the HTTP and JSON parser portions of the toolkit, it is only necessary to call HWTHCONST once to initialize all the variables.

Syntax

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

REXX parameters
<pre>address hwthttp "hwtconst", "ReturnCode", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Character string

Contains the return code from the service.

DiagArea.

Returned parameter.

- **Type:** Stem variable

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in the [Table 118 on page 596](#).

Table 118. Return codes for the HWTHCONST service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTH_OK	0 HWTH_OK	Meaning: Successful completion. Action: None.

HWTHCONN – Connect to an HTTP server

Call the HWTHCONN service to connect to an HTTP server.

Description

The **HWTHCONN** service attempts to connect to an HTTP server using all of the attributes, which are associated with the supplied connection handle, as previously set by one or more calls to the **HWTHSET** (set options) service.

In the case of a secure connection (HTTPS, for example, HTTP over TLS), the HWTHCONN service will verify that the URI provided for the target host matches the server identity that was presented in the server's certificate. If the verification is unable to find the server's identity within the provided server certificate, HWTHCONN will return an ERROR (RC=262/'106'x). The application may choose to take advantage of the HWTH_OPT_CERT_CHECK connection option to tune the checking performed and the return code severity in response to a verification failure. The DiagArea parameter will contain additional

information when an ERROR or WARNING is returned. For more information about what verification is performed, see [“Server identity”](#) on page 593.

This verification is always performed by default for a secure connection, regardless of whether or not the connection is being secured by AT-TLS or System SSL (**HWTH_OPT_USE_SSL = HWTH_SSL_USE**). If the verification fails, **HWTHCONN** will return an ERROR (RC=262/'106'x). The application can choose to take advantage of the **HWTH_OPT_CERT_CHECK** option to indicate a WARNING (RC=4) should be returned instead of the ERROR when verification fails. The DiagArea parameter will contain additional information when an ERROR or WARNING is returned.

If the connection is successful (RC=0 or RC=4), this connection is eligible to issue HTTP/HTTPS requests using the **HWTHRQST** service.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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 577 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHCONN service

All information for the HWTHCONN service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
CALL HWTHCONN(ReturnCode, ConnectionHandle, DiagArea);	address hwthttp "hwthconn", "ReturnCode", "ConnectionHandle", "DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter:

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ConnectionHandle

Supplied parameter:

- **Type:** Character string
- **Length:** 12 bytes

A value that was previously returned by an HWTHINIT call that specified a **handleType** of HWTH_HANDLETYPE_CONNECTION. The connection associated with this handle should have set the minimum number of connection options using the HWTHSET service before invoking the HWTHCONN service.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that can contain additional diagnostic information which is related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call can result in a X'04D' ABEND with a reason code of X'1001yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 119 on page 598](#).

Table 119. Return codes for the HWTHCONN service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTH_OK	0 HWTH_OK	Meaning: Successful completion. Action: None.
4 HWTH_WARNING	4 HWTH_WARNING	Meaning: Possible error. The connect request was processed successfully, but detected a condition that should be reported back to the application. Action: Consult the DiagArea for a detailed explanation of this return code. Modify the application, as necessary.

Table 119. Return codes for the HWTHCONN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
101 HWTH_HANDLE_INV	257 HWTH_HANDLE_INV	<p>Meaning: Program error. The value of the connectionHandle parameter that was specified on the service call is not a valid connect or request handle (one that was returned by the HWTHINIT service).</p> <p>Action: Check the calling program for a probable coding error.</p>
102 HWTH_HANDLE_INUSE	258 HWTH_HANDLE_INUSE	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. A previous caller using this handle that is abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one is allowed access. Change the application so that only one thread attempts to use the same handle at the same time. If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
103 HWTH_HANDLETYPE_INVALID	259 HWTH_HANDLETYPE_INVALID	<p>Meaning: Program error. The application specified a request handle for the connectionHandle parameter.</p> <p>Action: Check the calling program for a probably coding error. Specify a valid connection handle for the connectionHandle parameter.</p>

Table 119. Return codes for the HWTHCONN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
104 HWTH_INACCESSIBLE_PARM	260 HWTH_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service calls abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about actions to consider for this return code.</p>
106 HWTH_COMMUNICATION_ERROR	262 HWTH_COMMUNICATION_ERROR	<p>Meaning: A communication error has been detected. One or more of the following problems has occurred:</p> <ul style="list-style-type: none"> • A failure in the communication with the web server • An error in an underlying sockets or SSL/TLS service call • An error in obtaining the necessary system resources to process the connect. <p>Action: Check the diagArea for further diagnostic information. The toolkit uses many internal services, including sockets, SSL, and other calls when processing an HTTP API service call. If one of these internal services fails because of an error in communications with the targeted server or because of an internal environmental condition, the error is reported in the diagnostic area. The diagnostic area is often populated with z/OS Unix return codes (errno) or System SSL function return codes and their textual descriptions. This information can be useful to the application programmer but, in many cases, it is for the use of IBM Support. If one of these errors occurs, clean up the environment, check for possible communication configuration problems, and reissue the request. If the problem persists, contact the IBM Support Center.</p>

Table 119. Return codes for the HWTHCONN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
501 HWTH_HCONN_CONNECT_INV	1281 HWTH_HCONN_CONNECT_INV	<p>Meaning: Program error. One of the following errors occurred:</p> <ul style="list-style-type: none"> The caller did not specify the required minimum parameters before connecting. The caller specified incompatible or incomplete connection parameters. <p>Action: Check the calling program for a probable coding error. The diagArea should contain a detailed message explaining the problem.</p>
F01 HWTH_INTERRUPT_STATUS_INV	3841 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTH_LOCKS_HELD	3842 HWTH_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTH_MODE_INV	3843 HWTH_MODE_INV	<p>Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWTH_AUTHLEVEL_INV	3844 HWTH_AUTHLEVEL_INV	<p>Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services, which do not permit key 0 callers. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 119. Return codes for the HWTHCONN service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F05 HWTH_ENVIRONMENTAL_ERROR	3845 HWTH_ENVIRONMENTAL_ERROR	<p>Meaning: Language Environment could not create the proper environment for the request. This could occur for a number of reasons, the most likely of which are:</p> <ul style="list-style-type: none"> • The POSIX (ON) runtime option was not set (Language Environment callers). • A POSIX (ON) environment was already established in the same address space, possibly because an HTTP connection was already established (non-Language Environment callers). If the dubbing default is not set to DUBPROCESS, the limit is one POSIX (ON) environment per address space. If the dubbing default is set to DUBPROCESS, each thread in the address space can have its own POSIX (ON) environment, allowing for multiple connections. See “Environmental considerations” on page 579 for more information. <p>Action:</p> <ul style="list-style-type: none"> • For Language Environment callers, verify that the POSIX (ON) runtime option has been enabled for the application. • For non-Language Environment callers, verify the dubbing options selected for the address space and ensure that multiple POSIX (ON) runtime environments are not being requested. See “Environmental considerations” on page 579 for more information about how to enable this functionality.
F06 HWTH_UNSUPPORTED_RELEASE	3846 HWTH_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.</p>
FFF HWTH_UNEXPECTED_ERROR	4095 HWTH_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTHDISC – Disconnect from an HTTP server

Call the HWTHDISC service to disconnect from an HTTP server.

Description

The HWTHDISC service attempts to disconnect a connection created by the HWTHCONN service.

If the operation is successful, the connection is disconnected from the web server (all socket and SSL/TLS connections will be terminated), but all attributes associated with the connection handle remain intact.

This allows subsequent HWTHCONN service calls to make minimal or no changes to the options for this connection, should a similar connection be desired in the future.

If the specified connection handle is not currently connected to a web server or has already been disconnected, the disconnect request ends with a successful (HWTH_OK) return code.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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 577](#) for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHDISC service

All information for the HWTHDISC service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
CALL HWTHDISC(ReturnCode, ConnectionHandle, DiagArea);	address hwthttp "hwthdisc", "ReturnCode", "ConnectionHandle", "DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ConnectionHandle

Supplied parameter

- **Type:** Character string
- **Length:** 12 bytes

A value that was previously returned by an HWTHINIT call that specified a **handleType** of HWTH_HANDLETYPE_CONNECTION.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that may contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1002yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 120 on page 604](#).

Table 120. Return codes for the HWTHDISC service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTH_OK	0 HWTH_OK	Meaning: Successful completion. Action: None.
4 HWTH_WARNING	4 HWTH_WARNING	Meaning: Possible error. The connect request was processed successfully, but detected a condition that should be reported back to the application. Action: Consult the DiagArea for a detailed explanation of this return code. Modify the application, as necessary.
101 HWTH_HANDLE_INV	257 HWTH_HANDLE_INV	Meaning: Program error. The value of the connectionHandle parameter that was specified on the service call is not a valid connect or request handle (one that was returned by the HWTHINIT service). Action: Check the calling program for a probable coding error.

Table 120. Return codes for the HWTHDISC service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTH_HANDLE_INUSE	258 HWTH_HANDLE_INUSE	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> • The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. • A previous caller using this handle abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> • While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one will be allowed access. Change the application so that only one thread attempts to use the same handle at the same time. • If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
103 HWTH_HANDLETYPE_INV ALID	259 HWTH_HANDLETYPE_INV ALID	<p>Meaning: Program error. The application specified a request handle for the connectionHandle parameter.</p> <p>Action: Check the calling program for a probably coding error. Specify a valid connection handle for the connectionHandle parameter.</p>

Table 120. Return codes for the HWTHDISC service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
104 HWTH_INACCESSIBLE_PARM	260 HWTH_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service call abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about actions to consider for this return code.</p>
106 HWTH_COMMUNICATION_ERROR	262 HWTH_COMMUNICATION_ERROR	<p>Meaning: A communication error has been detected. One or more of the following problems has occurred:</p> <ul style="list-style-type: none"> • A failure in the communication with the web server • An error in an underlying sockets or SSL/TLS service call • An error in obtaining the necessary system resources to process the disconnect <p>Action: Check the diagArea for further diagnostic information. The toolkit uses many internal services, including sockets, SSL, and other calls when processing an HTTP API service call. If one of these internal services fails because of an error in communications with the targeted server or because of an internal environmental condition, the error is reported in the diagnostic area. The diagnostic area is often populated with z/OS Unix return codes (errno) or System SSL function return codes and their textual descriptions. This information can be useful to the application programmer but, in many cases, it is for the use of IBM Support. If one of these errors occurs, clean up the environment, check for possible communication configuration problems, and reissue the request. If the problem persists, contact the IBM Support Center.</p>

Table 120. Return codes for the HWTHDISC service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
109 HWTH_CONNECTION_NO T_ACTIVE	265 HWTH_CONNECTION_NO T_ACTIVE	<p>Meaning: Program error. The HWTHDISC service cannot be issued for a connection that has not been made active by the HWTHCONN service.</p> <p>Action: Probable coding error. Only issue the HWTHDISC service call for connections that have been successfully connected using the HWTHCONN service.</p>
F01 HWTH_INTERRUPT_STAT US_INV	3841	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTH_LOCKS_HELD	3842 HWTH_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTH_MODE_INV	3843 HWTH_MODE_INV	<p>Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWTH_AUTHLEVEL_INV	3844 HWTH_AUTHLEVEL_INV	<p>Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services which do not permit key 0 callers. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 120. Return codes for the HWTHDISC service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F05 HWTH_ENVIRONMENTAL_ERROR	3845 HWTH_ENVIRONMENTAL_ERROR	<p>Meaning: Language Environment could not create the proper environment for the request. This could occur for a number of reasons, the most likely of which are:</p> <ul style="list-style-type: none"> • The POSIX(ON) runtime option was not set (Language Environment callers). • A POSIX(ON) environment was already established in the same address space, possibly because an HTTP connection was already established (non-Language Environment callers). If the dubbing default is not set to DUBPROCESS, the limit is one POSIX(ON) environment per address space. If the dubbing default is set to DUBPROCESS, each thread in the address space can have its own POSIX(ON) environment, allowing for multiple connections. See “Environmental considerations” on page 579 for more information. <p>Action:</p> <ul style="list-style-type: none"> • For Language Environment callers, verify that the POSIX(ON) runtime option has been enabled for the application. • For non-Language Environment callers, verify the dubbing options selected for the address space and ensure that multiple POSIX(ON) runtime environments are not being requested. See “Environmental considerations” on page 579 for more information about how to enable this functionality.
F06 HWTH_UNSUPPORTED_RELEASE	3846 HWTH_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.</p>
FFF HWTH_UNEXPECTED_ERROR	4095 HWTH_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTHINIT — Initialize an HTTP connection or request

Call the HWTHINIT service to initialize a connection with a web server or prepare to issue an HTTP request.

Description

The HWTHINIT service prepares to establish a connection with a remote web server or to prepare to issue an HTTP request. This service must be invoked before any other z/OS HTTP/HTTPS enabler service in the toolkit. The service prepares the memory space in the callers address space as required by either a z/OS HTTP enabler connection or a z/OS HTTP enabler request. Based on the specified **handleType**, the service returns either a connection handle or a request handle. If initializing a connection, a connection handle is passed back to the application, which can be used on subsequent services that reference this connection. Likewise, if initializing a request, a request handle is passed back to the application, which can be used on subsequent services that reference this request.

Multiple connections and requests can be established for a single address space. See "z/OS UNIX limit of processes with a POSIX(ON) environment and its effect on concurrent connections" in ["Environmental considerations" on page 579](#) for a discussion about the limitations on concurrent connections within a single address space.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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 577](#) for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHINIT service

All information for the HWTHINIT service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTHINIT(ReturnCode, HandleType, ConnOrReqHandle, DiagArea);</pre>	<pre>address hwthttp "hwthinit", "ReturnCode", "HandleType", "ConnOrReqHandle", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

HandleType

Supplied parameter

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies the type of handle to be initialized, from one of the following values:

HWTH_HANDLETYPE_CONNECTION

Initialize a connection to be used to connect to an HTTP server.

HWTH_HANDLETYPE_HTTPREQUEST

Initialize a request to be used to send HTTP requests to a web server.

ConnOrReqHandle

Returned parameter.

- **Type:** Character string
- **Length:** 12 bytes

Specifies a value generated by the toolkit representing a handle to be used on all subsequent HTTP enabler services for this connection or request instance. This instance contains all of the data structures and storage areas needed for the HTTP enabler services to run efficiently.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area, which is provided by the caller might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1003yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 121 on page 611](#).

Table 121. Return codes for the HWTHINIT service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTH_OK	0 HWTH_OK	Meaning: Successful completion. Action: None.
103 HWTH_HANDLETYPE_INV	259 HWTH_HANDLETYPE_INV	Meaning: Program error. The application specified an invalid value for the handleType parameter. Action: Check the calling program for a probable coding error. The caller should change the handleType to one of the valid values. See the IBM-supplied include files for the possible constant values that you can supply for this parameter.
104 HWTH_INACCESSIBLE_PARM	260 HWTH_INACCESSIBLE_PARM	Meaning: Program error. The application passed an input or output parameter, which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about z/OS HTTP enabler recovery processing. Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service calls abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about actions to consider for this return code.
105 HWTH_CANNOT_OBTAIN_WORKAREA	261 HWTH_CANNOT_OBTAIN_WORKAREA	Meaning: System error. The STORAGE OBTAIN service might not obtain the work area storage needed by the z/OS HTTP enabler during the HWTHINIT service call. Action: Consult the diagArea for the return code from the STORAGE OBTAIN service and additional information found in the HWTH_ReasonDesc section. Ensure there is sufficient memory available for the toolkit to obtain the amount needed for the work area. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Table 121. Return codes for the HWTHINIT service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F01 HWTH_INTERRUPT_STATUS_INV	3841 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTH_LOCKS_HELD	3842 HWTH_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTH_MODE_INV	3843 HWTH_MODE_INV	<p>Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWTH_AUTHLEVEL_INV	3844 HWTH_AUTHLEVEL_INV	<p>Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services, which do not permit key 0 callers. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F05 HWTH_ENVIRONMENTAL_ERROR	3845 HWTH_ENVIRONMENTAL_ERROR	<p>Meaning: Program error. The calling program is associated with a user ID that does not have an OMVS segment defined, or the proper z/OS UNIX environment is not available to this program. The system rejects the service request.</p> <p>Action: Ensure that the user ID under which this program is executing has an OMVS segment defined and that z/OS UNIX has been initialized.</p>
F06 HWTH_UNSUPPORTED_RELEASE	3846 HWTH_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.</p>
FFF HWTH_UNEXPECTED_ERROR	4095 HWTH_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTHRQST – Send a request to an HTTP server

Call the HWTHRQST service to send a request to an HTTP server.

Description

The **HWTHRQST** service sends an HTTP request represented by a request handle using a connection represented by the connection handle. When the server sends the response, the service processes the response and invoke the appropriate response callback (exit) routines, if specified. (See [“Receiving data from a server \(non-REXX\)”](#) on page 661 for more information.)

Upon completion of the service, the text in the returned **diagArea** parameter contains the HTTP status value of the request or other information about the result of the request.

If the **HWTHRQST** request results in a cross-domain redirect to a secure connection (HTTPS), the **HWTHRQST** service will verify that the URI for the target of the redirect matches the server identity that was presented in the server's certificate. If the verification is unable to find the server's identity within the provided server certificate, **HWTHRQST** will return an ERROR (RC=262/'106'x). The application may choose to take advantage of the **HWTH_OPT_CERT_CHECK** connection option to tune the checking performed and the return code severity in response to a verification failure. The **diagArea** parameter will contain additional information when an ERROR or WARNING is returned. For more information about what verification is performed, see [“Server identity”](#) on page 593.

Once **HWTHCONN** and/or **HWTHRQST** has received a WARNING, this WARNING will continue to be returned for all subsequent **HWTHRQST** requests. The WARNING indicates that one or more server identity errors were already encountered during an initial **HWTHCONN** or preceding **HWTHRQST** even in the case where the subsequent **HWTHRQST** did not experience the error.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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 577 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHRQST service

All information for the HWTHRQST service applies for REXX requests except:

- The **StatusCode** and **ReasonCode** parameters are returned for REXX callers. These parameter descriptions are listed below for further explanation.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTHRQST (ReturnCode, ConnectionHandle, RequestHandle, DiagArea);</pre>	<pre>address hwthttp "hwthrqst", "ReturnCode", "ConnectionHandle", "RequestHandle", "StatusCode", "ReasonCode", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ConnectionHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes

A handle value that was previously returned by an HWTHINIT call that specified a **handleType** of HWTH_HANDLETYPE_CONNECTION.

RequestHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes

A handle value that was previously returned by an HWTHINIT call that specified a **handleType** of HWTH_HANDLETYPE_HTTPREQUEST.

StatusCode (REXX)

Returned parameter.

- **Type:** Character representation of an integer.

The name of a REXX variable that is set to the HTTP status code.

Note: In the case when a HWTH_WARNING return code is returned and the DiagArea.HWTH_ReasonCode field indicates that it is a redirect, this StatusCode may contain irrelevant information.

ReasonCode (REXX)

Returned parameter.

- **Type:** Character string

The name of a REXX variable that is set to the HTTP reason code.

Note: In the case when a HWTH_WARNING return code is returned and the DiagArea.HWTH_ReasonCode field indicates that it is a redirect, this ReasonCode might contain irrelevant information.

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area which is provided by the caller might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1004yyyy' for one of the following reasons:

yyyy**Reason****0000**

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 122 on page 615](#).

Table 122. Return codes for the HWTHRQST service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTH_OK	0 HWTH_OK	Meaning: Successful completion. Action: None.
4 HWTH_WARNING	4 HWTH_WARNING	Meaning: Possible error. The send request successfully received a response from the server, but detected a condition that should be reported back to the application. For instance, HWTHRQST returns this return code if the toolkit followed one or more redirects, or if the response header callback routine aborted further processing. Action: Consult the diagArea for a detailed explanation of this return code. Modify the application, as necessary.
101 HWTH_HANDLE_INV	257 HWTH_HANDLE_INV	Meaning: Program error. Either the connectionHandle or the requestHandle parameter that was specified on the service call is not a valid connect or request handle (one that was returned by the HWTHINIT service). Action: Check the calling program for a probable coding error.

Table 122. Return codes for the HWTHRQST service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTH_HANDLE_INUSE	258 HWTH_HANDLE_INUSE	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. A previous caller using this handle that is abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one is allowed access. Change the application so that only one thread attempts to use the same handle at the same time. If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
103 HWTH_HANDLETYPE_INV	259 HWTH_HANDLETYPE_INV	<p>Meaning: Program error. The application either specified a request handle for the connectionHandle parameter, or it specified a connection handle for the requestHandle parameter.</p> <p>Action: Check the calling program for a probable coding error. The diagArea indicates which handle parameter has the mismatch.</p>

Table 122. Return codes for the HWTHRQST service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
104 HWTH_INACCESSIBLE_PARM	260 HWTH_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter, which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service calls abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about actions to consider for this return code.</p>
106 HWTH_COMMUNICATION_ERROR	262 HWTH_COMMUNICATION_ERROR	<p>Meaning: A communication error has been detected. One or more of the following problems has occurred:</p> <ul style="list-style-type: none"> • A failure in the communication with the web server • An error in an underlying sockets or SSL/TLS service call • An error processing the HTTP request or the response coming back from the web server • An error in the translation of the data into the proper code page • An error in obtaining the necessary system resources to process the request. <p>Action: Check the diagArea for further diagnostic information. The toolkit uses many internal services, including sockets, SSL, and other calls when processing an HTTP API service call. If one of these internal services fails because of an error in communications with the targeted server or because of an internal environmental condition, the error is reported in the diagnostic area. The diagnostic area is often populated with z/OS Unix return codes (errno) or System SSL function return codes and their textual descriptions. This information can be useful to the application programmer but, in many cases, it is for the use of IBM Support. If one of these errors occurs, clean up the environment, check for possible communication configuration problems, and reissue the request. If the problem persists, contact the IBM Support Center.</p>

Table 122. Return codes for the HWTHRQST service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
109 HWTH_CONNECTION_NOT_ACTIVE	265 HWTH_CONNECTION_NOT_ACTIVE	<p>Meaning: Program error. The HWTHRQST service cannot be issued for a connection that has not been made active by the HWTHCONN service.</p> <p>Action: Probable coding error. Issue the HWTHCONN service to activate the connection specified on the HWTHRQST service call prior to the actual HWTHRQST call.</p>
601 HWTH_HRQST_REQUEST_INV	1537 HWTH_HRQST_REQUEST_INV	<p>Meaning: Program error. One of the following errors occurred:</p> <ul style="list-style-type: none"> The caller did not specify the required minimum parameters before issuing the send request. The caller specified incompatible or incomplete request parameters. <p>Action: Check the calling program for a probable coding error. The diagArea should contain a detailed message explaining the problem.</p>
F01 HWTH_INTERRUPT_STATUS_INV	3841 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTH_LOCKS_HELD	3842 HWTH_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTH_MODE_INV	3843 HWTH_MODE_INV	<p>Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWTH_AUTHLEVEL_INV	3844 HWTH_AUTHLEVEL_INV	<p>Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services which do not permit key 0 callers. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 122. Return codes for the HWTHRQST service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F05 HWTH_ENVIRONMENTAL_ERROR	3845 HWTH_ENVIRONMENTAL_ERROR	<p>Meaning: Language Environment could not create the proper environment for the request. This could occur for a number of reasons, the most likely of which are:</p> <ul style="list-style-type: none"> • The POSIX (ON) runtime option was not set (Language Environment callers). • A POSIX (ON) environment was already established in the same address space, possibly because an HTTP connection was already established (non-Language Environment callers). If the dubbing default is not set to DUBPROCESS, the limit is one POSIX (ON) environment per address space. If the dubbing default is set to DUBPROCESS, each thread in the address space can have its own POSIX (ON) environment, allowing for multiple connections. See “Environmental considerations” on page 579 for more information. <p>Action:</p> <ul style="list-style-type: none"> • For Language Environment callers, verify that the POSIX (ON) runtime option has been enabled for the application. • For non-Language Environment callers, verify the dubbing options selected for the address space and ensure that multiple POSIX (ON) runtime environments are not being requested. See “Environmental considerations” on page 579 for more information about how to enable this functionality.
F06 HWTH_UNSUPPORTED_RELEASE	3846 HWTH_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.</p>
FFF HWTH_UNEXPECTED_ERROR	4095 HWTH_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTHRSET — Reset an HTTP connection or request

Call the HWTHRSET service to reset an HTTP connection or request.

Description

The HWTHRSET service returns a connection or a request back to the same state as when the HWTHINIT service was initially invoked.

HWTHRSET

- If you specify a connection handle, the connection is disconnected, if necessary, and all options that are previously set for this connection handle are undone. New HWTHRSET calls can now be made to set new connection options. No changes are made to any request handle.
- If you specify a request handle, all options which are previously set for this request handle are undone. New HWTHRSET calls can now be made to set new request options. No changes are made to any connection handle.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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 577 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHRSET service

All information for the HWTHRSET service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTHRSET (ReturnCode, ConnOrReqHandle, DiagArea);</pre>	<pre>address hwthttp "hwthrsset", "ReturnCode", "ConnOrReqHandle", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ConnOrReqHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes

Either a connection handle or a request handle that was previously returned by an HWTHINIT call. The REXX variable is updated by this service.

diagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1005yyyy' for one of the following reasons:

yyyy**Reason****0000**

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 123 on page 621](#).

Table 123. Return codes for the HWTHRSET service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTH_OK	0 HWTH_OK	Meaning: Successful completion. Action: None.
4 HWTH_WARNING	4 HWTH_WARNING	Meaning: Possible error. The connect request was processed successfully, but detected a condition that should be reported back to the application. Action: Consult the diagArea for a detailed explanation of this return code. Modify the application, as necessary.
101 HWTH_HANDLE_INV	257 HWTH_HANDLE_INV	Meaning: Program error. The value of the connOrReqHandle parameter that was specified on the service call is not a valid connect or request handle (one that was returned by the HWTHINIT service). Action: Check the calling program for a probable coding error.

Table 123. Return codes for the HWTHRSET service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTH_HANDLE_INUSE	258 HWTH_HANDLE_INUSE	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. A previous caller using this handle abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one will be allowed access. Change the application so that only one thread attempts to use the same handle at the same time. If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
104 HWTH_INACCESSIBLE_PARM	260 HWTH_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service call abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about actions to consider for this return code.</p>

Table 123. Return codes for the HWTHRSET service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
106 HWTH_COMMUNICATION_ERROR	262 HWTH_COMMUNICATION_ERROR	<p>Meaning: A communication error has been detected. One or more of the following problems has occurred:</p> <ul style="list-style-type: none"> • A failure in the communication with the web server • An error in an underlying sockets or SSL/TLS service call • An error processing the HTTP request or response coming back from the web server • An error in the translation of the data into the proper codepage • An error in obtaining the necessary system resources to process the disconnect <p>Action: Check the diagArea for further diagnostic information. The toolkit uses many internal services, including sockets, SSL, and other calls when processing an HTTP API service call. If one of these internal services fails because of an error in communications with the targeted server or because of an internal environmental condition, the error is reported in the diagnostic area. The diagnostic area is often populated with z/OS Unix return codes (errno) or System SSL function return codes and their textual descriptions. This information can be useful to the application programmer but, in many cases, it is for the use of IBM Support. If one of these errors occurs, clean up the environment, check for possible communication configuration problems, and reissue the request. If the problem persists, contact the IBM Support Center.</p>
108 HWTH_CANNOT_FREE_WORKAREA	264 HWTH_CANNOT_FREE_WORKAREA	<p>Meaning: System error. The STORAGE RELEASE service could not release the work area storage or part of the work area storage, as requested by the z/OS HTTP enabler.</p> <p>Action: Consult the diagArea for the STORAGE RELEASE return code and for additional information found in the HWTH_ReasonDesc section. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
F01 HWTH_INTERRUPT_STATUS_INV	3841 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 123. Return codes for the HWTHRSET service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F02 HWTH_LOCKS_HELD	3842 HWTH_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTH_MODE_INV	3843 HWTH_MODE_INV	Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request. Action: Check the calling program for a probable coding error.
F04 HWTH_AUTHLEVEL_INV	3844 HWTH_AUTHLEVEL_INV	Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services which do not permit key 0 callers. The system rejects the service request. Action: Check the calling program for a probable coding error.
F06 HWTH_UNSUPPORTED_RELEASE	3846 HWTH_UNSUPPORTED_RELEASE	Meaning: The system level does not support this service. The system rejects the service request. Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.
FFF HWTH_UNEXPECTED_ERROR	4095 HWTH_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTHSET — Set HTTP connection or request options

Call the HWTHSET service to set HTTP connection or request options.

Description

The **HWTHSET** service sets the necessary options required for a connection or a request. The options are set one at a time; a single **HWTHSET** service call sets one connection option or one request option. An application will likely call **HWTHSET** multiple times to set all the necessary connection options, and again for a request to set all the necessary request options.

Connection options for a connection handle must be set before calling the **HWTHCONN** service for this handle. Set service calls for a connection that occur after the connection has been established will generally have no effect until the connection handle has been disconnected and reconnected.

Note: The **HHWTH_OPT_COOKIE_TYPE**, **HWTH_OPT_VERBOSE**, **HWTH_OPT_VERBOSE_OUTPUT**, **HWTH_OPT_COOKIE_INPUT_BUFFER**, **HWTH_OPT_COOKIE_OUTPUT_BUFFER**, **HWTH_OPT_CERT_CHECK**, and **HWTH_OPT_CONNECT_TIMEOUT_MS** options can be set at anytime.

Request options for a request handle must be completely set before calling the **HWTHRQST** service for this handle.

Note: It is possible to reset an individual option to its original state when the connection or request handle was first created by the **HWTHINIT** service by using the **HWTHSET** service. For non-REXX callers, if both *optionValueAddr* and *optionValueLen* are set to zero, the option is reset. For REXX callers, if the *optionValue* is set to null, the option is reset.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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 577](#) for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHSET service

All information for the HWTHSET service applies for REXX requests, except:

- **OptionValue** replaces **OptionValueAddr** and **OptionValueLen**.
- The following options are not supported for REXX:
 - HWTH_OPT_RESPONSEHDR_EXIT
 - HWTH_OPT_RESPONSEBODY_EXIT
 - HWTH_OPT_STREAM_SEND_EXIT
 - HWTH_OPT_STREAM_RECEIVE_EXIT
 - HWTH_OPT_REQUESTBODY_USERDATA
- HWTH_OPT_RESPONSEHDR_USERDATA and HWTH_OPT_RESPONSEBODY_USERDATA options have a different meaning than for the other languages supported by the toolkit. See these option descriptions in the section [“HTTP/HTTPS enabler options and values” on page 643](#) for further information.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTHSET (ReturnCode, ConnOrReqHandle, Option, OptionValueAddr, OptionValueLen, DiagArea);</pre>	<pre>address hwthttp "hwthset", "ReturnCode", "ConnOrReqHandle", "Option", "OptionValue", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ConnOrReqHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Either a connection handle or a request handle that was previously returned by a call to the HWTHINIT service.

Option

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies the option to be set. See [“HTTP/HTTPS enabler options and values” on page 643](#) for a list of valid options and their descriptions. The values of the options are defined in the IBM-supplied files.

OptionValue (REXX)

Supplied parameter.

- **Type:** Character string.
- Specifies the REXX variable which contains the option value being set.
- See [“HTTP/HTTPS enabler options and values” on page 643](#) for a list of valid options and their descriptions.

OptionValueAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes
- Specifies the address of the option value to be set. Generally, the value should be in the exact form that the server would expect to receive.

Note: If the option is an address itself or is an SLST, this parameter specifies the address of an address or the address of the SLST handle. Unless otherwise noted, the toolkit copies the contents of the buffer pointed to by this address into the toolkit’s work area. The application can modify the buffer contents after the service call has completed without adversely affecting the toolkit’s set value.

OptionValueLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the data pointed to by **optionValueAddr**. If the specified option is a constant value, the **optionValueLen** must be the length (in bytes) of the constant value, as specified in the IBM-supplied file. If the specified option is an address or an SLST, the **optionValueLen** must be 4.

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1006yyyy' for one of the following reasons:

yyyy**Reason****0000**

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 124 on page 627](#).

Table 124. Return codes for the HWTHSET service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTH_OK	0 HWTH_OK	Meaning: Successful completion. Action: None.
101 HWTH_HANDLE_INV	257 HWTH_HANDLE_INV	Meaning: Program error. The value of the connOrReqHandle parameter that was specified on the service call is not a valid connect or request handle (one that was returned by the HWTHINIT service). Action: Check the calling program for a probable coding error.

Table 124. Return codes for the HWTHSET service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTH_HANDLE_INUSE	258 HWTH_HANDLE_INUSE	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> • The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. • A previous caller using this handle abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> • While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one is allowed access. Change the application so that only one thread attempts to use the same handle at the same time. • If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
104 HWTH_INACCESSIBLE_PARM	260 HWTH_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service call abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about actions to consider for this return code.</p>

Table 124. Return codes for the HWTHSET service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
107 HWTH_CANNOT_INCREASE_WORKAREA	261 HWTH_CANNOT_INCREASE_WORKAREA	<p>Meaning: System error. The STORAGE OBTAIN service could not obtain additional work area storage, as required by the z/OS HTTP enabler during the HWTHSET service call.</p> <p>Action: Consult the diagArea for the return code and additional information found in the HWTH_ReasonDesc section. Ensure that there is sufficient memory available in order for the toolkit to obtain the necessary amount of work area storage. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
108 HWTH_CANNOT_FREE_WORKAREA	262 HWTH_CANNOT_FREE_WORKAREA	<p>Meaning: System error. The STORAGE RELEASE service could not release the work area storage or part of the work area storage, as requested by the z/OS HTTP enabler.</p> <p>Action: Consult the diagArea for the STORAGE RELEASE return code and for additional information found in the HWTH_ReasonDesc section. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
201 HWTH_HSET_OPTIONVALADDR_INV	513 HWTH_HSET_OPTIONVALADDR_INV	<p>Meaning: Program error. The specified optionValueAddr was zero, but the optionValueLen was non-zero.</p> <p>Action: Check the calling program for a probable coding error.</p>
202 HWTH_HSET_OPTIONVALLEN_INV	514 HWTH_HSET_OPTIONVALLEN_INV	<p>Meaning: Program error. The specified optionValueLen was zero, but the optionValueAddr was non-zero.</p> <p>Action: Check the calling program for a probable coding error.</p>
203 HWTH_HSET_OPTION_INV	515 HWTH_HSET_OPTION_INV	<p>Meaning: Program error. The caller specified an invalid option parameter, or the option parameter did not match the handle type associated with the connOrReqHandle parameter.</p> <p>Action: Check the calling program for a probable coding error. The caller should change the option value to one of the possible valid values and verify that the option is valid for the type of handle being specified. See the IBM-supplied include files for the valid constant values that can be supplied for this parameter.</p>

Table 124. Return codes for the HWTHSET service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
204 HWTH_HSET_OPTIONVALUE_INV	516 HWTH_HSET_OPTIONVALUE_INV	<p>Meaning: Program error. The caller specified an invalid option value (the value pointed to by optionValueAddr).</p> <p>Action: Check the calling program for a probable coding error. Each option is different. Some options have constants defined in the IBM-supplied include files that contain the range of possible values. Other options are character data that have certain rules which are enforced. Still others expect an address or an SLST. The diagArea will generally provide a specific reason why the toolkit did not accept the value that was specified. See “HTTP/HTTPS enabler options and values” on page 643 for more information.</p>
205 HWTH_HSET_CONN_ALREADY_ACTIVE	517 HWTH_HSET_CONN_ALREADY_ACTIVE	<p>Meaning: Program error. The specified connect option is not allowed when the connection associated with the connect handle has already been established. Most connection options may not be set after the HWTHCONN service has successfully completed.</p> <p>Action: Check the calling program for a probable coding error. Change the program to set this particular option prior to the HWTHCONN invocation for this connection.</p>
F01 HWTH_INTERRUPT_STATUS_INV	3841 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTH_LOCKS_HELD	3842 HWTH_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTH_MODE_INV	3843 HWTH_MODE_INV	<p>Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWTH_AUTHLEVEL_INV	3844 HWTH_AUTHLEVEL_INV	<p>Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services which do not permit key 0 callers. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 124. Return codes for the HWTHSET service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F06 HWTH_UNSUPPORTED_RELEASE	3846 HWTH_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.</p>
FFF HWTH_UNEXPECTED_ERROR	4095 HWTH_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTHSLST — Linked list append service

Call the HWTHSLST service to create, append, or free a linked list of option values.

Description

The HWTHSLST service creates a linked list, appends to it, or frees an existing linked list. The linked list created by this service (called an SLST) is used to allow certain HTTP enabler toolkit option values to be represented by more than one data item. For example, the **HWTH_OPT_HTTPHEADERS** parameter takes an SLST as its input value, a set of one or more HTTP headers.

An application first creates an SLST, which creates the initial data structure and appends the first data item in a single service call. Subsequent calls to this service can append more data items to the existing SLST. The SLST can then be used as input to certain designated options. (See “HTTP/HTTPS enabler options and values” on page 643 to determine which options take an SLST as input). When the SLST is no longer needed, it can be freed.

Important:

The only supported use of an **sList** created by **HWTHSLST** is to set (HWTHSET) the **HWTH_OPT_HTTPHEADERS** option on the same **RequestHandle** that was used to create the **sList**. Sharing an **sList** among **RequestHandle** instances is not supported.

Every sList is "bound" to a RequestHandle.

When you successfully pass a zero-valued **sList** parameter to the HWTHSLST service, using the **HWTH_SLST_NEW** function, the service returns a new **sList** handle.

HWTHSLST(..., RequestHandle1, HWTH_SLST_NEW, sList1, String1, ...)

This new **sList** represents the single-element list comprised of the **String1** you passed, and is “bound” to the **RequestHandle1** in the sense that any subsequent reference to this **sList1** (to **SLST_APPEND** or **SLST_FREE**) must also provide the same “bound” **RequestHandle1** as a parameter.

By repeatedly passing a zero-valued **sList** parameter to HWTHSLST, you may choose to bind more than one **sList** handle to a **RequestHandle**, and to manipulate each **sList** independently. While this use is supported, be aware that, at most, one **sList** – the value of the **HWTH_OPT_HTTPHEADERS** option – is applicable to a call to **HWTHRQST**.

Terminating a RequestHandle frees every sList that was bound to it.

When you terminate a **RequestHandle** with **HWHTERM**, Toolkit frees all the handle’s dynamic memory allocations, including its bound **sList** storage. That means you do not need to use **HWTH_SLST_FREE** on an **sList** before terminating its bound **RequestHandle**. Furthermore, if

your code references an **sList** handle after terminating its bound **RequestHandle** the results are unpredictable and likely to be disruptive.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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 577 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHSLST service

All information for the HWTHSLST service applies for REXX requests except:

- String replaces StringAddr and StringLen.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTHSLST (ReturnCode, RequestHandle, Function, sList, StringAddr, StringLen, DiagArea);</pre>	<pre>address hwthttp "hwthslst", "ReturnCode", "RequestHandle", "Function", "sList", "String", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

RequestHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes

A handle value that was previously returned by a call to the HWTHINIT service that specified a **handleType** of HWTH_HANDLETYPE_HTTPREQUEST.

Function

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies the requested operation. Valid values are:

HWTH_SLST_NEW

Create a new linked list, create the first element, and return an **sList** handle that can be used on subsequent HWTHSLST, HWTH_SLST_APPEND, and HWTH_SLST_FREE invocations.

HWTH_SLST_APPEND

Append a new linked list element to the specified **sList**.

HWTH_SLST_FREE

Delete a linked list.

sList

Supplied and returned parameter.

- **Type:** Integer (non-REXX), variable that identifies the sList (REXX).
- **Length:** 4 bytes (non-REXX)

Specifies the handle of the linked list that is to be created, modified, or deleted.

- When **function** is non-REXX.
- When **function** is HWTH_SLST_NEW. This value must be zero upon input. The new **sList** handle is returned upon output.
- When **function** is HWTH_SLST_APPEND or HWTH_SLST_FREE, you must supply a valid **sList** handle on the service call.

StringAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes
- Specifies the address of the character string to be added to the end of the linked list chain.
 - When **function** is HWTH_SLST_NEW or HWTH_SLST_APPEND, specify a valid address.
 - When **function** is HWTH_SLST_FREE, specify zero.

StringLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the data pointed to by **stringAddr**.

- When **function** is HWTH_SLST_NEW or HWTH_SLST_APPEND, specify a valid length.
- When **function** is HWTH_SLST_FREE, specify zero.

String (REXX)

Supplied Parameter.

- **Type:** Character string

Specifies the name of a REXX variable that contains the HTTP header to be added to the **sList**. This argument is required for each function but is not used on HWTH_SLST_FREE.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1007yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 125 on page 634](#).

Table 125. Return codes for the HWTHSLST service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTH_OK	0 HWTH_OK	Meaning: Successful completion. Action: None.
101 HWTH_HANDLE_INV	257 HWTH_HANDLE_INV	Meaning: Program error. The value of the requestHandle parameter that was specified on the service call is not a valid request handle (one that was returned by the HWTHINIT service). Action: Check the calling program for a probable coding error.

Table 125. Return codes for the HWTHSLST service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTH_HANDLE_INUSE	258 HWTH_HANDLE_INUSE	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> • The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. • A previous caller using this handle abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> • While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one is allowed access. Change the application so that only one thread attempts to use the same handle at the same time. • If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
103 HWTH_HANDLETYPE_INV	259 HWTH_HANDLETYPE_INV	<p>Meaning: The application specified a connection handle for the requestHandle parameter.</p> <p>Action: Check for a probable coding error. Change the specified handle to be a request handle instead of a connection handle.</p>

Table 125. Return codes for the HWTHSLST service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
104 HWTH_INACCESSIBLE_PARM	260 HWTH_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter, which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service calls abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about actions to consider for this return code.</p>
107 HWTH_CANNOT_INCREASE_WORKAREA	261 HWTH_CANNOT_INCREASE_WORKAREA	<p>Meaning: System error. The STORAGE OBTAIN service could not obtain additional work area storage, as required by the z/OS HTTP enabler during the HWTHSLST service call.</p> <p>Action: Consult the diagArea for the return code and additional information found in the HWTH_ReasonDesc section. Ensure that there is sufficient memory available in order for the toolkit to obtain the necessary amount of work area storage. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
108 HWTH_CANNOT_FREE_WORKAREA	262 HWTH_CANNOT_FREE_WORKAREA	<p>Meaning: System error. The STORAGE RELEASE service could not release the work area storage or part of the work area storage, as requested by the z/OS HTTP enabler.</p> <p>Action: Consult the diagArea for the STORAGE RELEASE return code and for additional information found in the HWTH_ReasonDesc section. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Table 125. Return codes for the HWTHSLST service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
301 HWTH_SLST_SLIST_INV	769 HWTH_SLST_SLIST_INV	<p>Meaning: Program error. The sList parameter specified on the service call is not a valid SLST. If the function is HWTH_SLST_NEW, the supplied sList was a nonzero value. If the function is either HWTH_SLST_APPEND or HWTH_SLST_FREE, the supplied sList is not a valid SLST handle that was returned from a previous HWTHSLST call with function = HWTH_SLST_NEW.</p> <p>Action: Check the calling program for a probable coding error.</p>
302 HWTH_HSLST_FUNCTION_INV	770 HWTH_HSLST_FUNCTION_INV	<p>Meaning: Program error. The specified function is not one of the valid function types.</p> <p>Action: Check the calling program for a probable coding error. The caller should change the function value to one of the possible valid values. See the IBM-supplied include files for the valid constant values that can be supplied for this parameter.</p>
303 HWTH_HSLST_STRINGLEN_INV	771 HWTH_HSLST_STRINGLEN_INV	<p>Meaning: Program error. If function is either HWTH_SLST_NEW or HWTH_SLST_APPEND, the specified stringLen parameter was zero. If function is HWTH_SLST_FREE, the specified stringLen was nonzero.</p> <p>Action: Check the calling program for a probable coding error.</p>
304 HWTH_HSLST_STRINGADDR_INV	772 HWTH_HSLST_STRINGADDR_INV	<p>Meaning: Program error. If function is either HWTH_SLST_NEW or HWTH_SLST_APPEND, the specified stringAddr parameter was zero. If function is HWTH_SLST_FREE, the specified stringAddr was nonzero.</p> <p>Action: Check the calling program for a probable coding error.</p>
F01 HWTH_INTERRUPT_STATUS_INV	3841 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTH_LOCKS_HELD	3842 HWTH_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTH_MODE_INV	3843 HWTH_MODE_INV	<p>Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 125. Return codes for the HWTHTSLST service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F04 HWTHT_AUTHLEVEL_INV	3844 HWTHT_AUTHLEVEL_INV	Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services which do not permit key 0 callers. The system rejects the service request. Action: Check the calling program for a probable coding error.
F06 HWTHT_UNSUPPORTED_RELEASE	3846 HWTHT_UNSUPPORTED_RELEASE	Meaning: The system level does not support this service. The system rejects the service request. Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.
FFF HWTHT_UNEXPECTED_ERROR	4095 HWTHT_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTHTERM — Terminate an HTTP connection or request

Call the HWTHTERM service to terminate an HTTP connection or request.

Description

The HWTHTERM service cleans up the resources that were obtained by a previous call to the HWTHTINIT service (including the entire work area) and invalidates its handle. If you do not invoke this service, the storage allocated by the HWTHTINIT service and other HTTP enabler services remain allocated and ineligible to be used by the application, and remains allocated until the address space terminates.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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 577 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHTERM service

All information for the HWTHTERM service applies for REXX requests.

Syntax

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

Non-REXX parameters	REXX parameters
<pre>CALL HWTHTERM(ReturnCode, ConnOrReqHandle, ForceOption, DiagArea);</pre>	<pre>address hwthttp "hwtthterm", "ReturnCode", "ConnOrReqHandle", "ForceOption", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ConnOrReqHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes

Either a connection handle or a request handle that was previously returned by a call to the HWTHTERM service.

ForceOption

Supplied parameter.

- **Type:** Integer (Non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Controls the behavior of the HWTHTERM service. Sometimes a handle representing a connection or request can be stuck in an in-use state and cannot be terminated successfully. The in-use state can occur if a prior z/OS HTTP enabler service call resulted in an ABEND condition. This option allows the caller to force the connection or request instance to terminate.

The valid values are:

HWTHTERM_NOFORCE

(Recommended) Terminates the specified connection or request and invalidates its associated handle only if the connection or request is not currently in an in-use state.

HWTHTERM_FORCE

Unconditionally terminates the specified connection or request and invalidates its associated handle, regardless of the in-use status of the connection or request.



Attention: Use the HWTHTERM_FORCE option only under both the following conditions:

- No other threads in the address space are using this connection or request.
- Multiple attempts to terminate the connection or request have resulted in a return code of HWTH_HANDLE_INUSE.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1008yyyy' for one of the following reasons:

yyyy

Reason

0000

The parameters passed by the caller are not in the primary address space.

0001

The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in [Table 126 on page 640](#).

Table 126. Return codes for the HWTHTERM service		
Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
0 HWTH_OK	0 HWTH_OK	Meaning: Successful completion. Action: None.
4 HWTH_WARNING	4 HWTH_WARNING	Meaning: Possible error. The connect request was processed successfully, but detected a condition that should be reported back to the application. Action: Consult the diagArea for a detailed explanation of this return code. Modify the application, as necessary.
101 HWTH_HANDLE_INV	257 HWTH_HANDLE_INV	Meaning: Program error. The value of the connOrReqHandle parameter that was specified on the service call is not a valid connect or request handle (one that was returned by the HWTHINIT service). Action: Check the calling program for a probable coding error.

Table 126. Return codes for the HWTHTERM service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
102 HWTH_HANDLE_INUSE	258 HWTH_HANDLE_INUSE	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. A previous caller using this handle abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one is allowed access. Change the application so that only one thread attempts to use the same handle at the same time. If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
104 HWTH_INACCESSIBLE_PARM	260 HWTH_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service calls abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 577 for details about actions to consider for this return code.</p>

Table 126. Return codes for the HWTHTERM service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
108 HWTHT_CANNOT_FREE_WORKAREA	264 HWTHT_CANNOT_FREE_WORKAREA	<p>Meaning: System error. The STORAGE RELEASE service could not release the work area storage or part of the work area storage, as requested by the z/OS HTTP enabler.</p> <p>Action: Consult the diagArea for the STORAGE RELEASE return code and for additional information found in the HWTHT_ReasonDesc section. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
401 HWTHT_HTERM_FORCEOPTION_INV	1025 HWTHT_HTERM_FORCEOPTION_INV	<p>Meaning: Program error. The caller specified an invalid forceOption.</p> <p>Action: Check the calling program for a probable coding error. The caller should change the forceOption value to one of the possible valid values. See the IBM-supplied include files for the possible constant values that can be supplied for this parameter.</p>
F01 HWTHT_INTERRUPT_STATUS_INV	3841 HWTHT_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTHT_LOCKS_HELD	3842 HWTHT_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTHT_MODE_INV	3843 HWTHT_MODE_INV	<p>Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWTHT_AUTHLEVEL_INV	3844 HWTHT_AUTHLEVEL_INV	<p>Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services, which do not permit key 0 callers. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 126. Return codes for the HWTHTERM service (continued)

Hexadecimal return code Equate symbol	Decimal return code Equate symbol	Meaning and action
F05 HWTHTERM_ENVIRONMENTAL_ERROR	3845 HWTHTERM_ENVIRONMENTAL_ERROR	<p>Meaning: Language Environment could not create the proper environment for the request. This could occur for a number of reasons, the most likely of which are:</p> <ul style="list-style-type: none"> • The POSIX (ON) runtime option was not set (Language Environment callers). • A POSIX (ON) environment was already established in the same address space, possibly because an HTTP connection was already established (non-Language Environment callers). If the dubbing default is not set to DUBPROCESS, the limit is one POSIX (ON) environment per address space. If the dubbing default is set to DUBPROCESS, each thread in the address space can have its own POSIX (ON) environment, allowing for multiple connections. See “Environmental considerations” on page 579 for more information. <p>Action:</p> <ul style="list-style-type: none"> • For Language Environment callers, verify that the POSIX (ON) runtime option has been enabled for the application. • For non-Language Environment callers, verify the dubbing options selected for the address space and ensure that multiple POSIX (ON) runtime environments are not being requested. See “Environmental considerations” on page 579 for more information about how to enable this functionality.
F06 HWTHTERM_UNSUPPORTED_RELEASE	3846 HWTHTERM_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.</p>
FFF HWTHTERM_UNEXPECTED_ERROR	4095 HWTHTERM_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HTTP/HTTPS enabler options and values

The HTTP/HTTPS enabler portion of the z/OS web enablement toolkit allows many customizable options to determine how the toolkit should process a request.

Options for connections

Options can be set, one at a time, by using the HWTHTERM service. (See [“HWTHTERM — Set HTTP connection or request options” on page 624.](#))

HWTH_OPT_URI

The Uniform Resource Identifier (URI). This supplied buffer is the target location of the connection. This option is required for a connection.

Valid values are either an IPv4 or IPv6 address, or a hostname. Optionally, the hostname might be prefixed with the HTTP scheme (`http://`) or the HTTPS scheme (`https://`). Neither of the formats should be suffixed by a port specification. Instead, the application should use the **HWTH_OPT_PORT** option to specify the port. In technical terms, specify only the *authority* portion of the URI, minus the port.

```
http://192.168.0.1
http://[2001:1890:1112:1::20]
http://www.example.com
```

When the URI value is provided in a hostname format in combination with the **HWTH_OPT_USE_SSL = HWTH_SSL_USE**, an automatic SNI extension is included during the connection negotiations. To enable an SNI extension, when the connection is secured by AT-TLS, see the **ClientHandshakeSNI** parameter of [TTLSCONNECTIONADVANCEDPARMS](#) statement in *z/OS Communications Server: IP Configuration Reference*.

Note: The toolkit accepts single-byte EBCDIC character data for the value of the URI option. The use of multibyte character encodings (such as UTF-8 and UTF-16) is not supported. For more information, see [“Code page consideration”](#) on page 400.

The toolkit indirectly supports Internationalized Domain Name (IDN). To specify a hostname that contains Unicode characters, you must first apply the Punycode algorithm to convert the hostname to an ASCII representation. (The Punycode algorithm converts each Unicode character outside of the US-ASCII character set to an encoded ASCII representation). After the entire hostname string has been encoded as an ASCII sequence, it is necessary to convert to EBCDIC before passing in the **HWTH_OPT_URI** set option, since the toolkit expects all input data to be in EBCDIC.

HWTH_OPT_CERT_CHECK

An optional 4-byte integer value that tunes the level of checking performed and the return code severity in response to a verification failure when checking the server's identity within the server certificate. This option may be used to allow connections and redirects to a server that supplies a legacy certificate which does not contain a subjectAltName extension, or a certificate that does not accurately reflect the server identity.

The value for this option may be reset throughout the usage of the connection, for example, after **HWTHCONN** but before **HWTHRQST**, or between two consecutive **HWTHRQSTs**.

For more information about what verification is performed, see [“Server identity”](#) on page 593.

Valid values are:

HWTH_CERT_CHECK_SAN_ONLY

Verifies server identity in accordance with best practices defined by [RFC 9110 sections 4.3.4 and 4.3.5](#). Only applicable subjectAltName extension entries (dnsName for DNS names and ipAddress for IP addresses) are used for server identification.

If the certificate provided by the server does NOT contain values that match the provided target identity, fails **HWTHCONN** and/or **HWTHRQST** with return code value of 262 decimal/'106'x.

HWTH_CERT_CHECK_SAN_CN_DNS

Verifies server identity in accordance with best practices defined by [RFC 2818 section 3.1](#), which also permits legacy server certificates that are missing subjectAltName extension fields for DNS Names, and instead uses the Common Name (CN) field in the Subject for server identification in those cases.

If the certificate provided by the server does NOT contain values that match the provided target identity, fails **HWTHCONN** and/or **HWTHRQST** with return code value of 262 decimal/'106'x.

HWTH_CERT_CHECK_WARN

Uses rules described under **HWTH_CERT_CHECK_SAN_ONLY** for certificate verification.

If the certificate provided by the server does **not** contain values that match the provided target identity, completes **HWTHCONN** and/or **HWTHRQST** with return code value of 4. Once **HWTHCONN** and/or **HWTHRQST** have received a WARNING due to a certificate verification failure, the WARNING will continue to be returned for all subsequent **HWTHRQSTs** to reflect the fact that one or more server identity errors were already encountered during an initial **HWTHCONN** or preceding **HWTHRQST** even in the case where the subsequent **HWTHRQST** did not experience the error.

The default value is **HWTH_CERT_CHECK_SAN_ONLY**.

HWTH_OPT_VERBOSE

A 4-byte integer optionally used to turn on verbose messaging to aid in the understanding of application logic or debugging of network configuration problems. Valid values are:

HWTH_VERBOSE_OFF

The toolkit produces no additional trace messages. Analysis of application results rely on the **returnCode** and **diagArea** values from the toolkit or other tracing outside of the toolkit.

HWTH_VERBOSE_ON

The toolkit produces redacted trace messages and directs them to the standard output for the application environment, unless directed elsewhere by the **HWTH_OPT_VERBOSE_OUTPUT** option.

Values associated with the following headers will be visible:

Headers
Accept
Accept-Charset
Accept-Encoding
Accept-Language
Accept-Ranges
Access-Control-Allow-Credentials
Access-Control-Allow-Headers
Access-Control-Allow-Methods
Access-Control-Allow-Origin
Access-Control-Expose-Headers
Access-Control-Max-Age
Access-Control-Request-Headers
Access-Control-Request-Method
Age
Allow
Cache-Control
Connection
Content-Encoding
Content-Language
Content-Length
Content-Location

Headers
Content-Range
Content-Type
Date
ETag
Expect
Expires
From
Host
Last-Modified
MIME-Version
Max-Forwards
Origin
Pragma
Proxy-Authenticate
Range
Referer
Retry-After
Server
TE
Trailer
Transfer-Encoding
Upgrade
User-Agent
Vary
Via
WWW-Authenticate
Warning

HWTH_VERBOSE_UNREDACTED

The toolkit produces trace messages and directs them to the standard output for the application environment, unless directed elsewhere by the **HWTH_OPT_VERBOSE_OUTPUT** option.

Default: HWTH_VERBOSE_OFF

HWTH_OPT_SSLTRACE

A fully qualified zFS file location for System SSL trace output. When specified, the trace environment variable **GSK_TRACE** is set to 255 (0xFF) and the variable **GSK_TRACE_FILE** is set to the user specified location passed in for this option. See [Capturing trace data through environment variables in z/OS Cryptographic Services System SSL Programming](#) for further details regarding usage of % in the file name for automatic inclusion of the process id and usage of **gsktrace** to format the resulting output.

Note: This option is only applicable when **HWTH_SSL_USE** is specified for **HWTH_OPT_USE_SSL** option. See “AT-TLS usage overview” on page 590 for tracing options when AT-TLS is in effect.

HWTH_OPT_VERBOSE_OUTPUT

An optional 1- 8 character name of a valid DD (data definition) statement that specifies where trace debugging messages are to be routed. The toolkit only uses this option if the **HWTH_OPT_VERBOSE** option has been set to **HWTH_VERBOSE_ON** or **HWTH_VERBOSE_UNREDACTED**. The DD statement can specify one of the following destinations:

- A pre-allocated, traditional z/OS data set with the following recommended attributes:
 - Physical sequential (DSORG=PS)
 - Unblocked variable or undefined record format (RECFM=V or RECFM=U)
 - Unspecified (or zero-valued) block size and record length, so that the default values will be set when the DD is opened
 - Expandable (nonzero primary and secondary extents)
 - Disposition of OLD (DISP=OLD), or NEW (DISP=NEW) if allocated in a **DD** statement in the same JCL job step that includes the **EXEC** for your toolkit application
- A zFS file.

The toolkit automatically wraps the trace messages in the output data set or file when all available space has been consumed. When the wrap occurs, the toolkit clears the destination file or data set, and then writes an informational record that includes the time the wrap occurred.

For REXX only, dynamic allocation is not supported in System REXX using the TS0=NO option. If your REXX exec runs in System REXX environment and **HWTH_OPT_VERBOSE_OUTPUT** is desired, you must use the TS0=YES option.

Default: None. (Debugging messages are directed to the application's standard output.)

General options

Communication (socket) options

HWTH_OPT_PORT

An optional 4-byte integer indicating the remote port number to which to connect, instead of the default HTTP or HTTPS port.

HWTH_OPT_IPSTACK

An optional 1- 8 character local z/OS TCP/IP stack name to be used when communicating to the specified hostname.

HWTH_OPT_LOCALIPADDR

An optional outgoing IP address from which the connection is to originate. This value should be in the same form as the **HWTH_OPT_URI** value for connections.

HWTH_OPT_LOCALPORT

An optional 4-byte integer indicating the outgoing port number from which the connection is to originate.

HWTH_OPT_CONNECT_TIMEOUT_MS

An optional 4-byte integer to set a timeout value, in milliseconds, for a connection to wait while attempting to connect a socket.

Valid range: 1 - 190,000 milliseconds.

- If this option is not specified, the current CONNECTTimeout setting of the TCPCONFIG setting is in control.
- When specified, this option can only shorten the duration of a timeout; the TCPCONFIG setting remains in effect, and the shorter timeout applies to the connection attempt.

HWTH_OPT_SNDTIMEOUTVAL

An optional 4-byte integer to set a particular timeout value, in seconds, for the connection to wait to send outgoing requests over a connected socket.

HTTP enabler options

Valid range: 1 - 2,678,400 seconds.

If this option is not specified, no timeout will be applied.

HWTH_OPT_RCVTIMEOUTVAL

An optional 4-byte integer to set a particular timeout value, in seconds, for the connection to wait to receive incoming responses over a connected socket.

Valid range: 1 - 2,678,400 seconds.

If this option is not specified, no timeout is applied.

Redirect options

HWTH_OPT_MAX_REDIRECTS

An optional 4-byte integer value that specifies the maximum number of redirects to follow (on a given request). If zero, redirects are not allowed by the application.

Valid range: 0 - 50

Default: 5

HWTH_OPT_XDOMAIN_REDIRECTS

An optional 4-byte integer value that specifies the cross-domain redirect behavior. This option is only effective when **HWTH_OPT_MAX_REDIRECTS** has a value greater than zero. Valid values are:

HWTH_XDOMAIN_REDIRS_NOTALLOWED

The toolkit will attempt to follow a redirect if the redirect targets the current domain of the connection associated with the request.

HWTH_XDOMAIN_REDIRS_ALLOWED

The toolkit attempts to follow a redirect even if the domain of the redirect is different from the current domain of the connection associated with the request.

Default: **HWTH_XDOMAIN_REDIRS_NOTALLOWED**

HWTH_OPT_REDIRECT_PROTOCOLS

An optional 4-byte integer value that specifies which additional protocols are allowed in the event that a redirect is received. Unlike many other options, calls to the set service that specify this option are cumulative. Multiple protocols can be specified by calling the set service multiple times, specifying one protocol at a time. This option is only effective when **HWTH_OPT_MAX_REDIRECTS** has a value greater than zero. Valid values are:

HWTH_REDIRECT_NOPROTCHANGE

Do not allow users to change protocols during a redirect. Effectively, this clears all prior set calls for this option and removes the default **HWTH_REDIRECT_HTTPS** as a valid redirect protocol.

HWTH_REDIRECT_HTTPS

Allow redirects to use the HTTPS protocol (SSL/TLS), even when the **HWTH_OPT_USE_SSL** value is set to **HWTH_SSL_NONE**. A protocol-changing redirect requiring a secure connection can occur only if either valid configuration information (such as **HWTH_OPT_SSLKEYTYPE** and other SSL support options) has been previously set by the application, or the connection has been transparently secured by AT-TLS.

HWTH_REDIRECT_HTTP

Allow redirects to use the HTTP protocol (non-SSL/TLS), even when the **HWTH_OPT_USE_SSL** value is set to **HWTH_SSL_USE**.

Note: Use this value carefully, as an SSL/TLS session can be downgraded to HTTP if this value is selected and a redirect to an HTTP host is requested.

HWTH_REDIRECT_NOHTTPS

Allow the user to override the default behavior of the toolkit and not allow redirects to use the HTTPS protocol.

Default: **HWTH_REDIRECT_HTTPS**

SSL/TLS support options**HWTH_OPT_USE_SSL**

An optional 4-byte integer value that determines how SSL/TLS can be used to secure the connection. Valid values are:

HWTH_SSL_NONE

The application does not want to explicitly specify any SSL/TLS security configuration details to z/OS System SSL on the initial connection. This value should be selected when one of the following is true:

- The HTTP protocol is the only protocol.
- An HTTPS connection must be secured by AT-TLS.

Note: If redirects are enabled (and allowed), additional SSL/TLS configuration may be specified. In particular, a redirect to HTTPS will require either AT-TLS intervention or user supplied SSL/TLS options. For more information, see the **HWTH_OPT_MAX_REDIRECTS** option to enable redirects and the **HWTH_OPT_REDIRECT_PROTOCOLS** option to allow HTTPS.

HWTH_SSL_USE

The application wants to explicitly specify the SSL/TLS security configuration details to z/OS System SSL on the initial connection. This value should be selected only when the applications requires an HTTPS connection and there is no AT-TLS enabled policy for this connection that would upgrade the connection automatically. If this option is specified, and there is an AT-TLS enabled policy for this connection, the request will fail with an **HWTH_COMMUNICATION_ERROR**.

Note: SSL/TLS will always be used when connecting to the user-specified URI. A redirect URI could use the HTTP protocol only if redirects have been enabled (see the **HWTH_OPT_MAX_REDIRECTS** option for more information) and the **HWTH_OPT_REDIRECT_PROTOCOLS** option allows HTTP.

Default: **HWTH_SSL_NONE**

HWTH_OPT_SSLVERSION

An optional 4-byte integer value that sets one or more SSL versions to be supported by this HTTP request. Unlike many other options, calls to the set service that specify this option are cumulative. Multiple versions can be specified by calling the set service multiple times, setting one SSL version at a time. Valid values are:

HWTH_SSLVERSION_TLSv13

Support TLS version 1.3.

HWTH_SSLVERSION_TLSv12

Support TLS version 1.2.

HWTH_SSLVERSION_TLSv11

Support TLS version 1.1.

HWTH_SSLVERSION_TLSv1

Support TLS version 1.0.

HWTH_SSLVERSION_SSLv3

Support SSL version 3.0. This is not recommended.

HWTH_SSLVERSION_DEFAULT

SSL is to be used but no security versions are sent on the request. The default SSL versions as determined by z/OS is sent. This option can also be used to reset all SSL version values after disconnecting, before reusing an existing connection handle.

Default: **HWTH_SSLVERSION_DEFAULT**

HWTH_OPT_SSLKEYTYPE

An optional 4-byte integer value that specifies the type of keystore to be used for HTTPS requests. This option is required when **HWTH_OPT_USE_SSL** is set to **HWTH_SSL_USE**. Valid values are:

HWTH_SSLKEYTYPE_KEYDBFILE

Use a key database file.

HWTH_SSLKEYTYPE_KEYRINGNAME

Use a SAF key ring name or a PKCS #11 token.

HWTH_OPT_SSLKEY

An optional buffer that specifies the name of the keystore to be used. This option is required when **HWTH_OPT_USE_SSL** is set to **HWTH_SSL_USE**. The value that you specify depends on the value of **HWTH_OPT_SSLKEYTYPE**, as follows:

When HWTH_OPT_SSLKEYTYPE is...	The valid value for HWTH_OPT_SSLKEY is...
HWTH_SSLKEYTYPE_KEYDBFILE	The path and file name of the key database file.
HWTH_SSLKEYTYPE_KEYRINGNAME	One of the following: <ul style="list-style-type: none"> A SAF key ring name, in the form <i>userid/keyring</i> A PKCS #11 token, in the form <i>*TOKEN*/token_name</i>

HWTH_OPT_SSLKEYSTASHFILE

Specifies the path and file name of the password stash file that is created by the System SSL gskkyman utility. This option is required when **HWTH_OPT_USE_SSL** is set to **HWTH_SSL_USE** and **HWTH_SSLKEYTYPE** is **HWTH_SSLKEYTYPE_KEYDBFILE**.

HWTH_OPT_SSLCLIENTAUTHLABEL

An optional label that represents a client certificate. If SSL client authentication is requested by the server, this option allows you to specify a client certificate other than the default client certificate to be used in the SSL handshake.

HWTH_OPT_SSLCIPHERSPECS

An optional string value that represents the specification of the cipher suites to be used by System SSL. This option is set to make System SSL aware of an alternate cipher set other than the default set to be used for the connection.

The value string must use only 4-character cipher suite definitions and should be ordered by preference of use. Valid values will always have a length that is an even multiple of 4 characters because every cipher specification must be provided in its full 4-character form. The toolkit does not validate the contents of the string. An example of a valid value for this option is 003500380039002F00320033.

This option requires the **HWTH_OPT_USE_SSL** option to be set to **HWTH_SSL_USE**, indicating that SSL-related parameters supplied by the application, rather than an AT-TLS policy, must be used in establishing the secure connection.

See *z/OS Cryptographic Services System SSL Programming* for more details.

Note: When a secure connection is established, the set of ciphers offered by the client to the server helps determine how data will be encrypted and decrypted.

System SSL allows an application to replace the default cipher set with an alternate specification. That specification is a string which concatenates 1 or more of these cipher suite definitions (without delimiters).

Ciphers are identified by 2-character or 4-character cipher suite definitions. Every 2-character definition has a corresponding 4-character equivalent (the reverse is not true).

Proxy options

For more information about using a proxy with the toolkit, see [“Using a proxy server” on page 581](#) in [“Syntax, linkage, and programming considerations” on page 577](#).

HWTH_OPT_PROXY

An optional buffer that specifies the HTTP proxy to use. This value should be in the same form as the **HWTH_OPT_URI** value for connections.

HWTH_OPT_PROXYPORT

An optional 4-byte integer indicating the proxy port to which to connect. The option is required if you specify the **HWTH_OPT_PROXY**.

HWTH_OPT_PROXYAUTH

An optional 4-byte integer value that specifies the proxy authentication type to be used for all proxy requests on this connection. Valid values are:

HWTH_PROXYAUTH_NONE

No proxy authorization is used.

HWTH_PROXYAUTH_BASIC

Use basic proxy authentication. The user name and password, as specify by the **HWTH_OPT_PROXYAUTH_USERNAME** and **HWTH_OPT_PROXYAUTH_PASSWORD** options, are processed and sent to the proxy as prescribed by the Basic authentication format. See [RFC 7617](https://tools.ietf.org/html/rfc7617) (tools.ietf.org/html/rfc7617).

HWTH_OPT_PROXYAUTH_USERNAME

An optional buffer that contains the user name to be used as part of authorizing to a proxy that requires Basic authentication. This option is used in combination with the **HWTH_OPT_PROXYAUTH** and **HWTH_OPT_PROXYAUTH_PASSWORD** options, and is required when **HWTH_OPT_PROXYAUTH** is set to **HWTH_PROXYAUTH_BASIC**.

HWTH_OPT_PROXYAUTH_PASSWORD

An optional buffer that contains the password to be used as part of authorizing to a proxy that requires Basic authentication. This option is used in combination with the **HWTH_OPT_PROXYAUTH** and **HWTH_OPT_PROXYAUTH_USERNAME** options, and is required when **HWTH_OPT_PROXYAUTH** is set to **HWTH_PROXYAUTH_BASIC**.

Cookie options**HWTH_OPT_COOKIE TYPE**

An optional 4-byte integer value that specifies the cookie store engine behavior to be used. Unlike other options, this option takes effect immediately, even after the connection has been established. Valid values are:

HWTH_COOKIE TYPE_NONE

Turns off cookie support.

- Cookies set with the **HWTH_OPT_COOKIE** request option or cookies that are sent in the "Cookie" header as part of the **HWTH_OPT_HTTPHEADERS** option will be passed to the server, but will not be retained for future use.
- If the toolkit cookie engine had been enabled previously, all cookies in the in-memory cookie store is deleted.

Non-REXX:

All cookies received in "Set-Cookie" response headers will be passed to the response header processor as specified by **HWTH_OPT_RESPONSEHDR_EXIT**, but will not be retained for future use by the toolkit.

REXX:

All cookies received in "Set-Cookie" response headers will be passed to the application in the REXX stem specified by the **HWTH_OPT_RESPONSEHDR_USERDATA** option.

HWTH_COOKIE TYPE_SESSION

Causes the toolkit cookie engine to activate and to start saving and sending session cookies for this connection. The cookies will be only available to the application while the connection is active and will not persist after the connection has ended.

- On each subsequent request, all eligible cookies that match the criteria specified by the cookie is propagated to the server. (An *eligible* cookie means a cookie that passes the expiration, domain, path, and secure filters).

- The **HWTH_OPT_COOKIE** request option sends any user-specified cookies in addition to any eligible cookies sent by the cookie engine, but will they not be retained by the engine.
- All cookies received in response headers will automatically be added as session cookies for this connection.
- If the application provides its own "Cookie" header as part of the **HWTH_OPT_HTTPHEADERS** option, this header is used and the cookie engine will not send any eligible cookies. In this case, all cookies specified by the **HWTH_OPT_COOKIE** request option is also ignored.
- The session cookies can be primed from an input cookie buffer by using the set services with the **HWTH_OPT_COOKIE_INPUT_BUFFER** option.

HWTH_COOKIE_TYPE_PERSIST

Causes the toolkit to start saving persistent cookies for this connection. If the **HWTH_OPT_COOKIE_OUTPUT_BUFFER** option has been set, the cookies are written to the application's output cookie buffer during connection disconnect processing. If **HWTH_OPT_COOKIE_OUTPUT_BUFFER** has not been set, this cookie type has the same behavior as **HWTH_COOKIE_TYPE_SESSION**.

- On each subsequent request, all eligible cookies that match the criteria specified by the cookie is propagated to the server. (An *eligible* cookie means a cookie that passes the expiration, domain, path, and secure filters).
- The **HWTH_OPT_COOKIE** request option sends any user-specified cookies in addition to any eligible cookies sent by the cookie engine, but they are not retained by the engine.
- All cookies received in response headers is automatically added as persistent cookies for this connection.
- If the application provides its own "Cookie" header as part of the **HWTH_OPT_HTTPHEADERS** option, this header is used and the cookie engine will not send any eligible cookies. In this case, all cookies specified by the **HWTH_OPT_COOKIE** request option is also ignored.
- The persistent cookies can be primed from an input cookie buffer by using the set services with the **HWTH_OPT_COOKIE_INPUT_BUFFER** option.

HWTH_OPT_COOKIE_INPUT_BUFFER

Optionally specifies a buffer containing a saved copy of the toolkit's cookie data store (cookie jar). This buffer can be used to prime the toolkit's cookie store for a new connection. (A previous connection with a **HWTH_OPT_COOKIE_TYPE** of **HWTH_COOKIE_TYPE_PERSIST** was disconnected, resulting in the toolkit writing its cookie store to a user-provided buffer, as specified by **HWTH_OPT_COOKIE_OUTPUT_BUFFER**).

- If you specify the same address for this option as for **HWTH_OPT_COOKIE_OUTPUT_BUFFER** and the **HWTH_OPT_COOKIE_TYPE** is set to **HWTH_COOKIE_TYPE_PERSIST**, the input buffer is overlaid.
- This parameter is ignored if **HWTH_OPT_COOKIE_TYPE** is set to **HWTH_COOKIE_TYPE_NONE**.
- If you are setting this option using the REXX API, the options specify the REXX variable containing prior output from a cookie output buffer.

HWTH_OPT_COOKIE_OUTPUT_BUFFER

Optionally specifies a buffer for cookies to be saved when a connection is disconnected and the **HWTH_OPT_COOKIE_TYPE** is set to **HWTH_COOKIE_TYPE_PERSIST**.

- If you specify the same address for this option as for **HWTH_OPT_COOKIE_INPUT_BUFFER** and the **HWTH_OPT_COOKIE_TYPE** is set to **HWTH_COOKIE_TYPE_PERSIST**, the input buffer is overlaid.
- This parameter is ignored if **HWTH_OPT_COOKIE_TYPE** is not set to **HWTH_COOKIE_TYPE_PERSIST**.
- If you are setting this option using the REXX API, the option specifies a REXX variable where the cookies are returned after a disconnect.

Guideline: The output buffer (cookie jar) specified here must be large enough to hold the cookies plus the metadata information about each cookie that is maintained by the cookie engine. Depending on how many cookies you plan to receive from the server (current maximum is 100), use the following formula to compute the size needed for the output buffer:

$$\text{number_of_cookies} \times (\text{average_cookie_data_size} + 1\text{K})$$

For instance, if the application is to hold 100 cookies with a maximum cookie size of 4 K bytes per cookie, the length of the output buffer would be: $100 \times (4\text{ K} + 1\text{ K}) = 500\text{K}$

Options for requests

Options can be set, one at a time, by using the HWTHSET service. (See [“HWTHSET — Set HTTP connection or request options”](#) on page 624).

HWTH_OPT_URI

The Uniform Resource Identifier (URI). This supplied buffer is the target location of the target resource of the request. This option is optional for a request.

The name or resource (URN path portion) of the URI. The query and fragment portions of a URI may also be present.

Examples:

```
/systems/z/  
/over/here?name=abc#frag1
```

HWTH_OPT_REQUESTMETHOD

A required 4-byte integer specifying the wanted HTTP create, read, update, and delete (CRUD) request methods. Valid values are:

HWTH_HTTP_REQUEST_POST

Use the POST method.

HWTH_HTTP_REQUEST_GET

Use the GET method.

HWTH_HTTP_REQUEST_PUT

Use the PUT method.

HWTH_HTTP_REQUEST_DELETE

Use the DELETE method.

HWTH_HTTP_REQUEST_HEAD

Use the HEAD method.

HWTH_HTTP_REQUEST_PATCH

Use the PATCH method.

HWTH_HTTP_REQUEST_OPTIONS

Use the OPTIONS method.

HWTH_OPT_HTTP_VERSION

An optional 4-byte integer specifying the wanted HTTP version. Valid values are:

HWTH_HTTP_VERSION_NONE

The toolkit chooses the default value (currently, HTTP/1.1).

HWTH_HTTP_VERSION_1_0

Use HTTP/1.0.

HWTH_HTTP_VERSION_1_1

Use HTTP/1.1.

HWTH_OPT_HTTPHEADERS

An optional 4-byte sList handle, as returned by the HWTHSLST service, which contains a linked list of HTTP request headers. These headers are sent as-is, without any modification by the toolkit. If a

HTTP enabler options

header is specified that the toolkit might add by default (for instance, Host or Cookie), these headers take precedence.

Note: The HTTP headers specified in the sList must not be terminated by a carriage return, line feed (CRLF), as the toolkit automatically terminates each of the headers with a CRLF.

HWTH_OPT_REQUESTBODY

This is useful mainly on an HTTP PUT, PATCH, or POST operation.

This option is mutually-exclusive with the HWTH_OPT_STREAM_SEND_EXIT option. If this option is set to a non-zero value, HWTH_OPT_STREAM_SEND_EXIT must be zero.

Non-REXX:

An optional 4-byte pointer to a single request body data buffer.

REXX:

A REXX variable name that contains the request body.

Notes:

1. Because the size of a request body could be substantial, the toolkit does not copy the buffer contents into the toolkit's work area. Therefore, any manipulation of the buffer data pointed to by this address after this option has been set and before the HWTHRQST service is called could yield undesired results.
2. If the data to be sent on the HTTP request cannot be contained in a single contiguous buffer, consider setting the HWTH_OPT_STREAM_SEND_EXIT option and write code in this exit to stream the data to the server piece by piece.

HWTH_OPT_STREAM_SEND_EXIT

An optional 4-byte address of a program to receive control when the HWTHRQST service is invoked to identify the data to be sent to the server. This exit will be called repeatedly until the exit has indicated that all data has been sent. This option is useful when the request body size is not known or is substantial. See [“Streaming send exit” on page 660](#) for more information about how this (exit) routine operates.

This option is mutually-exclusive with the HWTH_OPT_REQUESTBODY option. If this option is set to a non-zero value, the HWTH_OPT_REQUESTBODY option must be set to zero.

HWTH_OPT_REQUESTBODY_USERDATA

An optional 4-byte address of a user buffer to be passed into the streaming send exit. This can serve as a communication mechanism for the application to specify that the streaming send exit is to behave in a certain manner when it receives control for this particular request. For instance, it may be used for maintaining timing information to track the efficiency of the ongoing send request.

The exit will receive this value as part of the input parameters (the progress descriptor's user data field) the first time the streaming send exit receives control.

HWTH_OPT_TRANSLATE_REQBODY

An optional 4-byte integer value that specifies the codepage translation behavior to be performed on a request body. Valid values are:

HWTH_XLATE_REQBODY_NONE

The toolkit will not translate the request body and send it as-is to the server.

HWTH_XLATE_REQBODY_E2A

The toolkit attempts to translate the application-provided request body from EBCDIC (code page 1047) to ASCII (code page ISO8859-1).

HTTP authorization options

HWTH_OPT_HTTPAUTH

An optional 4-byte integer value that specifies the HTTP authentication level to be used on this request. Valid values are:

HWTH_HTTPAUTH_NONE

No HTTP authorization is to be built by the toolkit.

HWTH_HTTPAUTH_BASIC

Use HTTP basic client authentication. The user and password, as specified by the **HWTH_OPT_USERNAME** and **HWTH_OPT_PASSWORD** options, is sent in the clear in the prescribed basic client authentication format.

Note: This option is only recommended for an HTTPS connection.

HWTH_OPT_USERNAME

An optional buffer which contains the user name to be used as part of various authentication protocols (such as basic client authentication), used with the **HWTH_OPT_HTTPAUTH** and **HWTH_OPT_PASSWORD** options. This option is required when **HWTH_OPT_HTTPAUTH** is set to **HWTH_HTTPAUTH_BASIC**.

HWTH_OPT_PASSWORD

An optional buffer which contains the password to be used as part of various authentication protocols (such as basic client authentication), used with the **HWTH_OPT_HTTPAUTH** and **HWTH_OPT_USERNAME** options. This option is required when **HWTH_OPT_HTTPAUTH** is set to **HWTH_HTTPAUTH_BASIC**.

Response options**HWTH_OPT_RESPONSEHDR_EXIT (non-REXX)**

An optional 4-byte address of a program to receive control once for each response header received by the application. See [“Receiving data from a server \(non-REXX\)”](#) on page 661 for more information about how these callback (exit) routines operate.

HWTH_OPT_RESPONSEHDR_USERDATA**Non-REXX:**

An optional buffer of user data to be passed into the response header exit when it receives control. This can serve as a communication mechanism for the application to specify that the response header exit is to behave in a certain manner when it receives control for the response that is associated with this request.

REXX:

An optional buffer of a REXX API variable where response headers are returned. If the variable is not a stem, the variable name is appended with the values similar to: *var0*, *var1*, *var1.1*,... If the variable is the name of the stem:

- *stem.0* contains the number of the returned headers.
- *stem.n*, where *n* is a number from 1-*stem.0*, contains the header name.
- *stem.n.1* contains the header value.

HWTH_OPT_RESPONSEBODY_EXIT (non-REXX)

An optional 4-byte address of a program to receive control when the response body is received. See [“Receiving data from a server \(non-REXX\)”](#) on page 661 for more information about how these callback (exit) routines operate.

This option is mutually-exclusive with the **HWTH_OPT_STREAM_RECEIVE_EXIT** option. If this option is set to a non-zero value, the **HWTH_OPT_STREAM_RECEIVE_EXIT** option must be set to zero.

HWTH_OPT_STREAM_RECEIVE_EXIT

An optional 4-byte address of a program to receive control when the **HWTHRQST** service is invoked to accept the response body data returned from the server. This exit will be called repeatedly until the exit is notified that all data has been received. See [“Response body processing options”](#) on page 662 for more information about how this (exit) routine operates.

This option is mutually-exclusive with the **HWTH_OPT_RESPONSEBODY_EXIT** option. If this option is set to a non-zero value, the **HWTH_OPT_RESPONSEBODY_EXIT** option must be set to zero. For more information, see [“Large data body considerations”](#) on page 583.

HWTH_OPT_RESPONSEBODY_USERDATA

Non-REXX:

An optional buffer of user data to be passed into the response body exit or streaming receive exit when it receives control. This can serve as a communication mechanism for the application to specify that the exit is to behave in a certain manner when it receives control for the response associated with this request.

REXX:

The name of the REXX variable to contain the response body.

HWTH_OPT_TRANSLATE_RESPBODY

An optional 4-byte integer value that specifies the codepage translation behavior to perform on a response body. Valid values are:

HWTH_XLATE_RESPBODY_NONE

The toolkit will not translate the response body received from the server.

HWTH_XLATE_RESPBODY_A2E

The toolkit attempts to translate the response body received from the server from ASCII (code page ISO8859-1) to EBCDIC (code page 1047).

Cookie options

HWTH_OPT_COOKIE

An optional buffer containing one or more cookies to be explicitly specified in an HTTP request. The format of the value string should be *name=contents*, where *name* is the cookie name and *contents* is the value of the cookie. You can specify more than one cookie by separating each with a semicolon (;).

This option can work in conjunction with the cookie engine enabled (that is, with **HWTH_OPT_COOKIETYPE** set to **HWTH_COOKIETYPE_SESSION** or **HWTH_COOKIETYPE_PERSIST**). If enabled, any cookies specified by this option appear first in the list of cookies to be sent to the server, followed by any eligible cookies found by the cookie engine.

This option is ignored if the application has provided an explicit Cookie header as part of the **HWTH_OPT_HTTPHEADERS** option.

Capturing trace data through environment variables

The values for **HWTH_OPT_VERBOSE**, **HWTH_OPT_VERBOSE_OUTPUT** and **HWTH_OPT_SSLTRACE** connection handle options can be altered at runtime using environment variables with corresponding names. The value of the runtime environment variable, if valid, will take precedence over any value specified for that option using the **HWTHSET** service. A valid value for the integer option **HWTH_OPT_VERBOSE** is a string that matches the name of the value.

Override values that are not valid are ignored, and should not cause new errors. Specifically, any override that would cause its corresponding **HWTHSET** to fail is ignored, and any original application settings are preserved. An overridden DD name for **HWTH_OPT_VERBOSE_OUTPUT** that is not found at run time is ignored. An overridden path name for **HWTH_OPT_SSLTRACE** is always passed to SystemSSL unchanged. SystemSSL tolerates bad values but specifying bad paths may signal access violations.

For example, if a REXX application sets verbose to off using **HWTHSET**:

```
address hwthhttp hwthset ReturnCode SessionHandle HWTH_OPT_VERBOSE HWTH_VERBOSE_OFF DiagArea
```

The user executing the application, can set the **HWTH_OPT_VERBOSE** environment variable to **HWTH_VERBOSE_ON** to override the application setting and enable tracing.

Applications executed from z/OS UNIX operation environment

If the application is running in a Language Environment POSIX(ON) environment, then the user can set the **HWTH_OPT_VERBOSE** and **HWTH_OPT_SSLTRACE** override environment variables using the z/OS UNIX export command. See [Exporting variables in z/OS UNIX System Services User's Guide](#).

```
export HWTH_OPT_VERBOSE=HWTH_VERBOSE_UNREDACTED
export HWTH_OPT_SSLTRACE=/user/hwth/gskssl.trc
/usr/bin/mypgm parm1 parm2 parm3
```

Setting of the **HWTH_OPT_VERBOSE_OUTPUT** requires extra consideration. For the override value of **HWTH_OPT_VERBOSE_OUTPUT** to take effect, the program must run in the same address space that the output DD is allocated in. The following is an example of a REXX wrapper that performs the DD allocation and calls the passed in toolkit application.

Example:

wrapper.rexx

```
/* REXX */
parse arg prms
say "parms are:" || prms

diag='/tmp/' || userid() || '/diag'
call bpxwunix 'mkdir -p $(dirname' diag');rm -f' diag '2>/dev/null;touch' diag
if bpxwdyn("alloc fi(ggdd) path('"diag"') msg(2) reuse")<>0 then exit 1

say "Trace location setup:" || diag

call environment '_BPX_SHAREAS','MUST'
call environment 'HWTH_OPT_VERBOSE','HWTH_VERBOSE_UNREDACTED'
call environment 'HWTH_OPT_VERBOSE_OUTPUT',ggdd
rv=bpxwunix(prms)
call bpxwdyn 'free fi('ggdd')'
exit rv
```

usage

```
wrapper.rexx /usr/bin/mypgm parm1 parm2 parm3
```

The above wrapper can also be invoked for a REXX application running in z/OS UNIX.

Applications executed from a batch job

If a Language Environment application is invoked using BPXBATCH, the options depend on how BPXBATCH is used. If BPXBATCH invokes a shell (with the **SH** option) either a sequence of export statements can be added before the program is invoked or the values may be picked up from the caller's shell profile. If BPXBATCH uses the **PGM** option, the **//STDENV DD** is useful.

The following example demonstrates the BPXBATCH **PGM** option:

```
//TESTJOB      JOB ...
//STEP1       EXEC PGM=BPXBATCH
//STDPARM     DD *
PGM /usr/bin/mypgm parm1 parm2 parm3
/*
//STDENV     DD*
HWTH_OPT_SSLTRACE=/tmp/foo.%trc
HWTH_OPT_VERBOSE=HWTH_VERBOSE_UNREDACTED
/*
```

See [BPXBATCH](#) for general information and [Passing environment variables to BPXBATCH in z/OS UNIX System Services User's Guide](#) for details on how overrides can be specified.

If the application is NOT running in a Language Environment POSIX(ON) environment, then the user can set the above environment variables using the CEEOPTS DD statement. See [Using the CEEOPTS DD statement](#) in *z/OS Language Environment Programming Guide*.

Examples:

Example 1:

The following job step calls the REXX sample program SYS1.SAMPLIB(HWTHXR1) from TSO/E. The

//CEEOPTS DD is used to override the application's settings of two options:

```
HWTH_OPT_VERBOSE
HWTH_OPT_VERBOSE_OUTPUT
```

These overrides result in trace output being directed into a new dataset named MYHLQ.TESTOUT.SAMP.

```
//*-----
/* USING IKJEFT1A TO SUBMIT THE EXEC IN THE TSO ENVIRONMENT
/*-----
//REXXSTEP EXEC PGM=IKJEFT1A,DYNAMNBR=30,REGION=0M,TIME=9999
//SYSEXEC DD DSN=SYS1.SAMPLIB,DISP=SHR
//SYSTSIN DD *
%HWTHXR1
/*
//MYOVRDS DD DSN=MYHLQ.TESTOUT.SAMP,DISP=(NEW,CATLG,DELETE),
//          UNIT=SYSDA,RECFM=U,SPACE=(TRK,(1,1))
//CEEOPTS DD *
ENVAR(
  'HWTH_OPT_VERBOSE=HWTH_VERBOSE_ON',
  'HWTH_OPT_VERBOSE_OUTPUT=MYOVRDS'
)
/*
//SYSTSPRT DD SYSOUT=A
```

Example 2:

The following job step also calls the REXX sample program, but in this case the -V option is passed to the sample on the command line. This instructs the sample to use **HWTHSET** to set **HWTH_OPT_VERBOSE** to **HWTH_VERBOSE_ON**.

In this example, the CEEOPTS overrides cause *unredacted* trace output to be directed into a new file in /tmp.

```

/*-----
/* USING IKJEFT1A TO SUBMIT THE EXEC IN THE TSO ENVIRONMENT
/*-----
//REXXSTEP EXEC PGM=IKJEFT1A,DYNAMNBR=30,REGION=0M,TIME=9999
//SYSEXEC DD DSN=SYS1.SAMPLIB,DISP=SHR
//CEEOPTS DD *
ENVAR(
  'HWTH_OPT_VERBOSE=HWTH_VERBOSE_UNREDACTED',
  'HWTH_OPT_VERBOSE_OUTPUT=MYPATHDD'
)
/*
//MYPATHDD DD PATH='/tmp/testout.samp',
//          PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//          PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
//SYSTSIN DD *
%HWTHXR1 -V
/*
//SYSTSPRT DD SYSOUT=A

```

Applications executed from a TSO/E operation environment

Similar to a batch job, an application running in a TSO/E environment can set the above environment variables using the **CEEOPTS DD** statement.

In the following example, CEEOPTS points to a sequential data set with the following content:

```

ENVAR("HWTH_OPT_VERBOSE=HWTH_VERBOSE_UNREDACTED",
      "HWTH_OPT_VERBOSE_OUTPUT=GGDD",
      "HWTH_OPT_SSLTRACE=/tmp/hwt/gskssl1.trc")

```

A user in a TSO/E environment would first allocate the verbose output DD name and CEEOPTS prior to calling the REXX application:

```

alloc ddname(GGDD) path('/tmp/hwt/mystuff.trace') shr
alloc ddname(CEEOPTS) da('hwt.ibmuser.tracing.config')
ex 'hwt.test.rexx(myexec)'

```

See [Using the CEEOPTS DD statement in z/OS Language Environment Programming Guide](#).

See [Using runtime options in z/OS Language Environment Programming Guide](#) for further details regarding setting runtime options.

For information about supported runtime options **HWTH_OPT_VERBOSE**, **HWTH_OPT_VERBOSE_OUTPUT** and **HWTH_OPT_SSLTRACE**, see [“Options for connections” on page 643](#).

Sending data to a server (non-REXX)

There are two methods to send data to an HTTP server. The size of the data being sent and the complexity of writing a callback routine can be factors in determining which approach is most suitable for the toolkit application.

- For small pieces of data that can easily fit into a single contiguous buffer, the use of the **HWTH_OPT_REQUESTBODY** option is generally the best choice.

- For large pieces or unpredictable amounts of data that might not easily fit into a single contiguous buffer, data that is spread across multiple buffers, or data that is generated in real-time (streamed), the streaming send exit might be a better choice.

Buffer with the HWTH_OPT_REQUESTBODY option

For this method, the application populates a buffer with the desired request body, gets the 4-byte address of this buffer, and sets the HWTH_OPT_REQUESTBODY option with this address value. When the HWTHRQST service is invoked, the buffer is sent as the request body.

Streaming send exit

The toolkit provides a mechanism for an application to provide a streaming send exit to allow a large request body to be sent to an HTTP server. Using this method, the toolkit can send a virtually unlimited amount of data to the application (up to 9 exabytes) through the staging of the request body using multiple buffers and multiple invocations of the exit. The parameter list specifications for the streaming send exit routine are described in the IBM-supplied include files, as listed in [“Programming interface files provided by the HTTP enabler” on page 577](#). For additional implementation details, see [“Usage considerations for the toolkit callback routines” on page 664](#).

You specify the streaming send exit by setting the HWTH_OPT_STREAM_SEND_EXIT option using the HWTHSET service. The service calls the exit repeatedly until the exit indicates that the entire request body has been sent or the exit indicates that the toolkit should terminate the send request.

The input parameters are explained as follows:

1. A 4-byte address that points to a progress descriptor area that is owned and maintained by the toolkit for the purpose of providing context information which may be useful to the exit. The progress descriptor area contains:
 - a. An address pointing to the request URI and its length to identify the request to the streaming exit
 - b. Two 8-byte, unsigned integers that act as running counts for the number of data chunks sent and the total number of bytes sent
 - c. A 4-byte address pointing to a user area for the exit to use as needed
 - d. A 4-byte address pointing to a area to indicate the status of the response from the server
2. A 4-byte integer used by the exit and the toolkit to convey the current state of the request. For instance, the exit may set the HWTH_STREAM_SEND_EOD state to inform the toolkit that the entire request body has been sent or the HWTH_STREAM_SEND_ABORT state to inform the toolkit that it should terminate the send request. The toolkit may also set the HWTH_STREAM_SEND_ERROR state to inform the exit that an unexpected send error has occurred, so that the exit can react appropriately. The toolkit will set the state to HWTH_STREAM_SEND_RETRY when last sending attempt fails, but the toolkit successfully reconnects to the same server. Upon receiving HWTH_STREAM_SEND_RETRY, the exit should resend the same request from the beginning.

Notes:

- The exit may set the HWTH_STREAM_SEND_EOD state at the same time that it supplies the final data of the request body, or it may elect to wait and set the HWTH_STREAM_SEND_EOD state on the next callback that follows.
- The HWTH_STREAM_SEND_COMPLETE and HWTH_STREAM_SEND_ERROR states are specified by the toolkit to inform the exit that there are no subsequent callbacks to the exit for the current request.
- The initial state is set by the toolkit as HWTH_STREAM_SEND_CONTINUE.
- The toolkit treats any unsupported state value as a fatal error and results in a final callback with the state of HWTH_STREAM_SEND_ERROR.

- The exit should not set the status to `HWTH_STREAM_SEND_RETRY`, however if it does, the toolkit treats it as a fatal error.
 - If the exit receives `HWTH_STREAM_SEND_RETRY` status and does not modify the status to `HWTH_STREAM_SEND_CONTINUE`, `HWTH_STREAM_SEND_EOD` or `HWTH_STREAM_SEND_ABORT`, the toolkit receives `HWTH_STREAM_SEND_RETRY` as the status and treats it as a fatal error.
3. A 4-byte address that points to an array of data descriptors to describe an ordered list of contiguous data areas that comprise the next payload of the request body data to be sent to the remote HTTP server. Each array entry consists of a 4-byte address pointing to the start of the piece of request body being described and a 4-byte signed integer to indicate the size, in bytes, of the piece of data.

Notes:

- The array of data descriptors is owned and maintained by the exit. An empty array (an array whose first element is zero) or an array containing one or more nonsensical elements (descriptions with null addresses or negative lengths) while in `HWTH_STREAM_SEND_CONTINUE` state are treated as fatal errors.
 - To prevent any inadvertent duplication of sent data, the data descriptor array that is supplied to a successful streamed send operation is set to zeros. The toolkit makes no further assumptions about descriptor array persistence or reuse (on subsequent interactions). It assumes only that the array and the data areas that its elements describe persist for the full duration of time until the toolkit next returns control to the exit. The stream send exit must not provide reference to any storage area whose volatility compromises this assumption.
 - A request body of a size not exceeding 9 exabytes may be conveyed using repetitive interactions. On any given interaction, the exit is free to specify any number and size (or sizes) of next pieces of request body data, subject only to the inherent limitations of the data descriptor type (that is, no piece can exceed 2 gigabytes).
4. A 4-byte integer to contain the number of elements in the data descriptor array.

Additional streaming send exit details

- If the application supplies a `<Content-Length: NN>` header on a request that has a streaming send exit in effect, the toolkit does not use chunked encoding to send the NN bytes of request body content. This would be appropriate in cases where the body recipient does not support such an encoding.
- If the application supplies both `<Content-Length: NN>` and `<Transfer-Encoding: Chunked>` headers on a request, thereby violating the HTTP protocol, the toolkit treats the request as invalid.
- If the application supplies no `<Content-Length: NN>` header on a request that has a streaming send exit in effect, the toolkit sends a streamed request body to the receiving endpoint via chunked encoding. To comply with the HTTP protocol, a `<Transfer-Encoding: Chunked>` header must accompany the request. If the application fails to supply this header, the toolkit supplies one on its behalf.

Receiving data from a server (non-REXX)

A response from an HTTP server is comprised of two pieces: the response headers and the response buffer. The toolkit allows an application to be informed of all response headers sent from the HTTP server through the use of the response header callback (exit) routine. In addition, the toolkit gives the application a choice of two different methods to receive the response body. The size of the data being received and the complexity of writing a streaming callback routine can be factors in determining which approach is most suitable for the toolkit application.

- For small pieces of data that can easily fit in a single contiguous buffer, the use of the response body callback (exit) routine is generally the best choice.

Note: In the non-streaming case, the contents of a response body are only guaranteed to remain available in storage while the exit is running. If you need to manipulate the contents of a response body, outside the scope of the response body exit, you must copy the data to your own storage before the exit returns.

- For large pieces or unpredictable amounts of data that might not easily fit in a single contiguous buffer, data that is spread across multiple buffers, or data that is generated in real-time (streamed), the streaming receive exit might be the better choice.

Note: For response bodies that are larger than available storage (memory), you must employ the streaming response body exit (option `HWTH_OPT_STREAM_RECEIVE_EXIT`).

The exact parameter list specifications for the callback exit routines are described in the IBM-supplied include files, as listed in [“Programming interface files provided by the HTTP enabler”](#) on page 577. For additional implementation details, see [“Usage considerations for the toolkit callback routines”](#) on page 664.

- [Processing response headers with the response header callback routine](#)
- [Response body processing options](#)
- [Usage considerations for the toolkit callback routines](#)

Processing response headers with the response header callback routine

Whenever an application would like to know about all the response headers coming back from a given response, the application can set the `HWTH_OPT_RESPONSEHDR_EXIT` option to the address of a routine to process the headers. The callback routine is driven once for every response header, regardless of the overall status value sent back in the response. The status line of the response, the response header name, response header value, response header user data, and exit flags are passed as parameters to the callback routine. In addition, exit flags can be sent to the callback routine to indicate that a problem was detected during the processing of a particular response header before invoking the callback routine. The callback routine can react to these exit flags or ignore them. The callback routine also has the option to specify a return code back to the toolkit to indicate whether to continue processing the response (`HWTH_RESP_EXIT_RC_OK`) or to terminate further processing (`HWTH_RESP_EXIT_RC_ABORT`).

The response header callback routine (exit) normally gets driven one or more times (one for each response header) before the response body callback routine gets control (if data was sent from the server in the body). If the data returned from the server uses the chunked encoding transfer method and includes trailers, the header exit is driven once for each trailer, with an additional indicator set in the exit flags to indicate that this particular header is, in fact, a trailer.

Response body processing options

The toolkit provides the following options to process the response body:

- Response body callback (exit) routine
- Streaming receive exit

Response body callback (exit) routine

Whenever an application would like to know about the response body coming back from a given response, the application can set the `HWTH_OPT_RESPONSEBODY_EXIT` option to the address of a routine to process the body. The callback routine is driven once per request, regardless of the status value sent back in the response. The status line of the response, the response body, and response body user data are passed as parameters to the callback routine.

Under certain circumstances, the response body callback routine can be called with a response body having a length of zero. The programming of this callback routine should accommodate this possible condition.

The HTTP enabler supports the chunked encoding data transfer method (Transfer-Encoding: chunked). The toolkit automatically dechunks data that the server sent in chunk-encoded form. The response body data is presented to the exit as a contiguous whole with no trace of the encoding used in transfer. Any (optional) trailers detected at end of response are presented sequentially to the response header exit before the response body is presented to the response body exit.

Note: Any (optional) chunk extensions incorporated by the server into the chunk encoding are ignored by the HTTP enabler.

Streaming receive exit

The toolkit provides a mechanism for an application to provide a streaming receive exit to allow for a large response body to be received from an HTTP server. Using this method, the toolkit can receive a virtually unlimited amount of data to the application (up to 9 exabytes) through the staged receive of the response body using multiple buffers and multiple invocations of the exit.

You specify the stream receive exit by setting the `HWTH_OPT_STREAM_RECEIVE_EXIT` option using the `HWTHSET` service. The exit is called repeatedly until the entire response body has been conveyed by the toolkit to the exit, or the exit indicates that the toolkit should terminate the receive request.

The input parameters are as follows:

1. A 4-byte address pointing to a progress descriptor area that is owned and maintained by the toolkit for the purpose of providing context information which may be useful to the exit. This area contains:
 - a. The address and length of the request URI, which identifies the request to the streaming exit
 - b. Two 8-byte, unsigned integers, which act as running counts for the total number of bytes received and the number of chunks encountered
 - c. A 4-byte address that points to a user area for the exit to use as necessary
 - d. A 4-byte address that points to an area to indicate the status of the response from the server

Note: The toolkit clears the user data field initially on the first call to the exit unless the `HWTH_OPT_REQUESTBODY_USERDATA` option has been set, in which case the field is primed with the user-supplied option value. Thereafter, the toolkit neither reads nor writes the field, and any data area attached by the exit shall be owned and managed by the exit.

2. A 4-byte integer used by the exit and the toolkit to convey the current state of the request. For instance, the toolkit may set the `HWTH_STREAM_RECEIVE_EOD` state to inform the exit that the entire response body has been supplied to the exit or the `HWTH_STREAM_RECEIVE_ERROR` state to inform the exit that an unexpected (fatal) receive error has occurred. Similarly, the exit itself may set `HWTH_STREAM_RECEIVE_ABORT` to indicate that the toolkit should terminate the receive request.

Notes:

- On all but the initial callback, the exit examines the input state parameter for any `HWTH_STREAM_RECEIVE_EOD` indication by the toolkit that the response body has been completely conveyed. When this state is detected, the stream receive exit is recommended to set the `HWTH_STREAM_RECEIVE_COMPLETE` state to inform the toolkit as a final acknowledgment that the completed response body has been accepted.
 - The states of `HWTH_STREAM_RECEIVE_COMPLETE` and `HWTH_STREAM_RECEIVE_ERROR` are specified by the toolkit to inform the exit that there are no subsequent callbacks to the exit for the current request.
 - The initial state is set by the toolkit as `HWTH_STREAM_RECEIVE_CONTINUE`.
 - Any unsupported state value detected by the toolkit will be treated as a fatal error and it will result in a final callback with the state of `HWTH_STREAM_RECEIVE_ERROR`.
3. A 4-byte address pointing to an array of data descriptors whose elements describe writable buffers supplied by and owned by the exit. This is called the *supply list*. Response body data returned by the server is written in order, directly to the buffers described by the current supply list.

The exit may specify a supply list describing the same list of buffers on each callback, or may change the supply list at any time. The exit has complete flexibility to determine the number, size (or sizes), and location (or locations) of supply list buffers on a given callback. Not all data written to the supply list buffers is of interest to the exit, however. The exit must consult the return list parameter (described later, under parameter number “5” on page 664) to properly consume the returned data.

Note: The toolkit assumes that the supply list and the buffers its elements describe persist for the full duration of time until the toolkit next returns control to the exit. The stream receive exit must not provide reference to any storage area whose volatility compromises this assumption.

4. A 4-byte integer to contain the number of elements in the data descriptor array of the supply list.
5. A 4-byte address that points to an array of data descriptors whose elements describe areas of response body data written within buffers described by the supply list from the previous callback. This list is called the *return list*.

Owned and maintained by the toolkit, the return list uses the same type of descriptor elements to describe the locations and sizes of actual response body data (as opposed to possible metadata fragments that may be interleaved with actual body data pieces as artifacts of chunked encoding). That is, rather than performing a costly dechunking of chunk-encoded data that involves extensive shifting and recopying, the toolkit decodes the data, effectively hiding any metadata and describing only the actual body data pieces in the return list. In cases where the response body was not chunk-encoded, the return list resembles the supply list (as there is no metadata to exclude). The exit should consume all pieces of data described by the return list, in order, and must not modify or delete the list itself.

6. A 4-byte integer to contain the number of elements in the data descriptor array of the return list.

Usage considerations for the toolkit callback routines

- The callback routines must use standard z/OS linkage when receiving control from the toolkit (See [“Linkage considerations for high-level language programming”](#) on page 578 or [“Linkage considerations for assembler language programming”](#) on page 578 for the callback routine linkage requirements. Those instructions describe the same linkage requirements that these exits must follow). Failure to implement the proper entry and exit linkage in the callback routine could result in ABENDs from within the toolkit or cause the HWTHRQST service call to hang.
- Use care to ensure that these exits do not take excessive time when processing the responses. Excessive processing time could cause the socket connection to time out or for the HWTHRQST service to hang for an undesired amount of time.
- Samples for some of these callback routines are provided in the sample code as specified in [“z/OS HTTP enabler programming examples”](#) on page 589.

Note: In the case of a non-streaming request, the contents of any response body are only available while the exit is running. To manipulate the body's contents outside the exit, you must copy the body data to your own storage before the exit returns.

Part 10. SMF Services

In addition to the callable services described in the following topics, you can find information about other SMF programming interfaces in [*z/OS MVS System Management Facilities \(SMF\)*](#).

Chapter 22. SMF real-time interface

SMF provides an application programming interface (API) that offers real-time access to SMF in-memory resources.

The following callable services support real-time access to SMF records:

- “[IFAMCON — Connect to an SMF in-memory resource](#)” on page 667
- “[IFAMGET — Obtain data from an SMF in-memory resource](#)” on page 673
- “[IFAMDSC — Disconnect from an SMF in-memory resource](#)” on page 670
- “[IFAMQRY — Query SMF in-memory resources](#)” on page 677

The following steps describe the expected calling sequence:

1. Optional: Call the IFAMQRY service to determine which SMF in-memory resources are available to the application.
2. Call the IFAMCON service to connect to an in-memory resource.
3. Call the IFAMGET service in a loop to collect SMF records that are already in the in-memory resource or that are being recorded in real time.
4. Call the IFAMDSC service to disconnect from the in-memory resource and clean up resources.

For more information, see the topic on using the SMF real-time interface in [z/OS MVS System Management Facilities \(SMF\)](#).

IFAMCON — Connect to an SMF in-memory resource

Call the IFAMCON service to connect to an SMF in-memory resource.

Description

The IFAMCON service connects to SMF for in-memory data capture. The service establishes an environment to call the IFAMGET service to obtain SMF data from an in-memory resource. The caller must provide the target in-memory resource to access.

Notes:

- An active connection to an in-memory resource does not prevent the SET SMF=xx command from removing that resource from the configuration. No new data will be recorded to that resource; however, the resource is not removed from the configuration until the last connection disconnects. You can use the DISPLAY SMF,M command to display the connections to in-memory resources.
- An active connection prevents the SET SMF=xx command from changing the in-memory resource definition, such as changes to the **TYPE** or **RESSIZMAX** parameters.
- The system supports a maximum of 8 connections per in-memory resource, with a maximum of 128 connections per system.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Problem state, PSW key 8-15
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN = SASN
AMODE:	64-bit

Requirement	Details
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.

Programming requirements

The caller must include the IFAZSYSP macro to get a mapping of the query parameter block. The caller should include the IFARCINM macro to get equate symbols for the return and reason codes.

Authorization

The caller requires READ access to the SAF resource protecting the in-memory resource to which the caller wants to connect.

Restrictions

None.

Syntax

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

```
Call IFAMCON, (
                ConParmBlock,
                rc,
                rsn);
```

In assembler, it is recommended that you link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB.

An alternative method can be used to invoke IFAMCON:

```
LLGT 15,16(0,0)    Get CVT
LLGT 15,196(,15)
L     15,376(,15)
LLGT 15,180(,15)
L     15,4(,15)     Load address of IFAMCON
CALL (15),(ConParmBlock,rc,rsn)
```

The calling program should invoke SYSSTATE AMODE64(YES) before calling the service.

Parameters

The parameters are explained as follows:

ConParmBlock

Supplied parameter that identifies a connection parameter block.

Type: Connection parameter block

The connection parameter block has the following format and is mapped by the IFAZSYSP macro in SYS1.MACLIB:

Decimal offset	Length	Type	Field	Description
0	4	EBCDIC	ID	ID with value of CNPB

Decimal offset	Length	Type	Field	Description
4	2	binary	Length	Total length of the parameter block
6	1	binary	Unused	Unused; must be zero
7	1	binary	Version	Parameter block version number (X'01')
8	4	binary	Reserved	Reserved; must be zero
12	2	binary	Name length	Length of the name of the in-memory resource
14	26	EBCDIC	Name	Name of the in-memory resource, padded with blanks
40	16	binary	Reserved	Reserved; must be zero
56	16	binary	Token	Output token for access to the IFAMGET service
72	34	binary	Reserved	Reserved; must be zero

rc

Returned parameter that identifies the return code from the service.

Type: 4-byte integer

rsn

Returned parameter that identifies the reason code from the service.

Type: 4-byte integer

ABEND codes

IFAMCON might abnormally end with system completion code X'353'. See [z/OS MVS System Codes](#) for an explanation and programmer response.

Related services

[“IFAMDSC — Disconnect from an SMF in-memory resource” on page 670](#)

Return and reason codes

When the service returns control to the caller, the **rc** parameter contains a hexadecimal return code and the **rsn** parameter contains a hexadecimal reason code, as listed in [Table 127 on page 669](#). The return code and reason code symbols are mapped by the IFARCINM macro.

Table 127. Return and reason codes for the IFAMCON service		
Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'00' IFAINMRetCodeOK	X'0000' IFAINMRsnCodeOK	Meaning: Successful completion. Action: None.

Table 127. Return and reason codes for the IFAMCON service (continued)

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'08' IFAINMRetCodeError	X'0801' IFAINMBadMode	Meaning: The caller is running in an incorrect mode for one or more of the following reasons: <ul style="list-style-type: none"> • The caller is not running in task mode. • The caller is in cross-memory mode. • The caller is holding a lock. Action: Change the program to run under a task in PASN = HASN mode with no locks held.
X'08' IFAINMRetCodeError	X'0802' IFAINMBadParmlist	Meaning: The parameter block is not accessible or has an incorrect format. Action: Correct the program to pass a valid parameter block.
X'08' IFAINMRetCodeError	X'0803' IFAINMNoConnections	Meaning: There are no available connections. Action: Determine the cause for the lack of connections.
X'08' IFAINMRetCodeError	X'0805' IFAINMUnsupported	Meaning: The caller is attempting to pass unsupported options in the parameter block. Action: Initialize all unused fields in the parameter block to zero.
X'08' IFAINMRetCodeError	X'0807' IFAINMNoSuchResource	Meaning: The specified in-memory resource either does not exist or the caller does not have access to it. Action: Correct the resource name or ensure that the caller has access to the resource.
X'0C' IFAINMRetCodeEnvErr	X'0C02' IFAINMSMFNotActive	Meaning: SMF is not active. Action: None.
X'10' IFAINMRetCodeFatal	Not applicable	Meaning: Internal error. Action: Contact the IBM Support Center.

IFAMDSC – Disconnect from an SMF in-memory resource

Call the IFAMDSC service to disconnect from an SMF in-memory resource.

Description

The IFAMDSC service disconnects from an SMF in-memory resource when the calling program no longer needs to request any more data via the IFAMGET service.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Problem state, PSW key 8-15
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN = SASN
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts.

Requirement**Details****Locks:**

No locks held.

Programming requirements

A prior successful call to the IFAMCON service is required.

Authorization

None.

Restrictions

The disconnect request must be issued from the same address space as the connect request.

Syntax

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

```
Call IFAMDSC,(
           DscParmBlock,
           rc,
           rsn);
```

In assembler, it is recommended that you link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB.

An alternative method can be used to invoke IFAMDSC:

```
LLGT 15,16(0,0)    Get CVT
LLGT 15,196(,15)
L    15,376(,15)
LLGT 15,180(,15)
L    15,12(,15)     Load address of IFAMDSC
CALL (15),(DscParmBlock,rc,rsn)
```

The calling program should invoke SYSSTATE AMODE64(YES) before calling the service.

Parameters

The parameters are explained as follows:

DscParmBlock

Supplied parameter that identifies a disconnect parameter block.

Type: Disconnect parameter block

The disconnect parameter block has the following format and is mapped by the IFAZSYSP macro in SYS1.MACLIB:

Decimal offset	Length	Type	Field	Description
0	4	EBCDIC	ID	ID with value of DSPB
4	2	binary	Length	Total length of the parameter block
6	1	binary	Unused	Unused; must be zero

Decimal offset	Length	Type	Field	Description
7	1	binary	Version	Parameter block version number (X'01')
8	16		Token	Token provided by the IFAMCON service

rc

Returned parameter that identifies the return code from the service.

Type: 4-byte integer

rsn

Returned parameter that identifies the reason code from the service.

Type: 4-byte integer

ABEND codes

IFAMDSC might abnormally end with system completion code X'353'. See [z/OS MVS System Codes](#) for an explanation and programmer response.

Related services

- “[IFAMCON — Connect to an SMF in-memory resource](#)” on page 667
- “[IFAMGET — Obtain data from an SMF in-memory resource](#)” on page 673

Return and reason codes

When the service returns control to the caller, the **rc** parameter contains a hexadecimal return code and the **rsn** parameter contains a hexadecimal reason code, as listed in [Table 128 on page 672](#). The return code and reason code symbols are mapped by the IFARCINM macro.

Table 128. Return and reason codes for the IFAMDSC service		
Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'00' IFAINMRetCodeOK	X'0000' IFAINMRsnCodeOK	Meaning: Successful completion. Action: None.
X'04' IFAINMRetCodeWarn	X'0405' IFAINMGetInProgDsc	Meaning: There is currently a GET call in progress for this connection. The active GET call has been notified of the disconnection attempt. Action: Wait for the GET call to return before issuing another disconnect request.
X'04' IFAINMRetCodeWarn	X'0407' IFAINMDscInProgDsc	Meaning: There is currently a DISCONNECT call in progress for this connection. Action: None.
X'08' IFAINMRetCodeError	X'0801' IFAINMBadMode	Meaning: The caller is running in an incorrect mode for one or more of the following reasons: <ul style="list-style-type: none"> • The caller is not running in task mode. • The caller is in cross-memory mode. • The caller is holding a lock. Action: Change the program to run under a task in PASN = HASN mode with no locks held.

Table 128. Return and reason codes for the IFAMDSC service (continued)

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'08' IFAINMRetCodeError	X'0802' IFAINMBadParmlist	Meaning: The parameter block is not accessible or has an incorrect format. Action: Correct the program to pass a valid parameter block.
X'08' IFAINMRetCodeError	X'0804' IFAINMBadConToken	Meaning: The caller is attempting to pass an invalid token in the parameter block. Action: Correct the program to provide the correct token that was returned on the IFAMCON call.
X'08' IFAINMRetCodeError	X'0805' IFAINMUnSupported	Meaning: The caller is attempting to pass unsupported options in the parameter block. Action: Initialize all unused fields in the parameter block to zero.
X'0C' IFAINMRetCodeEnvErr	X'0C02' IFAINMSMFNotActive	Meaning: SMF is not active. Action: None.
X'0C' IFAINMRetCodeEnvErr	X'0C03' IFAINMObtainFailure	Meaning: SMF is unable to obtain storage to generate in-memory data in response to this request. Action: None.
X'10' IFAINMRetCodeFatal	Not applicable	Meaning: Internal error. Action: Contact the IBM Support Center.

IFAMGET — Obtain data from an SMF in-memory resource

Call the IFAMGET service to obtain data from an SMF in-memory resource.

Description

The IFAMGET service provides access to SMF data as it is being recorded to an in-memory resource. Depending on the requested input parameters, the service can return one or more records into the caller-provided output buffer. The service can optionally immediately return to the caller when there are no records currently being recorded to the in-memory resource, or it can wait for a new record to be recorded before returning. Refer to [“Parameters” on page 674](#) for more information about the service.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Problem state, PSW key 8-15
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN = SASN
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Programming requirements

The caller-provided output buffer must be at least 32,768 bytes long. IFAMGET may return one or more contiguous records into this buffer. Programs that consume the data in the buffer must be coded to use the first 2 bytes of each record as the length of the record.

Authorization

None.

Restrictions

None.

Syntax

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

```
Call IFAMGET,(
           GetParmBlock,
           rc,
           rsn);
```

In assembler, it is recommended that you link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB.

An alternative method can be used to invoke IFAMGET:

```
LLGT 15,16(0,0)    Get CVT
LLGT 15,196(,15)
L     15,376(,15)
LLGT 15,180(,15)
L     15,8(,15)     Load address of IFAMGET
CALL (15),(GetParmBlock,rc,rsn)
```

The calling program should invoke SYSSTATE AMODE64(YES) before calling the service.

Parameters

The parameters are explained as follows:

GetParmBlock

Supplied parameter that identifies a get parameter block.

Type: Get parameter block

The get parameter block has the following format and is mapped by the IFAZSYSP macro in SYS1.MACLIB:

Decimal offset	Length	Type	Field	Description
0	4	EBCDIC	ID	ID with value of GTPB
4	2	binary	Length	Total length of the get parameter block
6	1	binary	Unused	Unused; must be zero
7	1	binary	Version	Parameter block version number (X'01')

Decimal offset	Length	Type	Field	Description
8	4	binary	Flags	The following flags are supported: Value Meaning X'80' Return multiple records in a single call. X'40' Reserved; must be zero. X'20' Return to the caller immediately when an SMF30 subtype 5 record is returned. X'10' If no record is available, return immediately to the caller.
12	4	binary	Reserved	Reserved; must be zero
16	16	binary	Token	Token provided by the IFAMCON service
32	4	binary	Buffer length	Length of the caller-provided output buffer Requirement: The buffer length must be 32,768 bytes or larger.
36	16	binary	Reserved	Reserved; must be zero
52	4	binary	Returned length	Length of the data returned in the output buffer
56	8	binary	Buffer address	Pointer to the caller-provided output buffer that is to hold the returned SMF records

rc

Returned parameter that identifies the return code from the service.

Direction: Output

Type: 4-byte integer

rsn

Returned parameter that identifies the reason code from the service.

Direction: Output

Type: 4-byte integer

ABEND codes

IFAMGET might abnormally end with system completion code X'353'. See [z/OS MVS System Codes](#) for an explanation and programmer response.

Related services

- “[IFAMCON — Connect to an SMF in-memory resource](#)” on page 667
- “[IFAMDSC — Disconnect from an SMF in-memory resource](#)” on page 670

Return and reason codes

When the service returns control to the caller, the **rc** parameter contains a hexadecimal return code and the **rsn** parameter contains a hexadecimal reason code, as listed in [Table 129 on page 676](#). The return code and reason code symbols are mapped by the IFARCINM macro.

Table 129. Return and reason codes for the IFAMGET service		
Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'00' IFAINMRetCodeOK	X'0000' IFAINMRsnCodeOK	Meaning: Successful completion. Action: None.
X'04' IFAINMRetCodeWarn	X'0401' IFAINMMissedData	Meaning: Records were skipped due to buffer re-use—that is, wrapping of the data in the in-memory resource. In this case, the output buffer might not contain a valid record. Action: The cursor is re-synched to the newest data on the next GET call. The application must be able to handle the missing data condition.
X'04' IFAINMRetCodeWarn	X'0402' IFAINMNoMoreData	Meaning: No records are available due to the removal of the in-memory resource from the configuration. In this case, the output buffer does not contain a valid record. Action: The caller should disconnect from the resource.
X'04' IFAINMRetCodeWarn	X'0403' IFAINMNoMoreDataTmp	Meaning: No records are available for this non-blocking request. In this case, the output buffer does not contain a valid record. Action: The caller can reinvoke the IFAMGET service to check for new records.
X'04' IFAINMRetCodeWarn	X'0404' IFAINMGetInProgGet	Meaning: A GET call is in progress for this connection. Action: Wait for the active GET call to complete before issuing another GET call.
X'04' IFAINMRetCodeWarn	X'0406' IFAINMDscInProgGet	Meaning: The caller attempted a GET call while a DISCONNECT call was in progress for this connection. When the active DISCONNECT call completes, the connection will no longer be active. Action: Reestablish a connection if you wish to perform a GET call.
X'04' IFAINMRetCodeWarn	X'0408' IFAINMGetForcedOut	Meaning: A request was made to disconnect an active connection while a GET call was in progress. The GET call ends, and subsequent GET calls cannot be issued for this connection. Action: Reestablish a connection if you wish to perform a GET call.
X'08' IFAINMRetCodeError	X'0801' IFAINMBadMode	Meaning: The caller is running in an incorrect mode for one or more of the following reasons: <ul style="list-style-type: none"> • The caller is in cross-memory mode. • The caller is holding a lock. Action: Change the program to run under a task or SRB in PASN = HASN mode with no locks held.

Table 129. Return and reason codes for the IFAMGET service (continued)

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'08' IFAINMRetCodeError	X'0802' IFAINMBadParmlist	Meaning: The parameter block is not accessible or has an incorrect format. Action: Correct the program to pass a valid parameter block.
X'08' IFAINMRetCodeError	X'0804' IFAINMBadConToken	Meaning: The caller is attempting to pass an invalid token in the parameter block. Action: Correct the program to provide the correct token that was returned on the IFAMCON call.
X'08' IFAINMRetCodeError	X'0805' IFAINMUnSupported	Meaning: The caller is attempting to pass unsupported options in the parameter block. Action: Initialize all unused fields in the parameter block to zero.
X'08' IFAINMRetCodeError	X'0806' IFAINMNotEnoughSpace	Meaning: There is not enough space in the output buffer to hold the returned record. Action: Call the IFAMGET service with an output buffer that is large enough to contain the record.
X'0C' IFAINMRetCodeEnvErr	X'0C02' IFAINMSMFNotActive	Meaning: SMF is not active. Action: None.
X'10' IFAINMRetCodeFatal	Not applicable	Meaning: Internal error. Action: Contact the IBM Support Center.

IFAMQRY – Query SMF in-memory resources

Call the IFAMQRY service to query the SMF in-memory resources that are available to the application.

Description

Your application can call the IFAMQRY service to determine which SMF in-memory resources are available. Only those in-memory resources that are available to this caller, as determined by SAF, are returned.

Note: The returned data represents point-in-time information that is subject to change because of configuration changes before a call to IFAMCON is made. Results are determined based on the caller's access to the data.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Problem state, PSW key 8-15
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN = SASN
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Programming requirements

The caller must include the IFAZSYSP macro to get a mapping of the query parameter block. The caller should include the IFARCINM macro to get equate symbols for the return and reason codes.

Authorization

None.

Restrictions

None.

Syntax

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

```
Call IFAMQRY,(
    QryParmBlock,
    rc,
    rsn);
```

In assembler, it is recommended that you link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB.

An alternative method can be used to invoke IFAMQRY:

```
LLGT 15,16(0,0)    Get CVT
LLGT 15,196(,15)
L    15,376(,15)
LLGT 15,180(,15)
L    15,20(,15)    Load address of IFAMQRY
CALL (15),(QryParmBlock,rc,rsn)
```

The calling program should invoke SYSSTATE AMODE64(YES) before calling the service.

Parameters

The parameters are explained as follows:

QryParmBlock

Supplied parameter that identifies a query parameter block.

Type: Query parameter block

The query parameter block has the following format and is mapped by the IFAZSYSP macro in SYS1.MACLIB:

Decimal offset	Length	Type	Field	Description
0	4	EBCDIC	ID	ID with value of QRPB
4	2	binary	Length	Total length of the query parameter block
6	1	binary	Unused	Unused; must be zero
7	1	binary	Version	Parameter block version number (X'01')

Decimal offset	Length	Type	Field	Description
8	2	binary	Flags	The following flags are supported: Value Meaning 1 Request return of extended record types. When on, the output area will be mapped by QrPbX_InMemResource_Ext 2 - 15 Unused, must be zero
8	4	binary	Unused	Unused; must be zero
10	2	binary	Unused	Unused; must be zero
12	4	binary	Returned IMRs	Number of returned in-memory resources
16	4	binary	Buffer size	Size of the output buffer that is to contain the information for the returned in-memory resources
20	4	binary	Unused	Unused; must be zero
24	8	binary	Buffer address	Address of the output buffer that is to contain the information for the returned in-memory resources

Each in-memory resource returned in the output buffer (pointed to by the address in the buffer address field) has the following format when extended record types are not requested:

Decimal offset	Length	Type	Field	Description
0	2	binary	Name length	Length of the name
2	26	EBCDIC	Name	Name of the in-memory resource, padded with blanks
28	32	binary	Types	Bit mask (0 - 255 bit array) of SMF record types that are available from this in-memory resource
60	8	binary	Reserved	Reserved

Each in-memory resource returned in the output buffer (pointed to by the address in the buffer address field) has the following format when extended record types are requested:

Decimal offset	Length	Type	Field	Description
0	2	binary	Name length	Length of the name
2	26	EBCDIC	Name	Name of the in-memory resource, padded with blanks
28	4	binary	*	Reserved for alignment

Decimal offset	Length	Type	Field	Description
32	256	binary	Types	Bit mask (0 - 2047 bit array) of SMF record types that are available from this in-memory resource

rc

Returned parameter that identifies the return code from the service.

Type: 4-byte integer

rsn

Returned parameter that identifies the reason code from the service.

Type: 4-byte integer

ABEND codes

IFAMQRY might abnormally end with system completion code X'353'. See [z/OS MVS System Codes](#) for an explanation and programmer response.

Related services

- [“IFAMCON — Connect to an SMF in-memory resource” on page 667](#)
- [“IFAMGET — Obtain data from an SMF in-memory resource” on page 673](#)
- [“IFAMDSC — Disconnect from an SMF in-memory resource” on page 670](#)

Return and reason codes

When the service returns control to the caller, the **rc** parameter contains a hexadecimal return code and the **rsn** parameter contains a hexadecimal reason code, as listed in [Table 130 on page 680](#). The return code and reason code symbols are mapped by the IFARCINM macro.

Table 130. Return and reason codes for the IFAMQRY service		
Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'00' IFAINMRetCodeOK	X'0000' IFAINMRsnCodeOK	Meaning: Successful completion. Action: None.
X'08' IFAINMRetCodeError	X'0801' IFAINMBadMode	Meaning: The caller is running in an incorrect mode for one or more of the following reasons: <ul style="list-style-type: none"> • The caller is not running in task mode. • The caller is in cross-memory mode. • The caller is holding a lock. Action: Change the program to run under a task in PASN = HASN mode with no locks held.
X'08' IFAINMRetCodeError	X'0802' IFAINMBadParmlist	Meaning: The parameter block is not accessible or has an incorrect format. Action: Correct the program to pass a valid parameter block.
X'08' IFAINMRetCodeError	X'0805' IFAINMUnsupported	Meaning: The caller is attempting to pass unsupported options in the parameter block. Action: Initialize all unused fields in the parameter block to zero.

Table 130. Return and reason codes for the IFAMQRY service (continued)

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'08' IFAINMRetCodeError	X'0808' IFAINMNotEnoughQrySp	Meaning: There is not enough space in the output buffer to hold all of the in-memory resources available to the caller. Action: Call the IFAMQRY service with an output buffer that is large enough to contain the data.
X'0C' IFAINMRetCodeEnvErr	X'0C02' IFAINMSMFNotActive	Meaning: SMF is not active. Action: None.
X'0C' IFAINMRetCodeEnvErr	X'0C03' IFAINMObtainFailure	Meaning: The system was unable to obtain storage to generate in-memory data in response to a request. Action: None.
X'10' IFAINMRetCodeFatal	Not applicable	Meaning: Internal error. Action: Contact the IBM Support Center.

Part 11. Cloud Data Access (CDA) Services

The z/OS DFSMSdfp Cloud Data Access (CDA) component provides an environment that other z/OS products may utilize in communicating with Cloud Object Storage.

Cloud Data Access services facilitates access to cloud object storage objects. Developers of new and existing z/OS applications can easily access the objects via the Application Programming Interfaces (APIs) provided by the Cloud Data Access (CDA) services. A method for storing the cloud object storage credentials for later use by the caller of the APIs is also provided.

Cloud object storage provider REST API details are hidden so that the z/OS application can make the same API call, only specifying the name of different cloud provider configuration to utilize different cloud object storage providers.

Chapter 23. Introduction to DFSMSdfp Cloud Data Access (CDA)

Cloud object storage provides a method to access data via RESTful APIs. Different cloud object storage providers utilize different methods for authentication as well as different specifics when using REST APIs for object actions like READ or STORE. Additionally, a user's cloud credentials for a specific cloud object storage provider should be stored securely so that they do not have to enter the credentials every time you communicate with the cloud.

The z/OS DFSMSdfp CDA component is intended to facilitate access to cloud object storage for other z/OS applications. It provides an ISPF Panel application to help with cloud credential storage. It also provides a set of APIs for z/OS programs to invoke, allowing for cloud provider agnostic interaction with the Cloud. The z/OS program does not have to worry about the details of how the object is retrieved, only that an object is retrieved. CDA uses provider json files that describe the details on how to interact with a particular cloud object server. Sample provider files are available for common providers, but they may be modified for your particular environment.

Chapter 24. Cloud Data Access configuration

Some configuration needs to be performed before using a program that utilizes the Cloud Data Access services. The RACF (or equivalent) userid associated with the environment using a program that uses CDA services must have an OMVS segment with a home directory defined. A gdk/ directory must be created in the user's home directory (referred to as ~/gdk/). Permissions for the ~/gdk/ directory should be set so that only the user has read and write authority to the files and directories within.

System administrator configuration quick-start

This quick-start guide shows a high-level view of the steps needed to set up and use Cloud Data Access services:

1. Configure the CSFKEYS general resource class to protect the keylabels for the encryption keys:
 - a. The CSFKEYS general resource class must be active and RACLISTed.
 - b. The ICSF segment of the CSFKEYS class profile CSF-PROTECTED-KEY-TOKEN (or its generic equivalent) must contain SYMPACFWRAP(YES).
 - c. The user's ID must have READ access to the CSF-PROTECTED-KEY-TOKEN profile (or its generic equivalent).
 - d. Define a profile for CSFKEYS resources beginning with GDK.** with a universal access (UACC) of NONE along with ICSF(SYMPACFWRAP(YES) SYMPACFRET(YES)).
 - e. The user's ID must have READ access to the new CSFKEYS profile for resources beginning with GDK.<userid>** along with ICSF(SYMPACFWRAP(YES)SYMPACFRET(YES)).
 - f. The security administrator or person who will be entering the cloud provider keys must have UPDATE access to the new CSFKEYS profile for resources beginning with GDK.<userid>.**

Example of z/OS Security Server RACF commands for keylabel protection:

```
/* Define a generic label with UACC(NONE) so default access is NONE */
RDEFINE CSFKEYS GDK.** UACC(NONE) ICSF(SYMPACFWRAP(YES) SYMPACFRET(YES))

/* Define a generic label specific to the CDAUSER */
RDEFINE CSFKEYS GDK.CDAUSER.** UACC(NONE) ICSF(SYMPACFWRAP(YES)
SYMPACFRET(YES))

/* Permit the CDAUSER to their keylabels */
PERMIT GDK.CDAUSER.** CLASS(CSFKEYS) ID(CDAUSER) ACCESS(UPDATE)

SETROPTS RACLIST(CSFKEYS) CLASSACT(CSFKEYS) REFRESH
```

2. Ensure access to the required ICSF entry points. The user must have at least READ authority to the following CSFSERV Class resources:
 - CSFKGN
 - CSFRNGL
 - CSFKRD
 - CSFKRC2
 - CSFOWH
3. CDA uses HTTPS connections with the remote Cloud Object Storage server. The System SSL processing performed may require some setup. For more information, see [RACF CSFSERV resource requirements in z/OS Cryptographic Services System SSL Programming](#).
4. Configure CDA for system general use:

- a. Copy IBM COS . json from /usr/lpp/dfsms/gdk/samples/providers/ to /usr/lpp/dfsms/gdk/providers/ (see the note in “Provider file” on page 692). Permissions should be set to 644.
 - b. Rename IBM COS . json to any name of 20 character or fewer, keeping the . json suffix.
 - c. Modify the <provider>. json file to suit the Cloud Object Storage server that you will use. This information should come from the administrator of the Cloud Object Storage server. For more details about individual key/value pairs, see “Provider file” on page 692.
 - d. Change the host value to be the URL for the cloud provider server.
 - e. Change the port value if necessary.
 - f. Change the region value if necessary.
 - g. Change the sslCipher value if the cloud provider server uses other SSL Ciphers.
5. Configure a RACF key ring for the cloud object store for z/OS as a client for the TLS/SSL traffic. You may create a virtual key ring. For more information, see [RACF and key rings in z/OS Security Server RACF Security Administrator's Guide](#).
- a. Obtain the root CA certificate of the target cloud object server. For example, using a browser, enter the cloud server URL, and click on the lock icon to download the root CA certificate to a local PC, followed by transfer of the certificate to a data set on z/OS.
Make sure you trust the root CA.
 - b. Use RACDCERT ADD to add the root CA certificate of the cloud provider server under CERTAUTH so that it is considered to be in the virtual key ring of CERTAUTH.
 - c. Ensure that users of the CDA services have READ access to the virtual key ring where the certificates are stored. Only secure (HTTPS) connections are supported.
 - d. If the virtual key ring name is not *AUTH*/, then update the value of the provider file sslKey to be the name of the virtual key ring used.

Example of z/OS Security Server RACF commands to set up virtual key ring for the cloud object store client:

```
/* Define the following profile in the FACILITY class if it does
   not exist yet. */
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(READ)
/* Add the Root CA certificate of the remote Cloud Server under CERTAUTH.
   It is stored in the HLQ.ROOTCA.CLOUD data set. */
RACDCERT CERTAUTH +
          ADD('HLQ.ROOTCA.CLOUD') +
          WITHLABEL('CLOUD') TRUST
/* Refresh */
SETROPTS RACLIST (DIGTCERT) REFRESH
/* Make sure the certificate was added correctly */
RACDCERT CERTAUTH LIST(LABEL('CLOUD'))
```

6. CDA Authorization Panels: Ensure that SYS1.DFQPLIB is part of the ISPPLIB concatenation or that the following members in SYS1.DFQPLIB, are added to an ISPPLIB library:
- GDKAPPOP
 - GDKAUTHK
 - GDKAUTHL
 - GDKAUTHP
 - GDKMAINP
 - GDKOBJAC
 - GDKOBJAL

A RACF (or equivalent) profile should be created to ensure only authorized users have access to these members.

7. Users of CDA services require OMVS home directories.

- a. The users RACF ID must have an OMVS segment defining the home directory.

User configuration quick-start

1. Configure CDA for the user's environment:
 - a. Create ~/gdk/ in the user's UNIX home directory. Change the permissions to 700.
 - b. Create ~/gdk/providers/ in the user's UNIX home directory. Change the permissions to 700.
 - c. Copy /usr/lpp/dfsms/gdk/samples/gdkkeyf.json to ~/gdk/gdkkeyf.json . Change the permissions to 600.
 - d. The user may have a local config file in their ~/gdk/ directory. Copy /usr/lpp/dfsms/gdk/samples/gdkconfig.json to ~/gdk/config.json . Change the permissions to 600.
 - e. The user may have a local provider file in their ~/gdk/providers/ directory if they want to use a modified provider definition. Copy /usr/lpp/dfsms/gdk/providers/*.json to ~/gdk/providers/ and change the permissions to 600.
2. Use the CDA authorization panel to store the user's cloud credentials. For a detailed description and walkthrough, see [Chapter 27, “Cloud Data Access cloud credential storage,” on page 701](#).

```
EX 'SYS1.SAXREXEC(GDKAUTHP) '
```

- a. Enter the number of the cloud provider you are entering the credentials for.
- b. Enter **O** on the Option prompt to open the Credential Entry Panel.
- c. Enter the Access Key (or userID) for the cloud object storage server.
- d. Enter the Secret Access Key (or password) for the cloud object storage server.
- e. Enter **S** on the Option prompt to save the credentials.

Chapter 25. Cloud Data Access files

DFSMSdfp CDA utilizes these files during its processing:

key file

Contains the encrypted cloud credentials for the user.

config file

Contains some configuration used during the saving of the cloud credentials.

provider file

Contains configuration specific to a specific Cloud Object Store provider.

Key file

The cloud credentials to be used will be stored in a key file named gdkkeyf.json in the ~/gdk/directory. Before using the Cloud Data Access authorization panel described in Chapter 27, “Cloud Data Access cloud credential storage,” on page 701, the /usr/lpp/dfsms/gdk/samples/gdkkeyf.json file should be copied to the user's ~/gdk/ directory. If a security administrator is using the z/OS Cloud Data Access authorization utility to store the cloud credentials on behalf of the user, then the security administrator should also have read/write permission to the ~/gdk/gdkkeyf.json file.

Config file

The config file contains some configuration key:value pairs that the CDA services may use during processing. However, some programs that use the CDA services may request that the config file not be used during processing.

The system administrator may copy the sample config file from /usr/lpp/dfsms/gdk/samples/gdkconfig.json to /usr/lpp/dfsms/gdk/config.json. If not explicitly told to not read the config file, the CDA services will first look in the user's directory for a config file (~/gdk/config.json), and if not found, it will look in the CDA global directory /usr/lpp/dfsms/gdk/config.json.

A user may want to copy the /usr/lpp/dfsms/gdk/samples/gdkconfig.json file to their home directory ~/gdk/config.json so it can be modified for additional logging or debugging of processing.

Possible key:value pairs:

log-level

During CDA services processing different levels of logging are allowed:

ERROR

Default: Only Error messages are logged.

WARNING

Warning messages and above are logged.

NOTICE

Notice messages and above are logged.

INFO

Informational messages and above are logged.

DEBUG

Flow and processing messages are logged.

NONE

No logging is performed by the CDA services.

web-toolkit-logging

Request logging from the z/OS Client Web Enablement Toolkit.

true
Log messages

false
Do not log messages.

translation

Whether to request conversion of text data during READ or WRITE.

true
Convert EBCDIC to ASCII on WRITE to Cloud or ASCII to EBCDIC on READ from Cloud.

Default: false
Do not convert data.

allow-no-CEX

If no Crypto Express card is available on the system, then the user accepts the reduced security of having the encryption keys stored in the clear in the ICSF CKDS.

true

Default: false

Provider file

The provider file is a JSON object made up of key names and values. A sample provider file should be modified for the particular Cloud Object Storage provider available to the z/OS LPAR. Usually only the host and region need to be modified, and possibly others depending on the needs of the environment. What follows is a list of the recognized key names and their descriptions. The GDKVALD API may be used to retrieve the values associated with key names, allowing applications to retrieve their own application-specific key:value pairs. The following list of key names as well as any prefixed by "cda_" are reserved for CDA usage.

The specifics of how CDA services should interact with a Cloud Object Storage provider are detailed in a provider file. Each Cloud Object Storage provider that is used on the system should be placed in a different provider file. Each provider file has a .json suffix, and is stored in either the users home directory (~/gdk/providers/), or in the CDA default location /usr/lpp/dfsms/gdk/providers/. CDA services will first examine the user's ~/gdk/providers/ directory for a provider file ending in .json. If it no ~/gdk/providers/ directory is found, then /usr/lpp/dfsms/gdk/providers/ is searched for the provider file. The maximum length of the provider name is 20 characters, not including the .json suffix. The provider names are case-sensitive, so the name that is passed to the CDA program must match the name found in the gdk/ providers directory.

The system administrator may modify and place provider files for the Cloud Object Storage providers that can be used into the /usr/lpp/dfsms/gdk/providers/ directory. The permissions of these files should be read only for users (rw-r--r-- or chmod 644).

Note:

The files in the /usr/lpp/dfsms/gdk/providers directory are intended to be default versions that are tailored specifically to your Cloud Object Storage provider. Users can copy the tailored version to their personal directory if they want to make modifications.

Many sites copy the zFS that contains /usr/lpp/dfsms/gdk/providers across to each system in the sysplex, thus losing any tailored files placed in /usr/lpp/dfsms/gdk/providers. To avoid this, you can create a zFS data set and mount it to the /usr/lpp/dfsms/gdk/providers directory, placing the specifics in the SYS1.PARMLIB(BPXPRMxx) member so it is automatically mounted during IPL.

Sample provider files can be found in /usr/lpp/dfsms/gdk/samples/providers/.

IBMCOS.json

Describes an IBM Cloud Object Storage provider.

S3CLOUD.json

Describes an AWS S3 Cloud Object Storage provider.

A provider file is a JSON object. The provider file should be modified for the particular Cloud Object Storage provider available to the z/OS LPAR. Usually only the host and region need to be modified, and possibly others depending on the needs of the environment.

name

Descriptive name for this cloud provider (not used by CDA service).

host

URI for the Cloud Object Storage provider. May also be a JSON object with a "preferred" key and optional "backup" key that specifies a list of URI values for the Cloud Object Storage provider.

preferred

A list of string values with the format: {https://}<uri>{:<port>},{<sslKeyRingName>}. Values surrounding by curly braces {} are optional. The string value may contain the uri, port number, and sslKey ring name, but only the uri is a required value. When connecting to the host, the host uri entries are attempted to be used, chosen in random order, until one is successfully connected.

backup

A list of string values with the same format as the preferred list. The entries in this list are only used when none of the entries in the preferred list can be used.

cloudserverTZ

Allows override of the default TZ=GMT as the environment variable for what time zone the current time should be used when calculating the AWS4 signature.

encodeobjname

Specifies a string of characters to url-encode when found in the requested object name. For example, z/OS data set names can contain the national characters of #\$. The pound-sign/hash character (#) is a special character for object names, and would need to be url-encoded in order to be used in an object name.

localIPAddr

Specifies a value to set on the z/OS Client for Web Enablement Toolkit option, HWTH_OPT_LOCALIPADDR.

localPort

Specifies a number value to use as the originating port number in conjunction with the localIPAddr value. Sets the HWTH_OPT_LOCALPORT option.

region

Region identifier such as us-west-1.

port

HTTPS port number used (if not default 443).

httpMechanism

Must be set to HTTPS.

sendTimeout

Optional: Length of time in seconds for send request to timeout.

receive Timeout

Optional: Length of time in seconds for a receive to timeout.

IPStack

Optional: Name of alternate z/OS IP Stack to use in HTTP communication.

sslCertCheckWarn

Optional: If "true", will log WARNING with corresponding diagnostic area reason code in case the URI of target host doesn't match the server identity in the SSL certificate.

Recommendation: Add this key value pair to your provider file if you encounter a situation where your SSL certificate is incorrectly configured and you need to use this SSL certificate temporarily.

sslVersion

Type of SSL to use:

- TLSV10

- TLSV11
- TLSV12
- TLSV13

sslCiphers

Override the SSL ciphers to be used in the communication (not required unless the HTTPS connection uses other than the default). For more information, see [Cipher suite definitions](#) in *z/OS Cryptographic Services System SSL Programming*.

sslKey

Override the default virtual key ring name of *AUTH*/*.

encode

Optional: What type of URL-encoding to use.

special

Use AWS URL-encoding for object names.

encodeUrlChars

Characters in the object name that should be URL-encoded by CDA.

errorUrlChars

Characters in the object name that should result in a failure, if specified.

Note: The backslash character is a special JSON character, and must be escaped by a backslash character if you want to specify it.

objectNameConvention

Optional. Can be **file** or **object**.

file

File indicates that the passed name follows file system naming conventions where characters between forward-slash characters are directory names, and if the end of the passed name does not end in a forward-slash, then the last portion is a file name.

object

Object is the default when not specified and indicates that the passed name conforms to object name conventions, which consist of a bucket name starting with a forward slash, and optionally ending with another forward slash. After the first two forward slash characters, all subsequent characters are considered to be the object name.

calculateUploadSize

Optional. Can be set to **true**. Any other value is ignored. This field is used to indicate if the file size must be calculated, before proceeding with the operation during a GDKGEN API call. The name of a z/OS data set or UNIX file must have been passed. The calculated size will be saved in the CDA variable store variable: **GDK_UPLOAD_SIZE**.

authentication

model: AWS4

Use AWS4 authentication model when communicating with the Cloud Object Storage provider.

model: Azure

Use Microsoft Azure authentication when communicating with the Cloud Object Storage provider.

auto method: SharedKey | SharedKeyLite

When authenticating with a Microsoft Azure service, the authentication method may be selected. There are two options available: **SharedKeyLite**, which is the default when the authMethod key is not specified, and **SharedKey**.

model: OAUTH_2

The Cloud Object Storage provider uses the OAUTH_2 authentication method. The saved cloud credentials will be used to retrieve an Access Token that is used on the requests.

model: NONE

The Cloud Object Storage provider doesn't use authentication.

model: KEYSTONE

The Cloud Object Storage provider expects Keystone v3 authentication.

model: BASIC

The Cloud Object Storage provider uses the BASIC authentication method.

model: TEMPAUTH

The Cloud Object Storage provider expects authentication using TEMPAUTH.

credential_schema

Specifies a schema used to parse the credentials JSON file containing information to build a JSON Web Token (JWT).

JWT_Duration

Specifies the length of time to request validity of the JWT in minutes.

JWT_claims

A JSON object containing the claims used to build the JWT.

supportedOperations

Array of operations objects. **name** is one of { GETOBJECT, GETLARGEOBJECT, WRITEOBJECT, WRITELARGEOBJECT, LISTOBJECT, DELETEOBJECT, LISTBUCKETS, HEADOBJECT, CREATEBUCKET, and DELETEBUCKET }.

Additional operations may be added to the supportedOperations array with unique names to be used with the GDKGEN API.

name

GETOBJECT

apiEndpoint

How to build URL for request.

```
<HOST><GDK_OBJECT_NAME>
```

httpMethod

```
{ GET, PUT, POST, DELETE, HEAD }
```

multipartChunksize

The size of each chunk in bytes to request when retrieving an object in a GETLARGEOBJECT operation, or sending an object in the WRITELARGEOBJECT operation. The default size is 8388608 bytes. This size will be used when a GDKWRITE API caller is using GDK_EXIT_DATALOCATION, and did not specify the Content-Length or Content-LengthE optional parameters.

multipartThreshold

The size in bytes that marks the threshold whether to perform a multipart upload during a WRITELARGEOBJECT operation. z/OS UNIX files or data sets smaller than this size will not use a multipart upload operation. Additionally this applies when using GDK_EXIT_DATALOCATION for a GDKWRITE API call when the Content-Length or Content-LengthE optional parameters are not specified.

actions

An array of operation objects that describe how to perform a multi-step request such as GETLARGEOBJECT, or WRITELARGEOBJECT.

requestParameters

JSON object describing parameters on the request.

mechanism

Details about how to build parts of request.

HEADER

Specifics about what headers need to be included (can be multiple).

MESSAGE_BODY

Request body content.

METAHEADER

Describes how to construct a metadata header.

The "descriptor" key value contains the header prefix for metadata headers that are built as part of the metadata optional parameter processing.

URL_PARM

Add url parameters to the url before signing and issuance of the HTTP command. The url will have the format of: <url>?<descriptor>&<descriptor>

descriptor

How to build the mechanism.

- "some data" passed as-is unless <text> is found such as DATE_ISO_8601.
- **<GDK_DATA>**
The data passed into CDA services.
- **<GDK_PREFIX>**
The value passed on the "prefix" optional parameter.
- **<GDK_DELIMITER>**
The value passed on the "delimiter" optional parameter.
- **<GDK_MARKER>**
The value passed on the "marker" optional parameter.
- **<GDK_VERSION_ID>**
The value passed on the "versionID" optional parameter.

contentType

How to set content type for HTTP request.

- text/plain
- application/xml
- application/octet-stream

signedS3Payload

Parameter to enable S3 signed payload

Allowed values <true/false>

responseResults

Allows definition of how to parse the response from the request.

mechanism

HEADER to indicate this is a rule to be used when processing response headers, or MESSAGE_BODY to indicate this is the rule to be used when processing the response body.

name

Used in a mechanism: HEADER rule. The value is matched with the HTTP header name.

content

The value specifies a CDA variable that the value from the header is stored under for later retrieval.

contentType

Specifies the type of content being returned from the object server. Common options are: application/xml, application/json, text/plain

schema

A JSON object that describes a schema used to interpret the content of the response body.

b64_urlEncodeGDKPART

"true" indicates that the multipart part numbers need to be base64 url encoded.

elementName

The XML tag or JSON key name to match.

xmlns

The XML namespace value to add to the XML document being built as the request body.

entries

Descriptor of an array of XML entries.

type

Indicator of the type of value. "object", "string", "array", and "number" are possible values.

item

Describes one entry in the "entries" array. This may be a single JSON object that describes the individual fields for an entry, such as name, modified time, or size. It may also be a JSON array of JSON objects that describes the individual fields for an entry with a particular name.

modifiedTimeFormatP

Describes how to parse a returned inject modified time value according to the strftime() XLC runtime function.

createdTimeFormatP

Describes how to parse a returned object created time value according to the strftime() XLC runtime function.

A GETLARGEOBJECT operation is optional, but extends the functionality described by a GETOBJECT operation. It is expected to be made up of a name key:value pair, and an actions array. The actions array must have the following objects:

name: getSize - Description on how to retrieve the size of a single Cloud Object.

name: data - Description how to retrieve the data.

The data action may have an optional multipartChunksize: "number" that overrides the default 8MB size for each retrieve that is used to retrieve the entire cloud object.

A WRITELARGEOBJECT operation is optional, but extends the functionality described by a WRITEOBJECT operation. It is expected to be made up of a name: WRITELARGEOBJECT pair and an actions array. The actions array consists of the following objects.

name: "init"

Optional: Describes the action to take to initialize a multipart upload sequence.

name: "data"

Required: Describes the action to upload a part of the data to be sent. The chunk size for each part is 8MB, except for the last.

name: "complete"

Optional: Describes the action to complete a multipart upload sequence.

name: "error"

Optional: Describes the action needed to abort a multipart upload so that individual parts do not take up space when an error occurred, and the multipart upload was unable to complete successfully.

CDA uses some variables during its processing. Many of the variables can be specified in the provider file. The variable name and description, along with how it is used, is described in [Table 131 on page 697](#).

Table 131. DFSMSdfp CDA variables	
CDA Variable Name	Description
GDK_DATA	Data for operation (varies depending on dataLocationType).
GDK_DATA_LEN	Length of data in buffer for WRITE type operations.
DATE_ISO_8601	GMT current timestamp in ISO 8601 format.
AZURE_ACCOUNT	Account name from saved Cloud Credentials.

Table 131. DFSMSdfp CDA variables (continued)

CDA Variable Name	Description
DATE_GMT	GMT current timestamp.
GDK_OBJECT_NAME	The remote object name is referenced as a whole (bucket name and object name).
GDK_BUCKET	Bucket portion of the GDK_OBJECT_NAME (first forward slash through second forward slash).
GDK_OBJECT_PART	Object name portion of GDK_OBJECT_NAME (all text following second forward slash).
GDK_PART	Part number from a multipart upload.
GDK_GMT_UNIX_TIME	GMT current timestamp in number of seconds since January 1, 1970.
GDK_JWT_EXP	JWT Expiration time calculated from the GDK_GMT_UNIX_TIME + JWT_JWT_duration value from provider file.
HOST	Host value from Provider file "host" key.
PARAMETER_SET	Used internally by DFSMSdfp CDA for collecting requestParameters from the provider file.
UPLOADID	Used internally by DFSMSdfp CDA for tracking the Upload ID during a Multi-Part upload.
GDK_ETAG	Used internally by DFSMSdfp CDA for tracking the returned eTag during a Multi-Part upload.
GDK_PREFIX	Used during a LIST request to specify the prefix of objects to return.
GDK_DELIMITER	Used during a LIST request to specify the delimiter that is used to identify objects.
GDK_LENGTH	Used during a GETLARGEOBJECT request to hold the total amount of data needed to be retrieved.
CLIENT_ID	Used during OAUTH_2 authentication and pulled from the Credentials file. Usually equivalent to the client_email.
ACCESS_TOKEN	Returned Access Token resulting from authenticating with the cloud server.

Chapter 26. Cloud Data Access (CDA) API basics

Elements of Cloud Data Access

The Cloud Data Access (CDA) APIs are organized into single API calls to perform a specific task, such as GET or WRITE an object. All of the HTTP requests needed to perform that task are performed by the Cloud Data Access API. The API calls depend on a configuration file for the requested cloud provider and an entry in the credentials file for the cloud credentials to be used.

Supported APIs allow an application to:

- Read an object from Cloud storage
- Write an object to Cloud storage
- Delete an object from Cloud storage
- List an object/objects in Cloud storage
- Retrieve a list of known Cloud providers
- Store Cloud Credentials
- Retrieve Cloud Credentials
- Delete Cloud Credentials
- Retrieve a list of resources for Cloud Credentials
- Translate a return code into message text
- Start a CDA Session that will contain 0 or more related API calls.
- Stop a CDA Session and perform cleanup of the session.
- Perform a named operation from the provider file.
- Validate a provider file and retrieve values from that provider file.
- Query availability of CDA functions.
- Compress an object before writing to cloud storage.
- Decompress an object before downloading it to z/OS.
- Upload a z/OS data set, imbedding record prefixes in between data records.
- Download an object with imbedded record prefixes, writing data records to the z/OS data set.
- Create a z/OS data set based on metadata tags from the object.

Chapter 27. Cloud Data Access cloud credential storage

ISPF panels are provided to allow a user or security administrator to enter the cloud credentials for the user that will be using invoking the program calling the Cloud Data Access (CDA) APIs. SYS1.SAXREXEC(GDKAUTHP) will display the z/OS Cloud Data Access Authorization Utility ISPF panel. From this utility you can save the cloud credentials for a cloud provider that will be used when a z/OS USERID runs an application that calls the CDA APIs.

The Cloud Credentials will be encrypted and saved in a credentials file in the z/OS USERID's UNIX System Services home directory.

For the CDA services to perform its communication with the cloud object storage server, it needs to use the cloud credentials of the user. Additionally, programs using the CDA services do not want to ask the user to enter the cloud credentials every time they use the CDA services. Therefore, the cloud credentials need to be available to the CDA services in a secure format.

The cloud credentials that will be used by the program, when executed by the user, are entered using the z/OS Cloud Data Access authorization utility. The cloud credentials are encrypted using a random key that is stored in the IBM Cryptographic Service Facility (ICSF) CKDS. The encrypted cloud credentials are written to the user's gdkkeyf.json file in the ~/gdk/ directory.

The encryption key is stored in ICSF under a keylabel of the form GDK.<userid>.<provider>.Annnn . Where <userid> is the SAF userID for the user and <provider> is the cloud provider that the cloud credentials are associated with. A suffix, Annnnn, where nnnnn is a 5 digit number used internally by CDA, is appended. The keylabel is protected by the CSFKEYS general resource class, which must have SYMCPACFWRAP(YES) in the ICSF segment. The userID must have READ access to the keylabel profile.

GDK.<userid>.*

If a security administrator is using the z/OS Cloud Data Access authorization utility to store the cloud credentials on the behalf of the user, then the security administrator should also have read/write permission to the ~/gdk/gdkkeyf.json file.

Note:

To ensure the highest level of security of the encryption key, the system should have a crypto express card installed so that the encryption key stored in the ICSF CKDS is wrapped by the master key of the crypto express card.

If no card is available, the config sample file may be copied from /usr/lpp/dfsms/gdk/samples/gdkconfig.json and placed in the user's home directory as ~/gdk/config.json . The key:value pair of "allow-no-CEX":true can be added to the config.jsonfile to indicate that the user accepts the reduced security of the encrypted cloud credentials when no crypto xpress card is available for use. When no crypto express card is available, and the "allow-no-CEX":true key:value pair exists, this indicates that the user accepts the data encryption key for the encrypted cloud credentials being stored in the clear in the ICSF Cryptographic Key Data Set (CKDS).

Ensure that the security administrator or user who will be entering the cloud provider keys has sufficient authority to write to the user's gdkkeyf.json file (~/gdk/gdkkeyf.json) and UPDATE permission to the CSFKEYS profile for resources beginning with GDK.<userid>.* .

The process of saving the cloud credentials entails encrypting the cloud credentials using ICSF services. For more information on CCA and ICSF entry points, see [*z/OS Cryptographic Services ICSF Administrator's Guide*](#).

The user must have at least READ authority to the following CSFSERV class resources:

- CSFKGN
- CSFRNGL

- CSFKRD
- CSFKRC2
- CSFOWH

The CDA authorization utility may be invoked from the ISPF Command Shell using the following command:

```
EX 'SYS1.SAXREXEC(GDKAUTHP)'
```

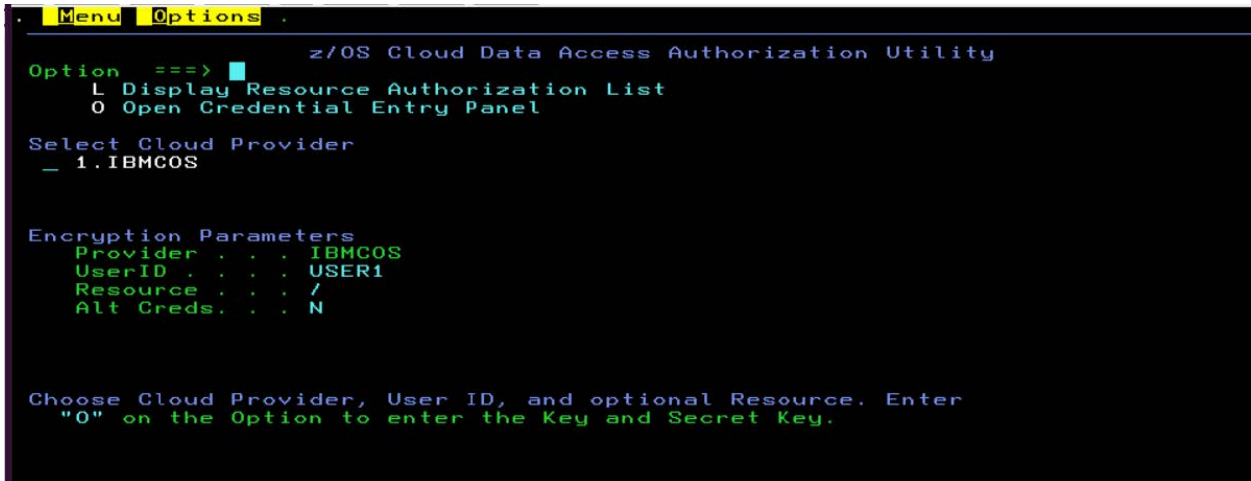


Figure 41. Invoking the Cloud Data Access authorization utility from the ISPF command shell

This will start a CDA panel where the cloud credentials will be encrypted and saved.

1. A list of cloud provider names are displayed. Any unique names ending in .json that are found in either the user's ~/gdk/providers directory or the default /usr/lpp/dfsms/gdk/provider directory are displayed, without the .json extension.
2. Enter the RACF (or equivalent) userID that will be using the CDA cloud object utility into the **UserID** field under **Encryption Parameters**.
3. Select the cloud provider associated with the key pair being added by entering the associated number under **Select Cloud Provider**. The currently chosen provider will be displayed in **Provider** under **Encryption Parameters**.
4. If this key pair is intended to be used with a specific bucket, enter / followed by the bucket name in **Resource** under **Encryption Parameters**. Otherwise, simply enter a / to indicate that this key pair is valid for any bucket associated with this cloud provider.

Generally, the keys for accessing objects in specific buckets are associated with the / resource. Keys for accessing objects in specific buckets that require different credentials can be added and associated with that **/bucket_name**. CDA services will attempt to utilize specific keys tied to buckets before utilizing the generic key for the provider.

Note: Only 1 generic key is used per provider. If a second generic key is entered, it will overwrite the first.

5. If this key pair is intended to be used as the alternate credentials by a user of the CDA APIs, enter **Y** for **Alt Creds**.
6. Enter **O** for **Option** to continue to the next panel.

```

. Menu Options .
z/OS Cloud Data Access Authorization Utility
Option ==> S S Save Resource Authorization C Clear Secret Key Field
              (for hidden input)
Encryption Parameters
Provider . . . IBMCOS
KeyLabel . . . GDK.USER1.IBMCOS
Keystore . . . /u/user1/gdk/gdkkeyf.json
Resource . . . /
Authorization Parameters
Key . . . . . My_access_key
Secret Key . . *****
Enter the Key and Secret Key used to access the specified Cloud Provider.

```

Figure 42. Cloud Data Access authorization utility Options Menu

7. Enter the Key and Secret key values into the associated fields under **Authorization Parameters** and then press Enter. The characters are not echoed to the screen and are displayed as * after hitting enter.
8. Enter **S** on the top Option line to encrypt and save the key pair.

Note:

The first time this panel is executed, the user may receive the following warning messages:

```

ERROR: getpwnam() error: EDC5121I Invalid argument.
ERROR: getpwnam() error: EDC5129I No such file or directory.

```

This behavior is expected because the UserID field has not yet been populated. Once the UserID field is at least specified once, the warning messages will no longer be displayed.

When entering alternate credentials, the keylabel is additionally appended with **.ALT**, and an informational line of entering alternate credentials is displayed.

```

. Menu Options .
z/OS Cloud Data Access Authorization Utility
Option ==> S S Save Resource Authorization C Clear Secret Key Field
              (for hidden input)
Encryption Parameters
Provider . . . IBMCOS
KeyLabel . . . GDK.USER1.IBMCOS.ALT
Keystore . . . /u/user1/gdk/gdkkeyf.json
Resource . . . /
Entering Alternate Credentials
Authorization Parameters
Key . . . . . My_alternate_access_key
Secret Key . . *****
Enter the Key and Secret Key used to access the specified Cloud Provider.

```

Figure 43. Cloud Data Access authorization utility entering alternate credentials

Error conditions

If an error occurs during the saving of the cloud credentials, an error message will be written indicating that it was unable to store the cloud credentials indicating the IBM Cryptographic Service Facility (ICSF) API that encountered the error along with the return code and reason code returned by the service. For more information, see [ICSF and cryptographic coprocessor return and reason codes in z/OS Cryptographic Services ICSF Application Programmer's Guide](#).

Table 132. ICSF reason codes

ICSF reason code	Description
0BFB	Received when the ICSF segment of the CSFKEYS class does not have SYMCPACFWRAP(YES).
3E80	Permission to the required CSFSERV Class is not READ or better.
3E84	Received when the RACF userid does not have permission to the necessary keylabel in the CSFKEYS class needed to store the encryption key for the encrypted cloud credentials.
271C	Received when the ICSF keylabel can't be found. Usually the keylabel was mistakenly deleted from the ICSF CKDS. The Cloud Credentials must be saved again.

Credential panels for different authentication types

On z/OS 3.1 and above, the Credential Entry Panel displayed will vary depending on the authentication model specified in the provider file that is chosen.

AWS4

When you choose a provider that specifies AWS4 as the authentication model, the Credential Entry panel displays Key and Secret Key for entry into the panel. Enter the AWS4 HMAC Access Key and Secret Access Key obtained from the Security Administrator for the Object Storage account.

AZURE

When you choose a provider that specifies AZURE as the authentication model, the Credential Entry panel displays Storage account name and Access Key. Enter the credentials as specified from the account holder.

OAUTH_2

When you choose a provider that specifies OAUTH_2 as the authentication model, the first panel displayed allows a choice of the type of credentials to be entered. The type of credentials are:

1. Access Token

Some cloud providers may give you an Access Token that can be used when authenticating with the service. This access token is used directly in the Authorization: Bearer header.

2. Credentials File

Some cloud providers will allow you to create a Service Account to connect with their object store. When credentials are created for the service account they can be delivered in a JSON format with:

a. client id

An identifier for the service account, like a client email.

b. private_key

A private RSA key that is used to sign the JSON Web Token (JWT) which is used to authenticate to the service, resulting in a short-lived Access Token.

Note: A Crypto Express card is required when using a Credentials file. This is because the Private RSA Key needs to be stored in the ICSF Private Key Data Set (PKDS). For more information about required hardware for CSNDPKI, see [PKA Key Import \(CSNDPKI and CSNFPKI\)](#) in *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Once you select the type of credentials, a subsequent panel allowing entry of the Access Token, or the absolute path to the Credentials JSON file, is displayed.

TEMPAUTH

When you choose a provider that specifies TEMPAUTH as the authentication model, the Credential Entry panel displays Userid and Password, along with Access URL. Enter the Userid and Password, along with the Access URL needed to authenticate with the cloud server.

BASIC

When you choose a provider that specifies BASIC as the authentication model, the Credential Entry panel displays Userid and Password. Enter the Userid and Password that you want to use to authenticate with the cloud server.

KEYSTONE

When you choose a provider that specifies KEYSTONE as the authentication model, the Credential Entry panel displays Userid, Password, and Tenant. Access URL is also displayed. Enter the Userid, Password, and Tenant for your credentials. If you have an access URL, it may be entered here as well.

Chapter 28. Availability of the Cloud Data Access callable services

The Cloud Data Access services depend on the availability of the z/OS Client Web Enablement Toolkit services as well as ICSF services.

Chapter 29. Syntax, linkage, and programming considerations

The z/OS Cloud Data Access services are available to many programs running in various address spaces. Many z/OS execution environments are supported, as well as various programming languages.

Programming interface files provided by the Cloud Data Access (CDA) services

Program language Programming interface file

Table 133 on page 709 lists the programming interface files provided by the z/OS Cloud Data Access parser.

Table 133. CDA parser programming interface	
Programming language	Programming interface file
C / C++	Include file GDKIC provided in SYS1.SIEAHDV.H.
Assembler	Include file GDKKASM provided in SYS1.MACLIB.
COBOL	Include file GDKICOB provided in SYS1.MACLIB.

Calling formats

Table 134 on page 709 lists specific calling formats for languages that can invoke the z/OS Cloud Data Access (CDA) parser callable services.

Table 134. Calling formats for the z/OS Cloud Data Access parser callable services	
Programming language	Calling format
C / C++	<code>gdkapi_service_name (return_code,param1,param2,...)</code> where the <code>gdkapi_service_name</code> is all lower case
Assembler	<code>GDKapiservice_name (return_code,param1,param2,...)</code>

Linkage considerations

There are two ways for a compiled application to find the z/OS Cloud Data Access callable services:

Linkage stub method

(Recommended) Use the linkable stub routine GDKCSS from SYS1.CSSLIB to bind your object code. GDKCSS assumes a Language Environment already exists. If you want CDA to create and tear down a Language Environment for you, use GDKCSSNL to bind with your object code. If you attempt to use the z/OS Cloud Data Access services on a previous release of z/OS that does not support the z/OS Cloud Data Access Services, this method results in the service call receiving a return code of X 'F02 ' (GDKK_UNSUPPORTED_RELEASE).

Dynamic Link Library

(Recommended for AMODE31 C language callers) Use the GDKDLL31 side file found in SYS1.SIEASID when binding your object code. GDKDLL31 assumes that you are already in a Language Environment in AMODE 31, and the calling language is XLC. At runtime, the GDKDLL31 dynamic load library found in SYS1.SIEALNKE will be found and used by the appropriate API. When Binding your C code into a program, ensure that DYNAM=DLL is specified as a IEWL parameter.

(Recommended for AMODE64 C language callers) Use the GDKDLL64 side file found in SYS1.SIEASID when binding your object code. GDKDLL64 assumes that you are already in a Language Environment

in AMODE 64, and the calling language is XLC. At runtime, the GDKDLL64 dynamic load library found in SYS1.SIEALNKE will be found and used by the appropriate API. When Binding your C code into a program, ensure that DYNAM=DLL and AMODE=64 is specified as a IEWL parameter.

Direct linkage method

Code the linkage to the z/OS DFSMSdfp Cloud Data Access services directly. This can be done if the program first confirms that the level of z/OS contains the CDA APIs.

The macros that define those names are:

- SYS1.MACLIB(CVT)
- SYS1.MACLIB(IHADFA)

The following example shows the assembler linkage:

```
L      R05,CVTPTR
L      R10,CVTDFA(,R05)
L      R12,DFADFVAD(,R10)
L      R05,240(,R12)
L      @15,8+4*GDK_serv_XXXXX(,R05)
BASR   @14,@15
```

In the example, XXXXX represents the last five letters of the service you want to call. This requires that the GDKKASM assembler macro be included. If you attempt to call the Cloud Data Access APIs on a previous release of z/OS that does not support the APIs, this method may result in the application receiving an abend X'019'.

If using the Direct Linkage method, some APIs require an existing Language Environment. To facilitate better performance when using the I/O APIs via the Direct Linkage method, it is assumed that a Language Environment exists. It is also assumed that one will not be created or destroyed upon entry or exit to the API. A Language Environment preinitialization environment can be used (PIPI) for this requirement.

API Entry Offset from DFVCDAVT	API Name	Language Environment Requirement
12 (C)	GDKDEL	YES
16 (10)	GDKGET	YES
20 (14)	GDKGETP	YES
24 (18)	GDKLIST	YES
28 (1C)	GDKWRITE	YES
32 (20)	GDKKEYAD	NO
36 (24)	GDKKEYDL	NO
40 (28)	GDKKEYGR	NO
44 (2C)	GDKKEYSR	NO
48 (30)	GDKMSGTR	NO
52 (34)	GDKGEN	YES
56 (38)	GDKGENN	NO
60 (3C)	GDKINIT	YES
64 (40)	*	*
68 (44)	GDKTERM	YES
72 (48)	*	*

API Entry Offset from DFVCDAVT	API Name	Language Environment Requirement
76 (4C)	GDKVALD	YES
80 (50)	GDKVALDN	NO
84 (54)	GDKQUERY	YES
88 (58)	GDKQRYN	NO
92 (5C)	GDKMSGTL	YES
96 (60)	GDKLISTL	YES
100 (64)	GDKKYSRL	YES

Linkage considerations for high-level language programming

Callers must ensure that the proper linkage is made to the Cloud Data Access (CDA) Application Programming Interface (API) services. The supplied IDF files for the various high-level languages contain the necessary definitions that ensure that the parameter list passed to the CDA API has the high-order bit turned on for the last parameter. For example, for C, the linkage must be specified as OS linkage, such as:

```
#pragma linkage(HWTHxxxx_CALLTYPE,OS)
```

The GDKIC C header file includes this pragma.

Linkage considerations for assembler language programming

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 - 14 are restored (registers 0, 1 and 15 are not restored).

Compilation consideration

The DFSMSdfp CDA product provides one or more sample programs in most of the supported languages to aid in the creation of applications that use the APIs. Refer to the sample JCL jobs in the prolog section of the sample of your language choice for the recommended compiler and linking options. The DFSMSdfp CDA API programming sample programs can be found in SYS1.SAMPLIB with the 3 character ID of GDK. For example, GDKIC1 is a sample C program.

Code page consideration

All input data into the Cloud Data Access services, except for body data, is assumed to be in EBCDIC encoding (code page 1047) before making any Cloud Data Access service call. The z/OS Client Web Enablement Toolkit converts any required data to meet any RFC code page standards when sending or receiving HTTP header data. Data specified in the requested body or received in the response body is treated as-is unless the application requests conversion of the data. Optional parameters on the GDKGET

or GDKWRITE APIs allow for override of the default local code page, IBM-1047, and remote code page ISO8859-1.

Environmental considerations

The following environmental considerations apply:

OMVS segment

(Required) The CDA APIs use z/OS Web Enablement Toolkit services, which use TCP/IP sockets and Cryptographic Services System SSL services. Because these services require a z/OS UNIX (POSIX) environment, the CDA APIs run with a Language Environment POSIX(ON) environment. A POSIX(ON) environment requires the user ID associated with the address space using the CDA API services to have an OMVS segment defined and associated with it. See the appropriate security product documentation, as applicable to your installation, for instructions on how to define an OMVS segment to a user.

z/OS UNIX limit of processes with a POSIX(ON) environment and its effect on concurrent connections:

z/OS UNIX limits the number of concurrent POSIX(ON) environments that can be defined to a single address space. Since each CDA API call attempts to initialize a new POSIX(ON) environment (because of the environmental requirements listed earlier), each connection gets implicitly dubbed by Language Environment. (Dub means to make a z/OS address space known to z/OS UNIX System Services. Once dubbed, an address space is considered to be a process.) If multiple connections are wanted from the same application (address space), a consideration of the dubbing configuration for the address space in which the CDA API service is running may be necessary.

The dubbing behavior that z/OS UNIX takes is customizable by using the z/OS UNIX set dub default service (BPX1SDD or BPX4SDD). If the dub default for the address space is set to DUBPROCESS, this allows concurrent subtasks to each have their own POSIX(ON) environment, which allows multiple connections from within the same address space, provided that each subtask has at most one connection.

z/OS Language Environment Heap runtime option considerations:

The CDA API service obeys the current Language Environment Runtime HEAP option for storage management. Applications can customize the HEAP storage option for their execution environment as needed. IBM does not recommend using a primary heap size that is smaller than 32K.

z/OS Language Environment Runtime Environment REUSE (RTEREUS) option consideration:

COBOL applications running in Information Management System (IMS) should not use the RTEREUS(ON) runtime option. For more information, see [RTEREUS \(COBOL only\)](#) in *z/OS Language Environment Customization*.

Security considerations

There are several aspects of security to consider at both the connection level and the request level.

Connection security: Considerations for connection security include:

TCP/IP stack and security product control

A CDA API application connecting to a server is limited by security profiles and definitions that are already in effect on the system where the application resides. Powerful controls, such as z/OS Communications Server NetAccess, in conjunction with security profiles defined using the SERVAUTH class, can be used to control network access authority, TCP/IP stack access authority, port access authority, and more. These security definitions can be configured to be as granular as required by an installation. For more information, see [z/OS Communications Server: IP Configuration Guide](#) and [z/OS Communications Server: IP Configuration Reference](#).

Large data body considerations

Under some circumstances, applications require a large data body to be sent on a single request or received on a corresponding response. In these cases, you can supply a streaming send exit that conforms to the requirements for the z/OS Client Web Enablement Toolkit streaming exit, which can

be used to provide the request body as an ordered sequence of contiguous pieces of data whose number, size, and location are completely at the discretion of the exit. Similarly and independently, applications that expect a very large response body can supply a streaming receive exit to accept the response body as an ordered sequence of contiguous pieces of data of unpredictable number and size.

Problem determination considerations

Problem determination in an application can be challenging. Here are a few debugging options that can aid in the debugging of your application.

In the config.json file, stored in the ~/gdk/ directory, two options are available:

log-level

Can be used to indicate the logging level during CDA API processing. CDA API logging is written to stderr. When executing in a JCL environment, stderr is defined by the C MSGFILE environment variable. The default is a DD named SYSOUT. For more information, see [“Config file” on page 691](#).

web-toolkit-logging

Requests logging from the z/OS Client Web Enablement Toolkit. Output is written to stdout. Specify true to request logging. The special key, unredacted may be specified to request unredacted output in the log.

A Diagnostic area may be passed to the CDA API via pointer in the list of Optional Parm's. This diagnostic area (diagArea) is passed to the z/OS Client Web Enablement Toolkit, where they may place a return code and text for errors encountered during their processing. For more information, see [Part 9, “z/OS client web enablement toolkit,” on page 465](#).

Recovery considerations

The Cloud Data Access Services runs in the address space of the application. In addition, all the storage needed by the CDA API is obtained in the application's address space. Because every application has its own programming environment, it is impossible for the CDA API to predict the recovery environment required by the application.

When the CDA API attempts to access application-provided parameters and those parameters are either inaccessible, point to an inaccessible location, or specify a length that goes beyond the available storage obtained by the application, an abend occurs.

Note: Language Environment callers can see a Language Environment-specific abend code other than 0C4. Under certain circumstances, the Language Environment message, CEE3501S The module xxxxxxxx was not found, can appear in standard output, where xxxxxxxx is the application program's Language Environment condition handler. The calling Language Environment program still receives control with the failing API return code and diagArea information.

Chapter 30. Cloud Data Access callable services

The z/OS Cloud Data Access callable services are:

- “GDKGET — Retrieve a cloud object” on page 716
- “GDKWRITE — Write an object to cloud storage” on page 728
- “GDKDEL — Delete an object from cloud storage” on page 742
- “GDKLIST — List cloud objects” on page 748
- “GDKLISTL — List cloud objects” on page 755
- “GDKMSGTR — Translate a CDA return code into text” on page 762
- “GDKGETP — List cloud providers” on page 765
- “GDKKEYSR — Retrieve cloud credentials” on page 769
- “GDKKEYAD — Store cloud credentials” on page 773
- “GDKKEYDL — Delete cloud credentials” on page 778
- “GDKKEYGR — List resources for provider” on page 781
- “GDKINIT — Initialize a Session” on page 785
- “GDKTERM — Terminate a Session” on page 789
- “GDKGEN — Perform a Configurable request” on page 792
- “GDKVALD — Validate Provider File” on page 802
- “GDKQUERY — Query available CDA functions” on page 807

All APIs accept a pointer to an optional parameter structure that describe a group of key:value pairs that can be used to modify the processing done by the particular API. The specific keys that are recognized varies per API and are described with each API. The APIs accept a pointer to the GDK_OPTIONAL_PARMS_TYPE structure, or NULL if none are desired. GDK_OPTION_KEY_TYPE is a pointer to character array (char *). GDK_OPTION_VALUE_TYPE is also a pointer to a character array unless the value is expected to be different.

```
typedef struct {
    uint32_t GDK_NumberOfOptions;
    GDK_OPTION_TYPE GDK_OptionArray[];
} GDK_OPTIONAL_PARMS_TYPE;

typedef struct {
    GDK_OPTION_KEY_TYPE GDK_optionKey;
    GDK_OPTION_VALUE_TYPE GDK_optionValue;
} GDK_OPTION_TYPE;
```

Table 135. GDK_OPTIONAL_PARMS_TYPE

GDK_OPTIONAL_PARMS_TYPE				
Offset	Len	Type	Name	Description
0		STRUCTURE	GDK_OPTIONAL_PARMS_TYPE	Structure of one optional parameter.
0	4	NUMBER	GDK_OptionArray	Array of GDK_OPTION_TYPE entries
4	*	ARRAY	GDK_OptionArray	Array of GDK_OPTION_TYPE entries.

Table 136. GDK_OPTION_TYPE

GDK_OPTION_TYPE				
Offset	Len	Type	Name	Description
0		STRUCTURE	GDK_OPTION_TYPE	Structure of one optional parameter.
0	4	POINTER	GDK_OptionKey	Pointer to Null-terminated character string for the key of the optional parameter.
4	4	POINTER	GDK_optionValue	Pointer to Null-terminated character string or value for the value related to the optional parameter.

GDKGET – Retrieve a cloud object

The GDKGET API is used to retrieve a cloud object from the specified cloudProvider into the location specified by dataLocationType and dataLocation.

Description

The GDKGET API is used to retrieve an object from the specified cloud provider. When called, the provider definition for the requested cloud provider will be retrieved and the security credentials for the invoking user will be retrieved. Those security credentials will be used to send a request to retrieve the requested object name. An HTTP request is sent to the cloud object storage server to retrieve the object with the requested data according to the entry in the GETOBJECT entry in the supported operations array in the cloud provider json file. If the dataLocationType is GDK_FILE_DATALOCATION and the GETLARGEOBJECT entry exists, that will be used instead, as it contains getSize and data required entries. If the retrieve is successful, the object will be placed in the specified location.

The GDKGET API processing can additionally be modified through the optionalParmStruct.

Syntax

```
gdkget (retCodeAddr,
        cloudProvider,
        objectName,
        dataLocationType,
        dataLocation,
        dataLocationLenAddr,
        optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProvider

Specifies the address of a pointer to a name of the cloud provider that this request should contact. The name must be null-terminated. When retrieving the cloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory

does not exist or the <cloudProvider>.json file is not found there, then the CDA system default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

The application may request a list of all supported providers by using the **GETPROVIDERS** operation.

objectName

Specifies the address of a pointer to the remote object name to be retrieved. This value must specify the complete remote cloud provider location description including bucket/container and/or path and/or file name to be retrieved (for example, /bucket2/dir1/CDA.pdf). The name must start with a forward-slash (/), and the bucket/container name is the characters up to the next forward-slash (/). The name passed must be null-terminated. The bucket/container name is folded to lowercase. The name is not checked in general for valid URL characters. For more information, see [RFC 3986](http://www.ietf.org/rfc/rfc3986.txt) (www.ietf.org/rfc/rfc3986.txt).

When interacting with a provider using the AWS4 authentication model, some characters are encoded as follows:

Character	Encoded version
/	%2F
"encode": "special" in provider file	
!	%21
*	%2A
(%28

dataLocationType

Specifies a pointer to an unsigned number. When retrieving an object from the cloud data provider, the z/OS Cloud Data API provides several options to the application as to where the retrieved object should be directed. The possible values are defined as integer constants and are defined in the various language include files. The possible values are:

GDK_BUFFER_DATALOCATION

Tells the CDA API to use a buffer defined by the user. The length of the buffer must be specified by dataLocationLen. This parameter specifies placing the object file into the buffer location specified in dataLocation for a length up to dataLocationLen.

GDK_PATH_DATALOCATION

Tells the CDA API to store the cloud object in a zFS hierarchical file. The dataLocation specifies the name of the file, and the dataLocationLen tells how long the file name is.

GDK_EXIT_DATALOCATION

Tells the CDA API that special processing of the returned data is necessary to process the request. This option may be required when there is a very large amount of data that could be received in a format that is only known to the application. This also allows the internal management of buffer pools to maximize efficiency and throughput. This parameter specifies to invoke the z/OS Client Web Enablement Toolkit Streaming Receive exit to custom process the incoming cloud object. The dataLocation must be filled in with the address of the stream receive exit. The dataLocationLen must be set to zero. For more information, see [“Streaming receive exit” on page 663](#).

The GETLARGEOBJECT operation can be utilized by the caller's streaming receive exit to break up a large object across multiple HTTP requests. The "regex" optional parameter should be used to indicate that the GETLARGEOBJECT operation is proffered. The "regex" functionality can also be used to stream multiple objects yet be seen by the caller's streaming exit as a single object.

dataLocation

Specifies a pointer to a field for the location where the data is to be retrieved from, depending on the dataLocationType selected. The following table describes the relationship between the dataLocationType and the expected value type of the dataLocation parameter:

dataLocationType	dataLocation value
GDK_BUFFER_DATALOCATION	4-byte buffer address
GDK_PATH_DATALOCATION	zFS file name, null-terminated
GDK_EXIT_DATALOCATION	4-byte address of z/OS Client Web Enablement Toolkit streaming receive exit

dataLocationLenAddr

Specifies a pointer to a field containing the length of the field described by the dataLocation parameter (input/output). Upon return, this value is updated to indicate the number of bytes received. The following table describes the meaning of dataLocationLen, based on the dataLocationType specified:

dataLocationType	dataLocationLen value
GDK_BUFFER_DATALOCATION	Length of the buffer specified (supplied)/Length of buffer written/required (returned)
GDK_PATH_DATALOCATION	Length of the zFS file name specified
GDK_EXIT_DATALOCATION	N/A

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	UserID 8-byte char RACF UserID used to retrieve provider files for the GET request.
Content-Length	Integer	Integer value to place in the HTTP header of the request indicating the maximum length accepted.
Range-Start	Integer	Specifies the byte number for the data retrieve to start at. Values are 0 based and inclusive. If not specified, then it is assumed to start at 0. For an out of range (invalid) value for the Range-Start and Range-End parameters, GDKGET will return the whole data. Some cloud providers, however, may have another way of dealing with such requests in which case the data returned is dependent on the Cloud provider.

Optional Parameters		
NAME	Type	Description
Range-End	Integer	<p>Specifies the ending byte number for the request. If not specified (and Range-Start is specified), then read to end. If Range-End is specified (or is 0), but Range-Start is not specified, then the last number of bytes, inclusive, are returned from the object.</p> <p>For an out of range (invalid) value for the Range-Start and Range-End parameters, GDKGET will return the whole data. Some cloud providers, however, may have another way of dealing with such requests in which case the data returned is dependent on the Cloud provider.</p>
Get-HWTH-Code	Address	Specifies the address of a 4-byte Integer field where the resulting z/OS Client Web Enablement Toolkit return code should be stored. For more information, see “HWTHRQST — Send a request to an HTTP server” on page 613.
Get-HWTH-Diag	Address	Specifies the address of an HWTH_DIAGAREA_TYPE area where the resulting z/OS Client Web Enablement Toolkit diagnostic information should be stored. For more information, see “HWTHRQST — Send a request to an HTTP server” on page 613.
AutoConvert	Character	<p>true means that ASCII to EBCDIC translation should be performed on the data being retrieved from the cloud. false means that no translation should be performed.</p> <p>This optional parameter will override any default, or value in the config.json file.</p> <p>If not passed, the default is no translation of data.</p>
Set-Header-Buffer	Character	String of custom headers that should be included on the HTTP GET request. header is newline (\n) separated.
Get-Header-Buffer	Address	Pointer to storage area to place the headers from the HTTP Response.
Get-Header-Buffer-Size	Address	Pointer to a 4-byte signed number field that is the size of the provided buffer to contain the headers. Must be provided if Get-Header-Buffer is specified.
Get-Response-Code	Address	Pointer to 4-byte area to place the HTTP status code from the HTTP GET request.

Optional Parameters		
NAME	Type	Description
Use-Config-File	Character	false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.
log-level	Character	The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stdout. The levels in order of low to high severity are listed: <ul style="list-style-type: none"> • DEBUG means all logging messages are writteng. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.
web-toolkit-logging	Character	false means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should not be written. true means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should be written. This value will override any default or value in the config.json file. Logging messages are written to stdout.
userdata	Address	When the DataLocationType is GDK_EXIT_DATALOCATION, the address of user storage. This address is passed to the exit that the z/OS Client Web Enablement Toolkit calls.
CDA_Session	Address	Specifies the address of a 12-byte field that was filled in as part of a gdkinit() CPI call. This indicates a session is a current and the config file, keyfile, and provider file will not be read again during this API call.

Optional Parameters		
NAME	Type	Description
regex	Address	Specifies the address of a null-terminated string value that is to be used as a regular expression determining which objects are streamed via the caller's streaming receive exit. (Only valid when GDK_EXIT_DATALOCATION is specified as the dataLocationType.) When specified with a non-empty value, the objectName value is used as a PREFIX to retrieve a list of objects that begin with that name. The GETLARGEOBJECT operation is used to stream each of the objects to the caller's streaming exit. When specified with an empty value, the objectName is retrieved using the GETLARGEOBJECT operation and streamed to the caller's streaming exit.
sort-mechanism	Address	<p>Specifies the address of a null-terminated string value indicating the sorting mechanism used to order the list of objects that are retrieved. (Only examined when "regex" is a non-empty value.) Acceptable values are:</p> <ul style="list-style-type: none"> • "sort-alpha-with-num" – The results are sorted on the alphabetical portion followed by the number value. If you know that the object names returned from a LIST with PREFIX have the form <text><num><text>, this sorting mechanism should be used. • "sort-alphanum" – The names matched can be sorted in alphanumeric order. i.e. Name1 compared to Name2. <p>If not specified, then the order of object names returned from the cloud server is how they are delivered to the caller's streaming receive exit.</p>
max_objs	Address	Specifies the address of a null-terminated string value indicating the storage to contain the maximum amount of object names when finding the matching object names. If not specified, the default value is 1000.
prefix	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_PREFIX substitution text in a URL_PARM request parameter.
delimiter	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_DELIMITER substitution text in a URL_PARM request parameter.

Optional Parameters		
NAME	Type	Description
versionID	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_VERSION_ID substitution text in a URL_PARM request parameter.
logOutput	Address	Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.
providerFile	Address	Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.
localCodePage	Address	Specifies the address of a null terminated string containing a name of a valid code page to be used in conversion of local text data from local to remote, or remote to local.
remoteCodePage	Address	Specifies the address of a null terminated string containing a name of a valid code page to be used in conversion of remote text data from local to remote, or remote to local.

Optional Parameters		
NAME	Type	Description
compression	Character	<p>String with one of the following options:</p> <ul style="list-style-type: none"> • zEDC – DFSMSdfp CDA will use the zEDC compression algorithm to decompress the data from the object. • gzip – DFSMSdfp CDA will use the gzip compression algorithm to decompress the data from the object. <p>If this parameter is not passed, DFSMSdfp CDA will look for the “zos-compression” metadata tag to know whether to decompress data and which algorithm to use. If the object does not have the “zos-compression” metadata tag and the “compression” optional parameter wasn’t passed, DFSMSdfp CDA will not attempt to decompress the data. (Note the “decompress” optional parameter may override this.) The GETOBJECT and GETLARGEOBJECT operations must have a METAHEADER entry in the responseResults array that describes the prefix for a metadata header.</p>
decompress	Character	<p>String with one of the following options:</p> <ul style="list-style-type: none"> • “true” indicates that DFSMSdfp CDA will attempt to decompress the data from the object. If the data is not compressed, the operation is stopped with a return code 149 (GDK_DECOMPRESSION_FAILURE). • “false” indicates that DFSMSdfp CDA will not attempt any decompression, regardless of the existence of the “zos-compression” metadata tag on the object. • “attempt-inflate” indicates that DFSMSdfp CDA should attempt decompression of the data. If an error occurs due to the data not being compressed, processing will continue with the object data being given to the caller as-is. • <p>If this optional parameter is not passed, DFSMSdfp CDA will examine the object’s HTTP headers for a “zos- compression” metadata tag. If the object doesn’t have the zos-compression metadata header or the responseResults array doesn’t contain a METAHEADER description, then no decompression will be performed.</p>

Optional Parameters		
NAME	Type	Description
cloudformat	Character	<p>This can have a value of “record” or “none”.</p> <ul style="list-style-type: none"> • “record” indicates that you know that the object contains imbedded record prefixes separating the record data. • “none” indicates that no processing of imbedded record data should be performed. <p>When an object has the metadata tag indicating cloud format:record, or it is requested, metadata tags with the keys listed in the table below will be recognized and used to create the local data set when the local data set is not cataloged.</p>
storclas	Character	String containing a value to override the name found from the object metadata tags. The string can be simply the empty string, which indicates that no STORCLAS name should be passed on the allocation.
mgmtclas	Character	String containing a value to override the name found from the object metadata tags. The string can be simply the empty string, which indicates that no MGMTCLAS name should be passed on the allocation.
dataclas	Character	String containing a value to override the value found from the object metadata tags. The string can be simply the empty string, which indicates that no DATACLAS name should be passed on the allocation.
volumes	Character	String containing a comma separated list of volume serials to override the volume serials used on the create request.
unit	Character	String containing a value to override the unit name passed on the DYNALLOC request. The string can be simply the empty string, which indicates that no UNIT name should be passed. When not specified, the UNIT name used is SYSALLDA.
LocalSize	Character	String that contains a number indicating the number of bytes that the primary allocation of the data set should hold.

Metadata tags

When cloudformat is requested, or the zos-filedata metadata tag with a value of record is found, the following metadata tags will be recognized and used to create the local data set, if needed.

Metadata key	Value
zos-lrecl	Set to the logical record length of the data set

Metadata key	Value
zos-recfm	Set to the record format of the data set
zos-blksize	Set to the block size of the data set
zos-dsorg	Set to the data set organization of the data set.
zos-dataclas	Set to the DATACLAS for the data set
zos-mgmtclas	Set to the SMS Management Class for the data set
zos-storclas	Set to the SMS Storage Class for the data set
zos-secondary	Set to the secondary allocation amount in the format <nnnn><type>where type is CYLS, TRKS, BLKS
zos-vs-account	Set to the value of the ACCOUNT for the VSAM data set if it exists.
zos-vs-buffspace	Set to the value of the BUFFSPACE for the VSAM data set if not default.
zos-vs-bwotype	Set to the backup-while-open value for the VSAM data set.
zos-vs-dcsize	Set to the VSAM cluster Data Component CISIZE
zos-vs-icsize	Set to the Indexed VSAM cluster Index Component CISIZE.
zos-eattr	Set to the Extended Attribute value for the VSAM data set.
zos-vs-erase	Set to the ERASE setting for the VSAM data set
zos-vs-freespace	Set to the VSAM data set FREESPACE value
zos-vs-keylabel	Set to the VSAM data set KEYLABEL name if it exists
zos-vs-keys	Set to the Primary Key field information for the VSAM data set.
zos-vs-log	Set to the value of the LOG keyword for the VSAM data set.
zos-vs-logreplicate	Set to the value of the LOGREPLICATE keyword for the VSAM Data set.
zos-vs-logstreamid	Set to the LOGSTREAMID value for the VSAM data set if it exists.
zos-vs-owner	Set to the OWNER value for the VSAM data set if it exists.
zos-vs-recordsize	Set to the RECORDSIZE value for the VSAM data set.
zos-vs-spanned	Set to SPANNED when the VSAM data set was defined as having spanned records.
zos-volumes	Set to a hyphen separated list of volume serials for the data set.

Related services

- [“GDKWRITE – Write an object to cloud storage” on page 728](#)
- [“GDKDEL – Delete an object from cloud storage” on page 742](#)
- [“GDKLIST – List cloud objects” on page 748](#)

Usage notes

- When the API is invoked, if UserID wasn't provided in the optionalParms, the current user's RACF ID is used to retrieve the applicable cloudProvider definitions, as well as retrieve the appropriate security credentials for the User/Cloud Provider/Resource combination from the gdkkeyf.json document.
- When retrieving the cloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment in order to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory does not exist or the provider file is not found there, then the CDA System Default directory of /usr/lpp/dfsms/gdk/providers/ will be used.
- When retrieving the security credentials to use to communicate with the cloudProvider specified on the API call, the gdkkeyf.json file must first be located. The user's RACF ID, or UserID from the optionalParms, is used to find the home directory. That home directory is examined for a gdk/gdkkeyf.json document that can be accessed for READ. If not found, then the CDA API call fails.
- Once the keyfile has been found, it is parsed, looking for an entry associated with the user's RACF ID, or UserID from the optionalParms. The most specific bucket/container name (Resource) is searched for first. A bucket name is the first characters surrounded by a forward slash character (for example, / bucket_name/). If no entry is found, then the generic / Resource name is searched for.
- With the most appropriate entry found, the security credentials will be decrypted using ICSF. The decryption key is expected to be found using a keylabel of GDK.<cloudProvider>.<UserID/RACF ID>.<sequence_number>. If RACF protection of the ICSF key labels is being used, the invoker of the API must have authority to the key label.
- This API is supported on z/OS 2.4 and higher.

Restrictions

None.

Authorization

User.

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 137. Return and reason codes for the GDKGET service		
Return code Equate symbol	Constant Name	Explanation
0	GDK_OK	Processing was successful.
100	GDK_UNABLE_TO_READ_KEYFILE	The gdkkeyf.json document was not found in the CDA System directory.
102	GDK_UNABLE_TO_PARSE_KEYFILE	When parsing the gdkkeyf.json document, it is not a valid JSON document.
104	GDK_CLOUD_PROVIDER_NOT_FOUND	When reading the gdkkeyf.json document, for the requested user, the specified cloudProvider was not found.
105	GDK_NO_RESOURCE_FOUND_IN_KEYFILE	For the specified UserId, and cloudProvider, no credentials entry for / was found.
110	GDK_PROVIDER_OPEN_FAILURE	The JSON document for the requested cloudProvider does not exist, or was found, but cannot be opened for READ.
111	GDK_PROVIDER_SPECIFICATION_INVALID	When parsing the JSON document, it was found to be invalid.
112	GDK_FEATURE_UNSUPPORTED	An unknown contentType was found in the parameterSet.

Table 137. Return and reason codes for the GDKGET service (continued)

Return code Equate symbol	Constant Name	Explanation
113	GDK_BUFFER_TOO_SMALL	When decrypting the security credentials, the specified buffer was too small to hold the decrypted credential.
114	GDK_AUTH_INIT_FAILURE	When using the AWS4 authentication model, the appropriate entry in the gdkkeyf.json document for the MVSUserID, cloud provider, and name (resource) is missing either the key, or the secretkey key-value pairs.
115	GDK_COULD_NOT_OPEN_OUT PUT_FILE	Unable to open the requested local file for output.
116	GDK_AUTH_APPLY_FAILURE	An error occurred while applying the authorization parameters.
118	GDK_USER_INFO_NOT_FOUN D	When reading the gdkkeyf.json document, the requested user entry was not found.
123	GDK_CONN_NOT_HTTPS	HTTPS was not specified for httpMethod in the provider file.
141	GDK_PARAMETER_ERROR	One of the parameters on the API call was incorrect. Examine the ERROR log message for detail son the parameter in error.
142	GDK_EMPTY_PROVIDER_FILE	The specified <cloudProvider>.json file is empty. <cloudProvider> is the provider name specified on the API parameter list.
145	GDK_ICONV_ERROR	Conversion error, converting from one code page to another code page.
148	GDK_DECOMPRESSION_INIT_ FAILURE	An error occurred while initializing the stream to perform decompression.
149	GDK_DECOMPRESSION_FAIL URE	An error occurred while decompressing the data.
151	GDK_NO_MEMORY_AVAILABL E	Could not allocate memory for decompressing the data.
152	GDK_INVALID_COMP_PARMS	Invalid decompress or compression parameter passed.
158	GDK_IO_ERROR	An unexpected error occurred with I/O to a data set.
159	GDK_INVALID_METADATA	An invalid metadata string was passed. Examine the log for ERROR messages relating to which metadata symbol was invalid for the operation.
160	GDK_INVALID_CF_PARMS	"cloudformat" was passed, but the value was invalid.
161	GDK_UNSUPPORTED_DS	The requested data set is of a type that DFSMSdfp CDA does not support for processing.
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.
800	GDK_CONNECTION_FAILED	Unable to connect to the host set in the cloud provider JSON definition.
801	GDK_TOOLKIT_FAILED	Error occurred in the z/OS Client Web Enablement Toolkit HWTHRQST call.
900	GDK_OBJECT_NOT_FOUND	HTTP status 404 was returned indicating the object was not found.
901	GDK_ACCESS_DENIED	HTTP status 403 was returned indicating the user is not authorized to access the resource in the desired manner.
902	GDK_OPERATION_NOT_SUPP ORTED	When parsing the cloud provider JSON document, the GETOBJECT description was not found in the supportedOperations array.

Table 137. Return and reason codes for the GDKGET service (continued)

Return code Equate symbol	Constant Name	Explanation
903	GDK_RESPONSE_FORMAT_MISMATCH	The request was successful, but the data returned was in a format that did not match what is expected according to the contentType specified in the responseResults for the action in the provider file.
904	GDK_REQUEST_FAILED	A bad HTTP status (4xx or 5xx) was returned.

GDKWRITE – Write an object to cloud storage

The GDKWRITE API is used to store the cloud object using the specified cloudProvider. If the object does not exist, it will be created.

Description

The GDKWRITE API is used to put data into an object in a cloud provider. When called, the provider definition for the requested cloud provider will be retrieved and the security credentials for the invoking user will be retrieved. Those security credentials will be used to send a request to put the requested data into the cloud with the specified object name. An HTTP request is sent to the cloud object storage server to create the object with the requested data according to the entry in the WRITEOBJECT entry in the supportedOperations array in the Cloud Provider json file.

The GDKWRITE API processing can additionally be modified through the optionalParmStruct.

Syntax

```
gdkwrite (returnCodeAddr,
          cloudProvider,
          objectName,
          dataLocationType,
          dataLocation,
          dataLocationLen,
          optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProvider

Specifies the address of a pointer to a name of the cloud provider that this request should contact. The name must be null-terminated. When retrieving the cloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory does not exist or the <cloudProvider>.json file is not found there, then the CDA system default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

The application may request a list of all supported providers by using the **GETPROVIDERS** operation.

objectName

Specifies the address of a pointer to the remote object name to be retrieved. This value must specify the complete remote cloud provider location description including bucket/container and/or path and/or file name to be retrieved (for example, /bucket2/dir1/CDA.pdf). The name must start with

a forward-slash (/), and the bucket/container name is the characters up to the next forward-slash (/). The name passed must be null-terminated. The bucket/container name is folded to lowercase. The name is not checked in general for valid URL characters. For more information, see [RFC 3986 \(www.ietf.org/rfc/rfc3986.txt\)](http://www.ietf.org/rfc/rfc3986.txt).

When interacting with a provider using the AWS4 authentication model, some characters are encoded as follows:

Character	Encoded version
/	%2F
"encode": "special" in provider file	
!	%21
*	%2A
(%28

dataLocationType

Specifies a pointer to an unsigned number. When storing an object from the cloud data provider, the z/OS Cloud Data API provides a number of options to the application as to the source of the object to be stored. The possible values are:

GDK_BUFFER_DATALOCATION

Specifies to use the buffer location specified in dataLocation for a length up to dataLocationLen as the source of the object to be written.

GDK_PATH_DATALOCATION

Specifies to use a z/OS data set or zFS hierarchical file as the source for the object to be written. Sequential data sets and PDS members are supported. The dataLocation specifies the name of the file or data set, and the dataLocationLen tells how long the file or data set name is (including a possible member name in parentheses). If the object is larger than 8 megabytes, the WRITELARGEOBJECT entry in the supportedOperations array is used to perform a multi-part upload, where the init, complete, and error actions are optional. Only the data action is required. If the WRITELARGEOBJECT entry does not exist in the supportedOperations array, then the WRITEOBJECT action is used.

GDK_EXIT_DATALOCATION

Tells the CDA API that special processing of the data to be written is necessary to process the request. This option may be required when there is a very large amount of data that could be written in a format that is only known to the application. This also allows the internal management of buffer pools to maximize efficiency and throughput. This parameter specifies to invoke the z/OS Client Web Enablement Toolkit Streaming Send exit to custom process the incoming cloud object. The dataLocation must be filled in with the address of the streaming send exit. The dataLocationLen must be set to zero.

The REST API for some cloud storage providers require that a "Content-Length" header is included on every request. If you know the total amount of data that will be sent via the streaming exit, then you should use the "Content-length" or "Content-LengthE" optional parameters to pass that value to be included on the request. After application of OA65224, support is added to buffer data from the streaming exit so that the Multipart Upload support described by WRITELARGEOBJECT can be used. When no content length is specified via the optional parameters, the Multipart Upload support will be used.

The size of the buffer may be modified by the "multipartChunksize" optional parameter. If not specified, the buffer is 8MB in size. If the caller's streaming send exit sets the state to HWTB_STREAM_SEND_EOD before the first buffer has reached 5MB, the data will be sent in a single request, not utilizing the Multipart Upload feature.

dataLocation

Specifies a pointer to a field where the source data to be written to the cloud provider is to be obtained, depending on the dataLocationType selected. The following table describes the relationship between the dataLocationType and the expected value type of the dataLocation parameter:

dataLocationType	dataLocation value
GDK_BUFFER_DATALOCATION	4-byte buffer address
GDK_PATH_DATALOCATION	zFS file name, null-terminated
GDK_EXIT_DATALOCATION	4-byte address of z/OS Client Web Enablement Toolkit streaming receive exit

dataLocationLen

Specifies the length of the dataLocation parameter (input). The following table describes the meaning of dataLocationLen, based on the dataLocationType specified:

dataLocationType	dataLocationLen value
GDK_BUFFER_DATALOCATION	Length of the buffer specified.
GDK_PATH_DATALOCATION	Length of the zFS file name or z/OS data set name (including possible member name) specified.
GDK_EXIT_DATALOCATION	N/A

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	UserID 8-byte char RACF UserID used to retrieve provider files for the PUT request.
Content-Length	8-byte Integer	64-bit signed number value containing the amount of data being written.
Content-Length	Integer	4-byte signed number value to place in the HTTP header of the request indicating the length of the data sent. When using the GDK_EXIT_DATALOCATION type, some providers will require this to be set.
Range-Start	Integer	<p>Specifies the byte number for the data write to start at. Values are 0 based and inclusive. If not specified, then it is assumed to start at 0.</p> <p>GDKWRITE creates a request with range parameters but it depends on the cloud provider how to handle the request. Most of the providers will write the complete data to the cloud bucket.</p>

Optional Parameters		
NAME	Type	Description
Range-End	Integer	<p>Specifies the ending byte number for the request. If not specified (and Range-Start is specified), then read to end. If Range-End is specified (or is 0), but Range-Start is not specified, then the last number of bytes, inclusive, are returned from the object.</p> <p>GDKWRITE creates a request with range parameters but it depends on the cloud provider how to handle the request. Most of the providers will write the complete data to the cloud bucket.</p>
Get-HWTH-Code	Address	<p>Specifies the address of a 4-byte Integer field where the resulting z/OS Client Web Enablement Toolkit return code should be stored. For more information, see “HWTHRQST — Send a request to an HTTP server” on page 613.</p>
Get-HWTH-Diag	Address	<p>Specifies the address of an HWTH_DIAGAREA_TYPE area where the resulting z/OS Client Web Enablement Toolkit diagnostic information should be stored. For more information, see “HWTHRQST — Send a request to an HTTP server” on page 613.</p>
AutoConvert	Character	<p>true means that ASCII to EBCDIC translation should be performed on the data being retrieved from the cloud. false means that no translation should be performed.</p> <p>This optional parameter will override any default, or value in the config.json file.</p> <p>If not passed, the default is no translation of data.</p>
Set-Header-Buffer	Character	<p>String of custom headers that should be included on the HTTP PUT request. header is newline (\n) separated.</p>
Get-Body-Buffer	Address	<p>Pointer to storage to place the response body contents into.</p>
Get-Body-Buffer-Size	Address	<p>Pointer to a 4-byte signed number field that is the size of the provided body buffer. Must be provided if Get-Body-Buffer is specified.</p>
Get-Header-Buffer	Address	<p>Pointer to storage area to place the headers from the HTTP response.</p>
Get-Header-Buffer-Size	Address	<p>Pointer to a 4-byte signed number field that is the size of the provided buffer to contain the headers. Must be provided if Get-Header-Buffer is specified.</p>
Get-Response-Code	Address	<p>Pointer to 4-byte area to place the HTTP status code from the HTTP PUT request.</p>

Optional Parameters		
NAME	Type	Description
Use-Config-File	Character	false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.
log-level	Character	The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stdout. The levels in order of low to high severity are listed: <ul style="list-style-type: none"> • DEBUG means all logging messages are written to. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.

Optional Parameters		
NAME	Type	Description
metadata	Character	<p>Describes how to construct a metadata header. The “descriptor” key value contains the header prefix for metadata headers that are built as part of the metadata optional parameter processing.</p> <p>Null terminated string of comma-separated key:value pairs. These key value pairs will be turned into HTTP headers according to the METAHEADER entry in the requestParameters object in the GETOBJECT operation. The key portion will be folded to lowercase and appended to the descriptor value from the METAHEADER entry in order to create a provider appropriate metadata header tag.</p> <p>The value specified must be less than 2048 characters in length and may be:</p> <ul style="list-style-type: none"> • A constant value, which will be added as-is. Contents are not examined for sensitive data. • A System symbol specified with the ampersand character. i.e. &SYSNAME • A JCL symbol specified with the ampersand character (&) • A CDA Symbol specified with the ampersand character (&). See table below for list of supported CDA symbols. <p>If a symbol does not resolve to a value, it is ignored and not included in the metadata attached to the cloud object.</p> <p>The WRITEOBJECT, or WRITELARGEOBJECT operation in the provider file must have a mechanism:HEADER object with a META type in the requestParameters that describes the prefix specific to that Cloud Storage provider. The sample provider files include this. If the provider file does not have the META type object, the request will fail.</p> <p>An HTTP header will be created for each key:value pair in the comma separated string.</p>

Optional Parameters		
NAME	Type	Description
web-toolkit-logging	Character	<p>false means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should not be written.</p> <p>true means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should be written.</p> <p>This value will override any default or value in the config.json file. Logging messages are written to stdout.</p>
userdata	Address	When the DataLocationType is GDK_EXIT_DATALOCATION, the address of user storage. This address is passed to the exit that the z/OS Client Web Enablement Toolkit calls.
CDA_Session	Address	Specifies the address of a 12-byte field that was filled in as part of a gdkinit() CPI call. This indicates a session is current and the config file, keyfile, and provider file will not be read again during this API call.
multipartChunksize	Unsigned number	Specifies the size of a single chunk of data contained in a Multipart Upload request (described by the WRITELARGEOBJECT operation). The minimum value is 5242880.
prefix	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_PREFIX substitution text in a URL_PARM request parameter.
delimiter	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_DELIMITER substitution text in a URL_PARM request parameter.
versionID	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_VERSION_ID substitution text in a URL_PARM request parameter.

Optional Parameters		
NAME	Type	Description
logOutput	Address	Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.
providerFile	Address	Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.
localCodePage	Address	Specifies the address of a null terminated string containing a name of a valid code page to be used in conversion of local text data from local to remote, or remote to local.
remoteCodePage	Address	Specifies the address of a null terminated string containing a name of a valid code page to be used in conversion of remote text data from local to remote, or remote to local.

Optional Parameters		
NAME	Type	Description
compression	character	<p>String with one of the following options:</p> <ul style="list-style-type: none"> • zEDC – DFSMSdfp CDA will use the zEDC compression algorithm to compress the data sent to the object. • gzip – DFSMSdfp CDA will use the gzip compression algorithm to decompress the data sent to the object. <p>If the WRITEOBJECT and WRITELARGEOBJECT operations in the provider file have a METAHEADER description in the requestParameters, a metadata tag with the suffix “zos-compression” and a value indicating the type of compression used (zEDC or gzip) will be associated with the object.</p>
compLevel	character	<p>String with one of the following options:</p> <ul style="list-style-type: none"> • SPEED - perform fastest compression with minimal CPU usage; lower compression ratio. • MAX – Maximizes compression ratio with more CPU. • DEFAULT – Performs efficient compression, maintaining speed. <p>{1-9} – A number indicating the compression level to be used when initializing compression through the zlib APIs.</p>
Get-Sent-Data-LengthE	Address	The address of an 8-byte number field where total number of bytes sent to the server will be saved after successful completion.
cloudformat	character	<p>This can have a value of “record” or “none”.</p> <ul style="list-style-type: none"> • “record” indicates that you want the data set records to be separated by record prefixes as mapped by IGGRPFX, which indicates a length of the user record. Refer IGGRPFX mapping. • “none” is the default and indicates that no record prefixes should be imbedded in the cloud object. <p>Note: This optional parameter only applies when the data location parameter is GDK_PATHDATALOCATION, and the path name is for a z/OS Data Set. z/OS UNIX files are not supported.</p>

Table 138. IGGRPFIX mapping			
Offset	Length	Symbol	Description
0	1	RPFZ00	Reserved
1	3	RPFXLLL	Length of record that follows this prefix

Table 139. List of CDA symbols and their meaning	
CDA symbol	Meaning
GDK_BLKSIZE	Block size value from the z/OS data set
GDK_LRECL	LRECL value for the z/OS data set
GDK_RECFCM	Record Format value for the z/OS data set <ul style="list-style-type: none"> • F – Fixed records • FB – Fixed Blocked • FBA – Fixed Blocked ASA print-control characters • FBS – Fixed Blocked Standard • FS – Fixed Standard • V – Variable • VB – Variable Blocked • VBA – Variable Blocked with ASA print-control Characters • VBS – Variable Blocked Spanned records • U – Unknown
GDK_DSORG	Data Set organization <ul style="list-style-type: none"> • PS – Physical Sequential • PDS – Partitioned Data Set • LIBRARY – Partitioned Data Set Extended
GDK_EXPDATE	Expiration Date of the cataloged z/OS data set
GDK_CREDATE	Creation Date of the cataloged z/OS data set or UNIX file
GDK_REFDATE	Last Reference Date of the cataloged z/OS data set or last backup date of a z/OS UNIX file.
GDK_ISDATASET	True if z/OS data set. False if z/OS UNIX file
GDK_DATACLAS	Data Class associated with data set
GDK_MGMTCLAS	SMS Management Class associated with data set
GDK_STORCLAS	SMS Storage Class associated with data set
GDK_ONAME	Original Data Set or File name
GDK_JOBNAME	Original Data Set or File name
GDK_JOBNAME	JCL Job name that put this object into the Cloud
GDK_STEPNAME	JCL Step name that put this object into the Cloud

Table 139. List of CDA symbols and their meaning (continued)	
CDA symbol	Meaning
GDK_CTIME	Time the z/OS UNIX file metadata changed
GDK_MTIME	Time the z/OS UNIX file data changed
GDK_SECONDARY	Secondary allocation amount for the z/OS data set in the format: <nnn>-<unit>, where <i>nnn</i> is the number or amount, and unit is the allocation unit. e.g. CYL, TRK, BLK, REC, MB, KB, B
GDK_DSORGE	<ul style="list-style-type: none"> • PDSE for PDS/E dataset • PDS for PDS dataset • PS_LARGE when is a large format data set. (DS1LARGE) • PS_EXT when is extended format. (DS1STRP) • PS when data set is basic format sequential. • VSAMESDS when data set is Entry Sequenced VSAM • VSAMKSDS when data set is Key Sequenced VSAM
GDK_VSACCOUNT	32 bytes of accounting information and user data for a VSAM data set. Resolves to the value originally specified on the ACCOUNT()parameter when the VSAM data set was defined.
GDK_VSBUFFSPACE	The BUFFERSPACE value for the VSAM data set.
GDK_VSBWOTYPE	The backup-while-open (BWO) value for the VSAM data set. Will resolve to TYPECICS, TYPEIMS, or NO.
GDK_VSDCISIZE	The size of the control interval for the VSAM data set data component.
GDK_VSICISIZE	The size of the control interval for the VSAM data set index component.
GDK_EATTR	The Extended Attribute value for the VSAM data set. Resolved values are: OPT or NO.
GDK_VSERASE	Indicates whether the cluster's components are to be erased when its entry in the catalog are deleted. Resolved values are: ERASE or NOERASE.
GDK_VSFREESPACE	Value of the free space to be set aside after the initial load of the VSAM data set. Values are: cinnn-cannn
GDK_VSKEYLABEL	The key label name associated with the VSAM data set.

<i>Table 139. List of CDA symbols and their meaning (continued)</i>	
CDA symbol	Meaning
GDK_VSKEYS	<p>Primary key field information for the VSAM data set.</p> <p>Resolved values are in the form of len_nnn-off_nnn</p> <p>Where len_nnn is the length of the key, and off_nnn is the zero based offset in the record that the primary key starts at.</p>
GDK_VSLOG	<p>Value from the LOG keyword for the VSAM data set.</p> <p>Resolved values are: NONE, UNDO, and ALL.</p>
GDK_VSLOGREPLICATE	<p>Value for the VSAM data set eligibility for VSAM replication.</p> <p>Resolved values are: LOGREPLICATE, and NOLOGREPLICATE.</p>
GDK_VSLOGSTREAMID	Name for the forward recovery log stream for the VSAM data set.
GDK_VSOWNER	Resolves to the cluster's owner userid.
GDK_VSRECORDSIZE	<p>The average and maximum lengths of the records in the data component.</p> <p>Values are in the format: avg_nnn-max_nnn</p> <p>Where avg_nnn is the average record size, and max_nnn is the maximum record size.</p>
GDK_VSREUSE	<p>Resolves to the indicator whether the VSAM cluster can be opened again and again as a reusable cluster.</p> <p>Values are: REUSE or NOREUSE</p>
GDK_VSSHAREOPTIONS	<p>The share options value for the VSAM data set.</p> <p>Values are in the format: reg_n-sys_n</p> <p>Where reg_n is the cross region option value, and sys_n is the cross system option value. i.e. 1-3.</p>
GDK_VSSPANNED	<p>Attribute of the VSAM data set indicates whether it can contain records that cross control interval boundaries.</p> <p>Resolved values are: SPANNED, or NON_SPANNED</p>
GDK_VSPREFORMAT	<p>Attribute of the VSAM data set indicates whether the control areas of the VSAM data component should be per-formatted during loading.</p> <p>Resolved values are: SPEED, or RECOVERY.</p>

<i>Table 139. List of CDA symbols and their meaning (continued)</i>	
CDA symbol	Meaning
GDK_VOLSER	This will resolve to a hyphen separated list of volume serials for the current data set. e.g. VOL1-VOL002-VOLA1E.

Related services

- “GDKGET — Retrieve a cloud object” on page 716
- “GDKDEL — Delete an object from cloud storage” on page 742
- “GDKLIST — List cloud objects” on page 748

Usage notes

- When the API is invoked, if UserID wasn't provided in the optionalParms, the current user's RACF ID is used to retrieve the applicable cloudProvider definitions, as well as retrieve the appropriate security credentials for the User/Cloud Provider/Resource combination from the gdkkeyf.json document.
- When retrieving the cloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment in order to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory does not exist or the provider file is not found there, then the CDA System Default directory of /usr/lpp/dfsms/gdk/providers/ will be used.
- When retrieving the security credentials to use to communicate with the cloudProvider specified on the API call, the gdkkeyf.json file must first be located. The user's RACF ID, or UserID from the optionalParms, is used to find the home directory. That home directory is examined for a gdk/ gdkkeyf.json document that can be accessed for READ. If not found, then the CDA API call fails.
- Once the keyfile has been found, it is parsed, looking for an entry associated with the user's RACF ID, or UserID from the optionalParms. The most specific bucket/container name (Resource) is searched for first. A bucket name is the first characters surrounded by a forward slash character (for example, / bucket_name/). If no entry is found, then the generic / Resource name is searched for.
- With the most appropriate entry found, the security credentials will be decrypted using ICSF. The decryption key is expected to be found using a keylabel of GDK.<cloudProvider>.<UserID/RACF ID>.<sequence_number>. If RACF protection of the ICSF key labels is being used, the invoker of the API must have authority to the key label.
- This API is supported on z/OS 2.4 and higher.

Restrictions

When the WRITELARGEOBJECT operation doesn't exist in the provider, callers using the GDK_PATH_DATALOCATION or GDK_EXIT_DATALOCATION file will not be able to use signed payload to transfer the files larger than the buffer threshold of 8MB. In this case it fails with a return code 902(GDK_OPERATION_NOT_SUPPORTED) and an ERROR log message.

Authorization

User.

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 140. Return and reason codes for the GDKWRITE service

Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
100	GDK_UNABLE_TO_READ_KEYFILE	The gdkkeyf.json document was not found in the CDA System directory.
102	GDK_UNABLE_TO_PARSE_KEYFILE	When parsing the gdkkeyf.json document, it is not a valid JSON document.
104	GDK_CLOUD_PROVIDER_NOT_FOUND	When reading the gdkkeyf.json document, for the requested user, the specified cloudProvider was not found.
105	GDK_NO_RESOURCE_FOUND_IN_KEYFILE	For the specified UserId, and cloudProvider, no credentials entry for / was found.
110	GDK_PROVIDER_OPEN_FAILURE	The JSON document for the requested cloudProvider does not exist, or was found, but cannot be opened for READ.
111	GDK_PROVIDER_SPECIFICATION_INVALID	When parsing the JSON document, it was found to be invalid.
112	GDK_FEATURE_UNSUPPORTED	An unknown contentType was found in the parameterSet.
113	GDK_BUFFER_TOO_SMALL	When decrypting the security credentials, the specified buffer was too small to hold the decrypted credential.
114	GDK_AUTH_INIT_FAILURE	When using the AWS4 authentication model, the appropriate entry in the gdkkeyf.json document for the MVSUserID, cloud provider, and name (resource) is missing either the key, or the secretkey key-value pairs.
115	GDK_COULD_NOT_OPEN_OUTPUT_FILE	Unable to open the requested local file for output.
116	GDK_AUTH_APPLY_FAILURE	An error occurred while applying the authorization parameters.
118	GDK_USER_INFO_NOT_FOUND	When reading the gdkkeyf.json document, the requested user entry was not found.
123	GDK_CONN_NOT_HTTPS	HTTPS was not specified for httpMethod in the provider file.
141	GDK_PARAMETER_ERROR	One of the parameters on the API call was incorrect. Examine the ERROR log message for details on the parameter in error.
142	GDK_EMPTY_PROVIDER_FILE	The specified <cloudProvider>.json file is empty. <cloudProvider> is the provider name specified on the API parameter list.
145	GDK_ICONV_ERROR	Conversion error, converting from one code page to another code page.
146	GDK_COMPRESSION_INIT_FAILURE	An error occurred while initializing the stream to perform compression.
147	GDK_COMPRESSION_FAILURE	An error occurred while compressing the data.
151	GDK_NO_MEMORY_AVAILABLE	Could not allocate memory for compressing the data.
152	GDK_INVALID_COMP_PARMS	Invalid compression or compLevel parameter passed.
155	GDK_FULLY_QUALIFIED_DS_NOT_PASSED	The dataLocation name that is a z/OS data set is not a fully qualified data set name.

Table 140. Return and reason codes for the GDKWRITE service (continued)

Return code	Constant Name	Explanation
156	GDK_COULD_NOT_ALLOCATE_DATASET	An error occurred while CDA was trying to create a new data set. An ERROR level message will describe the S99ERROR or S99INFO values that describe the error. Details about the error codes can be found in “Interpreting error reason codes from DYNALLOC”. If the error was for a VSAM data set, refer to the IDCxxxxx messages that are included in the output. DEBUG level logging may provide additional information.
157	GDK_PDS_OR_PDSE_NOT_FOUND	The specified data set name for a PDS or PDSE data set was not found in the Catalog.
160	GDK_INVALID_CF_PARMS	“cloudformat” was passed, but the value was invalid.
161	GDK_UNSUPPORTED_DS	The requested data set is of a type that DFSMSdfp CDA does not support for processing.
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.
800	GDK_CONNECTION_FAILED	Unable to connect to the host set in the cloud provider JSON definition.
801	GDK_TOOLKIT_FAILED	Error occurred in the z/OS Client Web Enablement Toolkit HWTHRQST call.
900	GDK_OBJECT_NOT_FOUND	HTTP status 404 was returned indicating the object was not found.
901	GDK_ACCESS_DENIED	HTTP status 403 was returned indicating the user is not authorized to access the resource in the desired manner.
902	GDK_OPERATION_NOT_SUPPORTED	When parsing the cloud provider JSON document, the WRITEOBJECT description was not found in the supportedOperations array.
903	GDK_RESPONSE_FORMAT_MISMATCH	The request was successful, but the data returned was in a format that did not match what is expected according to the contentType specified in the responseResults for the action in the provider file.
904	GDK_REQUEST_FAILED	A bad HTTP status (4xx or 5xx) was returned.

GDKDEL – Delete an object from cloud storage

The GDKDEL API is used to delete the cloud object from the specified cloudProvider.

Description

The GDKDEL API is used to delete an object in a cloud provider. When called with a cloud provider and a object name, an HTTP request is sent according to the DELETEOBJECT array entry in the supportedOperations array in the provider file.

The GDKDEL API processing can additionally be modified through the optionalParmStruct.

Syntax

```
gdkdel (returnCodeAddr,
        cloudProvider,
        objectName,
        optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProvider

Specifies the address of a pointer to a name of the cloud provider that this request should contact. The name must be null-terminated. When retrieving the cloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory does not exist or the <cloudProvider>.json file is not found there, then the CDA system default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

The application may request a list of all supported providers by using the **GETPROVIDERS** operation.

objectName

Specifies the address of a pointer to the remote object name to be retrieved. This value must specify the complete remote cloud provider location description including bucket/container and/or path and/or file name to be retrieved (for example, /bucket2/dir1/CDA.pdf). The name must start with a forward-slash (/), and the bucket/container name is the characters up to the next forward-slash (/). The name passed must be null-terminated. The bucket/container name is folded to lowercase. The name is not checked in general for valid URL characters. For more information, see [RFC 3986 \(www.ietf.org/rfc/rfc3986.txt\)](http://www.ietf.org/rfc/rfc3986.txt).

When interacting with a provider using the AWS4 authentication model, some characters are encoded as follows:

Character	Encoded version
/	%2F
"encode": "special" in provider file	
!	%21
*	%2A
(%28

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	RACF UserID used to retrieve cloud security credentials for the DELETE request.
Get-HWTH-Code	Address	Specifies the address of a 4-byte Integer field where the resulting z/OS Client Web Enablement Toolkit return code should be stored. For more information, see “HWTHRQST — Send a request to an HTTP server” on page 613 .

Optional Parameters		
NAME	Type	Description
Get-HWTH-Diag	Address	Specifies the address of an HWTH_DIAGAREA_TYPE area where the resulting z/OS Client Web Enablement Toolkit diagnostic information should be stored. For more information, see “HWTHRQST — Send a request to an HTTP server” on page 613.
Get-Body-Buffer	Address	Pointer to storage to place the response body contents into.
Get-Body-Buffer-Size	Address	Pointer to a 4-byte signed number field that is the size of the provided body buffer. Must be provided if Get-Body-Buffer is specified.
Get-Response-Code	Address	Pointer to 4-byte area to place the HTTP status code from the HTTP DELETE request.
Set-Header-Buffer	Character	String of custom headers that should be included on the HTTP DELETE request. Each header is newline (\n) separated.
Use-Config-File	Character	false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.
log-level	Character	The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stdout. The levels in order of low to high severity are listed: <ul style="list-style-type: none"> • DEBUG means all logging messages are written to. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.

Optional Parameters		
NAME	Type	Description
web-toolkit-logging	Character	<p>false means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should not be written.</p> <p>true means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should be written.</p> <p>This value will override any default or value in the config.json file. Logging messages are written to stdout.</p>
CDA_Session	Address	Specifies the address of a 12-byte fields that was filling in as part of a gdkinit() CPI call. This indicates a session is current and the config file, keyfile, and provider file will not be read again during this API call.
prefix	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_PREFIX substitution text in a URL_PARM request parameter.
delimiter	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_DELIMITER substitution text in a URL_PARM request parameter.
versionID	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_VERSION_ID substitution text in a URL_PARM request parameter.
logOutput	Address	Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.

Optional Parameters		
NAME	Type	Description
providerFile	Address	Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.

Related services

- [“GDKWRITE — Write an object to cloud storage” on page 728](#)
- [“GDKDEL — Delete an object from cloud storage” on page 742](#)
- [“GDKLIST — List cloud objects” on page 748](#)

Usage notes

- When the API is invoked, if UserID wasn't provided in the optionalParms, the current user's RACF ID is used to retrieve the applicable cloudProvider definitions, as well as retrieve the appropriate security credentials for the User/Cloud Provider/Resource combination from the gdkkeyf.json document.
- When retrieving the cloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment in order to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory does not exist or the provider file is not found there, then the CDA System Default directory of /usr/lpp/dfsms/gdk/providers/ will be used.
- When retrieving the security credentials to use to communicate with the cloudProvider specified on the API call, the gdkkeyf.json file must first be located. The user's RACF ID, or UserID from the optionalParms, is used to find the home directory. That home directory is examined for a gdk/gdkkeyf.json document that can be accessed for READ. If not found, then the CDA API call fails.
- Once the keyfile has been found, it is parsed, looking for an entry associated with the user's RACF ID, or UserID from the optionalParms. The most specific bucket/container name (Resource) is searched for first. A bucket name is the first characters surrounded by a forward slash character (for example, / bucket_name/). If no entry is found, then the generic / Resource name is searched for.
- With the most appropriate entry found, the security credentials will be decrypted using ICSF. The decryption key is expected to be found using a keylabel of GDK.<cloudProvider>.<UserID/RACF ID>.<sequence_number>. If RACF protection of the ICSF key labels is being used, the invoker of the API must have authority to the key label.
- This API is supported on z/OS 2.4 and higher.

Restrictions

None.

Authorization

User.

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 141. Return and reason codes for the GDKDEL service		
Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
100	GDK_UNABLE_TO_READ_KEYFILE	The gdkkeyf.json document was not found in the CDA System directory.
102	GDK_UNABLE_TO_PARSE_KEYFILE	When parsing the gdkkeyf.json document, it is not a valid JSON document.
104	GDK_CLOUD_PROVIDER_NOT_FOUND	When reading the gdkkeyf.json document, for the requested user, the specified cloudProvider was not found.
105	GDK_NO_RESOURCE_FOUND_IN_KEYFILE	For the specified UserId, and cloudProvider, no credentials entry for / was found.
110	GDK_PROVIDER_OPEN_FAILURE	The JSON document for the requested cloudProvider does not exist, or was found, but cannot be opened for READ.
111	GDK_PROVIDER_SPECIFICATION_INVALID	When parsing the JSON document, it was found to be invalid.
112	GDK_FEATURE_UNSUPPORTED	An unknown contentType was found in the parameterSet.
113	GDK_BUFFER_TOO_SMALL	When decrypting the security credentials, the specified buffer was too small to hold the decrypted credential.
114	GDK_AUTH_INIT_FAILURE	When using the AWS4 authentication model, the appropriate entry in the gdkkeyf.json document for the MVSUserID, cloud provider, and name (resource) is missing either the key, or the secretkey key-value pairs.
115	GDK_COULD_NOT_OPEN_OUTPUT_FILE	Unable to open the requested local file for output.
116	GDK_AUTH_APPLY_FAILURE	An error occurred while applying the authorization parameters.
118	GDK_USER_INFO_NOT_FOUND	When reading the gdkkeyf.json document, the requested user entry was not found.
123	GDK_CONN_NOT_HTTPS	HTTPS was not specified for httpMethod in the provider file.
141	GDK_PARAMETER_ERROR	One of the parameters on the API call was incorrect. Examine the ERROR log message for details on the parameter in error.
142	GDK_EMPTY_PROVIDER_FILE	The specified <cloudProvider>.json file is empty. <cloudProvider> is the provider name specified on the API parameter list.
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.
800	GDK_CONNECTION_FAILED	Unable to connect to the host set in the cloud provider JSON definition.
801	GDK_TOOLKIT_FAILED	Error occurred in the z/OS Client Web Enablement Toolkit HWTHRQST call.

Table 141. Return and reason codes for the GDKDEL service (continued)

Return code	Constant Name	Explanation
900	GDK_OBJECT_NOT_FOUND	HTTP status 404 was returned indicating the object was not found.
901	GDK_ACCESS_DENIED	HTTP status 403 was returned indicating the user is not authorized to access the resource in the desired manner.
902	GDK_OPERATION_NOT_SUPPORTED	When parsing the cloud provider JSON document, the WRITEOBJECT description was not found in the supportedOperations array.
903	GDK_RESPONSE_FORMAT_MISMATCH	The request was successful, but the data returned was in a format that did not match what is expected according to the contentType specified in the responseResults for the action in the provider file.
904	GDK_REQUEST_FAILED	A bad HTTP status (4xx or 5xx) was returned.

GDKLIST – List cloud objects

The GDKLIST API is used to list objects in a bucket for the specified cloudProvider.

Description

The GDKLIST API is used to retrieve a list of buckets, a list of objects in a bucket, or a single object for a cloud provider. When called with a cloud provider and a bucket name and a buffer, an HTTP request is sent to the cloud object storage server to retrieve the information according to the definition in the LISTBUCKETS, LISTOBJECT, and HEADOBJECT entries in the supportedOperations array in the cloud provider json file, is sent and information about the cloud objects returned.

The GDKLIST API is processing can additionally be modified through the optionalParmStruct.

Syntax

```
gdklist (returnCodeAddr,
        cloudProvider,
        bucketName,
        buffer,
        bufferLen,
        optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProvider

Specifies the address of a pointer to a name of the cloud provider that this request should contact. The name must be null-terminated. When retrieving the cloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file exists, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory does not exist or the <cloudProvider>.json file is not found there, then the CDA system default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

The application may request a list of all supported providers by using the **GETPROVIDERS** operation.

bucketName

Specifies the address of a pointer to a name that should be listed. The name can be one of the following:

- / - A single forward slash indicates that all bucket names should be returned in the list. This list may be modified by the "filter" optional parameter. (The LISTBUCKETS operation must exist in the provider file.)
- /<bucket_name>/ - A list of all objects within that bucket_name should be returned along with the object size and creation date. This list may be modified by the "prefix" and "delimiter" optional parms. (The requestParameter object for the URL_PARM mechanism, either for GDK_PREFIX or GDK_DELIMITER, must exist in the LISTOBJECT operation.)
- /<bucket_name>/<object_name> - A specific object should be listed and details returned. (The HEADOBJECT operation must exist in the provider file.)

The name passed must be null-terminated. The bucket/container name is folded to lowercase. The name is not checked for valid URL characters. For more information, see [RFC 3986 \(www.ietf.org/rfc/rfc3986.txt\)](http://www.ietf.org/rfc/rfc3986.txt).

When interacting with a provider using the AWS4 authentication model, some characters are encoded as follows:

Character	Encoded version
/	%2F
"encode": "special" in provider file	
!	%21
*	%2A
(%28

buffer

Specifies the address of a pointer to a buffer where returned data should be placed. The buffer is formatted as described by the GDK_LISTOBJECTS_TYPE structure as detailed:

Table 142. GDK_LISTOBJECTS_TYPE				
GDK_LISTOBJECTS_TYPE				
Offset	Len	Type	Name	Description
0		STRUCTURE	GDK_LISTOBJECTS_TYPE	Structure of buffer containing LIST results.
0	4	FIXED	GDK_NumberOfObjects	Number of GDK_OBJECTDESCRIPTION_TYPE entries in the buffer.
4	*	ARRAY	GDK_ObjectDescArray	Array of GDK_OBJECTDESCRIPTION_TYPE entries

Table 143. GDK_OBJECTDESCRIPTION_TYPE				
GDK_OBJECTDESCRIPTION_TYPE				
Offset	Len	Type	Name	Description
0		STRUCTURE	GDK_OBJECTDESCRIPTION_TYPE	Structure of One LIST result.

Table 143. GDK_OBJECTDESCRIPTION_TYPE (continued)				
GDK_OBJECTDESCRIPTION_TYPE				
Offset	Len	Type	Name	Description
0	4	ADDRESS	GDK_objectName	Pointer to a null-terminated object name.
4	4	FIXED	createdTime	Number of seconds since January 1, 1970 when Object was created or -1 if unset.
8	4	FIXED	modifiedTime	Number of seconds since January 1, 1970 when Object was modified.
12	4	FIXED	objectSize	Number of seconds since January 1, 1970 when Object was modified.
16	4	FIXED	isDirectory	1 = pseudo directory

The GDKLIST API will process the response body according to the LISTOBJECT operation's contentType and specified schema detailing the structure of the response. If contentType is application/json, it will be processed as a json document. If contentType is application/xml, it will be processed as an XML document.

Additionally, the provider definition may specify "contentType": text/plain in the responseResults section of the LISTOBJECT operation. This will cause the buffer to be filled with the raw data returned from the Cloud Provider. Additionally, in this case, the return code will be set to GDK_LIST_RAW (300) to indicate that the buffer is filled with raw data and is not mapped by the structures above.

bufferLen

Specifies the address of a 4-byte signed field that contains the length of the passed buffer.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Table 144. Optional parameters		
Optional Parameters		
NAME	Type	Description
UserID	8-byte char	RACF UserID used to retrieve cloud security credentials for the DELETE request.
Get-HWTH-Code	Address	Specifies the address of a 4-byte Integer field where the resulting z/OS Client Web Enablement Toolkit return code should be stored. For more information, see “HWTHRQST — Send a request to an HTTP server” on page 613 .

Table 144. Optional parameters (continued)

Optional Parameters		
NAME	Type	Description
Get-HWTH-Diag	Address	Specifies the address of an HWTH_DIAGAREA_TYPE area where the resulting z/OS Client Web Enablement Toolkit diagnostic information should be stored. For more information, see “HWTHRQST – Send a request to an HTTP server” on page 613.
Get-Header-Buffer	Address	Pointer to storage area to place the headers from the HTTP Response.
Get-Header-Buffer-Size	Address	Pointer to a 4-byte signed number field that is the size of the provided buffer to contain the headers. Must be provided if Get-Header-Buffer is specified.
Set-Header-Buffer	Character	String of custom headers that should be included on the HTTP DELETE request. header is newline (\n) separated.
Use-Config-File	Character	false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.
log-level	Character	<p>The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stdout. The levels in order of low to high severity are listed:</p> <ul style="list-style-type: none"> • DEBUG means all logging messages are written. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.

Table 144. Optional parameters (continued)

Optional Parameters		
NAME	Type	Description
web-toolkit-logging	Character	<p>false means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should not be written.</p> <p>true means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should be written.</p> <p>This value will override any default or value in the config.json file. Logging messages are written to stdout.</p>
prefix	Address	Specifies the address of a null-terminated string that is the common object prefix that should be returned. This value would be placed on the PREFIX parameter of the request url. This is only done if the requestParameters object for the LISTOBJECTS operation has a "URL_PARM" mechanism. When prefix is specified, but a URL_PARM for prefix is not found, an error will be returned.
delimiter	Address	Specifies the address of a single char to be used on the delimiter parameter of the request url. This value would be placed on the DELIMITER parameter of the request url. This is only done if the requestParameters object for the LISTOBJECTS operation has a "URL_PARM" mechanism. When delimiter is specified, but a URL_PARM for delimiter is not found, and error will be returned.
filter	Address	Specifies the address of a null-terminated string that should be used to compare with the beginning of each bucket name returned. If the entire filter matches the beginning of the bucket name, the name is included in the returned results.
CDA_Session	Address	Specifies the address of a 12-byte field that was filled in as part of a gdkinit() CPI call. This indicates a session is current and the config file, keyfile, and provider file will not be read again during this API call.
marker	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_MARKER substitution text in a URL_PARM request parameter.
versionID	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_VERSION_ID substitution text in a URL_PARM request parameter.

Table 144. Optional parameters (continued)

Optional Parameters		
NAME	Type	Description
logOutput	Address	Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.
providerFile	Address	Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.

Related services

- [“GDKGET — Retrieve a cloud object” on page 716](#)
- [“GDKWRITE — Write an object to cloud storage” on page 728](#)
- [“GDKDEL — Delete an object from cloud storage” on page 742](#)
- [“GDKLISTL — List cloud objects” on page 755](#)

Usage notes

- When the API is invoked, if UserID wasn't provided in the optionalParms, the current user's RACF ID is used to retrieve the applicable cloudProvider definitions, as well as retrieve the appropriate security credentials for the User/Cloud Provider/Resource combination from the gdkkeyf.json document.
- When retrieving the cloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment in order to retrieve the home directory. That home directory is examined for a gdk/providers/sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to

communicate with the cloud provider. If the gdk/providers/ directory does not exist or the provider file is not found there, then the CDA System Default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

- When retrieving the security credentials to use to communicate with the cloudProvider specified on the API call, the gdkkeyf.json file must first be located. The user's RACF ID, or UserID from the optionalParms, is used to find the home directory. That home directory is examined for a gdk/gdkkeyf.json document that can be accessed for READ. If not found, then the CDA API call fails.
- Once the keyfile has been found, it is parsed, looking for an entry associated with the user's RACF ID, or UserID from the optionalParms. The most specific bucket/container name (Resource) is searched for first. A bucket name is the first characters surrounded by a forward slash character (for example, /bucket_name/). If no entry is found, then the generic / Resource name is searched for.
- With the most appropriate entry found, the security credentials will be decrypted using ICSF. The decryption key is expected to be found using a keylabel of GDK.<cloudProvider>.<UserID/RACF ID>.<sequence_number>. If RACF protection of the ICSF key labels is being used, the invoker of the API must have authority to the key label.
- This API is supported on z/OS 2.4 and higher.

Restrictions

None.

Authorization

User.

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 145. Return and reason codes for the GDKLIST service		
Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
96 (60)	GDKLISTL	NO
100	GDK_UNABLE_TO_READ_KEYFILE	The gdkkeyf.json document was not found in the CDA System directory.
102	GDK_UNABLE_TO_PARSE_KEYFILE	When parsing the gdkkeyf.json document, it is not a valid JSON document.
104	GDK_CLOUD_PROVIDER_NOT_FOUND	When reading the gdkkeyf.json document, for the requested user, the specified cloudProvider was not found.
105	GDK_NO_RESOURCE_FOUND_IN_KEYFILE	For the specified UserId, and cloudProvider, no credentials entry for / was found.
110	GDK_PROVIDER_OPEN_FAILURE	The JSON document for the requested cloudProvider does not exist, or was found, but cannot be opened for READ.
111	GDK_PROVIDER_SPECIFICATION_INVALID	When parsing the JSON document, it was found to be invalid.
112	GDK_FEATURE_UNSUPPORTED	An unknown contentType was found in the parameterSet.
113	GDK_BUFFER_TOO_SMALL	When decrypting the security credentials, the specified buffer was too small to hold the decrypted credential.
114	GDK_AUTH_INIT_FAILURE	When using the AWS4 authentication model, the appropriate entry in the gdkkeyf.json document for the MVSUserID, cloud provider, and name (resource) is missing either the key, or the secretkey key-value pairs.
116	GDK_AUTH_APPLY_FAILURE	An error occurred while applying the authorization parameters.

Table 145. Return and reason codes for the GDKLIST service (continued)

Return code	Constant Name	Explanation
118	GDK_USER_INFO_NOT_FOUND	When reading the gdkkeyf.json document, the requested user entry was not found.
123	GDK_CONN_NOT_HTTPS	HTTPS was not specified for httpMethod in the provider file.
141	GDK_PARAMETER_ERROR	One of the parameters on the API call was incorrect. Examine the ERROR log message for details on the parameter in error.
142	GDK_EMPTY_PROVIDER_FILE	The specified <cloudProvider>.json file is empty. <cloudProvider> is the provider name specified on the API parameter list.
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.
800	GDK_CONNECTION_FAILED	Unable to connect to the host set in the cloud provider JSON definition.
801	GDK_TOOLKIT_FAILED	Error occurred in the z/OS Client Web Enablement Toolkit HWTHRQST call.
900	GDK_OBJECT_NOT_FOUND	HTTP status 404 was returned indicating the object was not found.
901	GDK_ACCESS_DENIED	HTTP status 403 was returned indicating the user is not authorized to access the resource in the desired manner.
902	GDK_OPERATION_NOT_SUPPORTED	When parsing the cloud provider JSON document, the WRITEOBJECT description was not found in the supportedOperations array.
903	GDK_RESPONSE_FORMAT_MISMATCH	The request was successful, but the data returned was in a format that did not match what is expected according to the contentType specified in the responseResults for the action in the provider file.
904	GDK_REQUEST_FAILED	A bad HTTP status (4xx or 5xx) was returned.

GDKLISTL – List cloud objects

The GDKLISTL API is used to list objects in a bucket for the specified cloudProvider.

Description

The GDKLISTL API is used to retrieve a list of objects in a bucket for a cloud provider. When called with a cloud provider and a bucket name and a buffer, an HTTP request is sent to the cloud object storage server to retrieve the information according to the entry in the LISTOBJECT entry in the supportedOperations array in the cloud provider json file, is sent and information about the cloud objects returned. It is different from the GDKLIST API in that it supports reporting object sizes that are larger than 4*1024*1024*1024 (4GiB).

The GDKLISTL API processing can additionally be modified through the optionalParmStruct.

Syntax

```
gdklistl (returnCodeAddr,
         cloudProvider,
         bucketName,
         buffer,
         bufferLen,
         optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProvider

Specifies the address of a pointer to a name of the cloud provider that this request should contact. The name must be null-terminated. When retrieving the cloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file exists, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory does not exist or the <cloudProvider>.json file is not found there, then the CDA system default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

The application may request a list of all supported providers by using the **GETPROVIDERS** operation.

bucketName

Specifies the address of a pointer to a name that should be listed. The name can be one of the following:

- / - A single forward slash indicates that all bucket names should be returned in the list. This list may be modified by the "filter" optional parameter. (The LISTBUCKETS operation must exist in the provider file.)
- /<bucket_name>/ - A list of all objects within that bucket_name should be returned along with the object size and creation date. This list may be modified by the "prefix" and "delimiter" optional parms. (The requestParameter object for the URL_PARM mechanism, either for GDK_PREFIX or GDK_DELIMITER, must exist in the LISTOBJECT operation.)
- /<bucket_name>/<object_name> - A specific object should be listed and details returned. (The HEADOBJECT operation must exist in the provider file.)

The name passed must be null-terminated. The bucket/container name is folded to lowercase. The name is not checked for valid URL characters. For more information, see [RFC 3986 \(www.ietf.org/rfc/rfc3986.txt\)](http://www.ietf.org/rfc/rfc3986.txt).

When interacting with a provider using the AWS4 authentication model, some characters are encoded as follows:

Character	Encoded version
/	%2F
"encode": "special" in provider file	
!	%21
*	%2A
(%28

buffer

Specifies the address of a pointer to a buffer where returned data should be placed. The buffer is formatted as described by the GDK_LISTLARGEOBJECTS_TYPE structure as detailed:

Table 146. GDK_LISTOBJECTS_TYPE				
GDK_LISTLARGEOBJECTS_TYPE				
Offset	Len	Type	Name	Description
0		STRUCTURE	GDK_LISTLARGEOBJECT S_TYPE	Structure of buffer containing LIST results.

Table 146. GDK_LISTOBJECTS_TYPE (continued)				
GDK_LISTLARGEOBJECTS_TYPE				
Offset	Len	Type	Name	Description
0	4	FIXED	GDK_NumberOfObjects	Number of GDK_LARGEOBJECTDESCRIPTION_TYPE entries in the buffer.
4		FIXED	gdklist_ver	Version of gdklist
6		FIXED	_rsrvd	Reserved for future use
8	*	ARRAY	GSK_ObjectDescArray	Array of GDK_LARGEOBJECTDESCRIPTION_TYPE entries

Table 147. GDK_LARGEOBJECTDESCRIPTION_TYPE				
GDK_LARGEOBJECTDESCRIPTION_TYPE				
Offset	Len	Type	Name	Description
0		STRUCTURE	GDK_LARGEOBJECTDESCRIPTION_TYPE	Structure of One LIST result.
0	4	ADDRESS	GDK_objectName	Pointer to a null-terminated object name.
4	4	FIXED	createdTime	Number of seconds since January 1, 1970 when Object was created or -1 if unset.
8	4	FIXED	modifiedTime	Number of seconds since January 1, 1970 when Object was modified.
12	4	FIXED	isDirectory	1 = pseudo directory
16	8	FIXED	objectSize	Size of the object in bytes
24	12	FIXED	list_reserved	Reserved for future use

The GDKLIST API will process the response body according to the LISTOBJECT operation's contentType and specified schema detailing the structure of the response. If contentType is application/json, it will be processed as a json document. If contentType is application/xml, it will be processed as an XML document.

Additionally, the provider definition may specify "contentType": text/plain in the responseResults section of the LISTOBJECT operation. This will cause the buffer to be filled with the raw data returned from the Cloud Provider. Additionally, in this case, the return code will be set to GDK_LIST_RAW (300) to indicate that the buffer is filled with raw data and is not mapped by the structures above.

bufferLen

Specifies the address of a 4-byte signed field that contains the length of the passed buffer.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Table 148. Optional parameters

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	RACF UserID used to retrieve cloud security credentials for the DELETE request.
Get-HWTH-Code	Address	Specifies the address of a 4-byte Integer field where the resulting z/OS Client Web Enablement Toolkit return code should be stored. For more information, see “HWTHRQST — Send a request to an HTTP server” on page 613 .
Get-HWTH-Diag	Address	Specifies the address of an HWTH_DIAGAREA_TYPE area where the resulting z/OS Client Web Enablement Toolkit diagnostic information should be stored. For more information, see “HWTHRQST — Send a request to an HTTP server” on page 613 .
Get-Header-Buffer	Address	Pointer to storage area to place the headers from the HTTP Response.
Get-Header-Buffer-Size	Address	Pointer to a 4-byte signed number field that is the size of the provided buffer to contain the headers. Must be provided if Get-Header-Buffer is specified.
Set-Header-Buffer	Character	String of custom headers that should be included on the HTTP DELETE request. header is newline (\n) separated.
Use-Config-File	Character	false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.
log-level	Character	The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stdout. The levels in order of low to high severity are listed: <ul style="list-style-type: none"> • DEBUG means all logging messages are written. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.

Table 148. Optional parameters (continued)

Optional Parameters		
NAME	Type	Description
web-toolkit-logging	Character	<p>false means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should not be written.</p> <p>true means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should be written.</p> <p>This value will override any default or value in the config.json file. Logging messages are written to stdout.</p>
prefix	Address	Specifies the address of a null-terminated string that is the common object prefix that should be returned. This value would be placed on the PREFIX parameter of the request url. This is only done if the requestParameters object for the LISTOBJECTS operation has a "URL_PARM" mechanism. When prefix is specified, but a URL_PARM for prefix is not found, an error will be returned.
delimiter	Address	Specifies the address of a single char to be used on the delimiter parameter of the request url. This value would be placed on the DELIMITER parameter of the request url. This is only done if the requestParameters object for the LISTOBJECTS operation has a "URL_PARM" mechanism. When delimiter is specified, but a URL_PARM for delimiter is not found, and error will be returned.
filter	Address	Specifies the address of a null-terminated string that should be used to compare with the beginning of each bucket name returned. If the entire filter matches the beginning of the bucket name, the name is included in the returned results.
CDA_Session	Address	Specifies the address of a 12-byte field that was filled in as part of a gdkinit() CPI call. This indicates a session is current and the config file, keyfile, and provider file will not be read again during this API call.
marker	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_MARKER substitution text in a URL_PARM request parameter.
versionID	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_VERSION_ID substitution text in a URL_PARM request parameter.

Table 148. Optional parameters (continued)

Optional Parameters		
NAME	Type	Description
logOutput	Address	Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.
providerFile	Address	Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.

Related services

- [“GDKGET — Retrieve a cloud object” on page 716](#)
- [“GDKWRITE — Write an object to cloud storage” on page 728](#)
- [“GDKDEL — Delete an object from cloud storage” on page 742](#)
- [“GDKLIST — List cloud objects” on page 748](#)

Usage notes

- When the API is invoked, if UserID wasn't provided in the optionalParms, the current user's RACF ID is used to retrieve the applicable cloudProvider definitions, as well as retrieve the appropriate security credentials for the User/Cloud Provider/Resource combination from the gdkkeyf.json document.
- When retrieving the cloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment in order to retrieve the home directory. That home directory is examined for a gdk/providers/sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to

communicate with the cloud provider. If the gdk/providers/ directory does not exist or the provider file is not found there, then the CDA System Default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

- When retrieving the security credentials to use to communicate with the cloudProvider specified on the API call, the gdkkeyf.json file must first be located. The user's RACF ID, or UserID from the optionalParms, is used to find the home directory. That home directory is examined for a gdk/gdkkeyf.json document that can be accessed for READ. If not found, then the CDA API call fails.
- Once the keyfile has been found, it is parsed, looking for an entry associated with the user's RACF ID, or UserID from the optionalParms. The most specific bucket/container name (Resource) is searched for first. A bucket name is the first characters surrounded by a forward slash character (for example, /bucket_name/). If no entry is found, then the generic / Resource name is searched for.
- With the most appropriate entry found, the security credentials will be decrypted using ICSF. The decryption key is expected to be found using a keylabel of GDK.<cloudProvider>.<UserID/RACF ID>.<sequence_number>. If RACF protection of the ICSF key labels is being used, the invoker of the API must have authority to the key label.
- This API is supported on z/OS 2.4 and higher.

Restrictions

None.

Authorization

User.

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 149. Return and reason codes for the GDKLIST service		
Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
100	GDK_UNABLE_TO_READ_KEYFILE	The gdkkeyf.json document was not found in the CDA System directory.
102	GDK_UNABLE_TO_PARSE_KEYFILE	When parsing the gdkkeyf.json document, it is not a valid JSON document.
104	GDK_CLOUD_PROVIDER_NOT_FOUND	When reading the gdkkeyf.json document, for the requested user, the specified cloudProvider was not found.
105	GDK_NO_RESOURCE_FOUND_IN_KEYFILE	For the specified UserId, and cloudProvider, no credentials entry for / was found.
110	GDK_PROVIDER_OPEN_FAILURE	The JSON document for the requested cloudProvider does not exist, or was found, but cannot be opened for READ.
111	GDK_PROVIDER_SPECIFICATION_INVALID	When parsing the JSON document, it was found to be invalid.
112	GDK_FEATURE_UNSUPPORTED	An unknown contentType was found in the parameterSet.
113	GDK_BUFFER_TOO_SMALL	When decrypting the security credentials, the specified buffer was too small to hold the decrypted credential.
114	GDK_AUTH_INIT_FAILURE	When using the AWS4 authentication model, the appropriate entry in the gdkkeyf.json document for the MVSUserID, cloud provider, and name (resource) is missing either the key, or the secretkey key-value pairs.
116	GDK_AUTH_APPLY_FAILURE	An error occurred while applying the authorization parameters.

Table 149. Return and reason codes for the GDKLIST service (continued)

Return code	Constant Name	Explanation
118	GDK_USER_INFO_NOT_FOUND	When reading the gdkkeyf.json document, the requested user entry was not found.
123	GDK_CONN_NOT_HTTPS	HTTPS was not specified for httpMethod in the provider file.
141	GDK_PARAMETER_ERROR	One of the parameters on the API call was incorrect. Examine the ERROR log message for details on the parameter in error.
142	GDK_EMPTY_PROVIDER_FILE	The specified <cloudProvider>.json file is empty. <cloudProvider> is the provider name specified on the API parameter list.
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.
800	GDK_CONNECTION_FAILED	Unable to connect to the host set in the cloud provider JSON definition.
801	GDK_TOOLKIT_FAILED	Error occurred in the z/OS Client Web Enablement Toolkit HWTHRQST call.
900	GDK_OBJECT_NOT_FOUND	HTTP status 404 was returned indicating the object was not found.
901	GDK_ACCESS_DENIED	HTTP status 403 was returned indicating the user is not authorized to access the resource in the desired manner.
902	GDK_OPERATION_NOT_SUPPORTED	When parsing the cloud provider JSON document, the WRITEOBJECT description was not found in the supportedOperations array.
903	GDK_RESPONSE_FORMAT_MISMATCH	The request was successful, but the data returned was in a format that did not match what is expected according to the contentType specified in the responseResults for the action in the provider file.
904	GDK_REQUEST_FAILED	A bad HTTP status (4xx or 5xx) was returned.

GDKMSGTR — Translate a CDA return code into text

The GDKMSGTR API is used to provide human readable text explanation for a particular CDA API return code.

Description

The GDKMSGTR API may be called to translate a GDK return code value into text to provide more immediate understanding of the issue encountered.

Syntax

```
gdkmsgtr (returnCodeAddr,
          CDAReturnCodeVal,
          outputMessageBuff,
          outputMessageBuffLen,
          optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

CDAreturnCodeVal

Specifies the 4-byte integer containing the CDA API return code value to translate.

outputMessageBuff

Specifies the address of a buffer that should be filled with the message text associated with the requested CDA API return code value.

outputMessageBuffLen

Specifies a 4-byte integer field that contains the length of the outputMessageBuff area.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	RACF UserID used to retrieve cloud security credentials for the DELETE request.
Use-Config-File	Character	false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.
log-level	Character	The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stdout. The levels in order of low to high severity are listed: <ul style="list-style-type: none"> • DEBUG means all logging messages are written to. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.
CDA_Session	Address	Specifies the address of a 12-byte field that was filled in as part of a gdkinit() CPI call. This indicates a session is current and the config file, keyfile, and provider file will not be read again during this API call.

Optional Parameters		
NAME	Type	Description
logOutput	Address	Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.
providerFile	Address	Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.

Related services

- [“GDKGET — Retrieve a cloud object” on page 716](#)
- [“GDKWRITE — Write an object to cloud storage” on page 728](#)
- [“GDKDEL — Delete an object from cloud storage” on page 742](#)
- [“GDKLIST — List cloud objects” on page 748](#)

Usage notes

- When the API is invoked, if UserID wasn't provided in the OptionalParms, the current user's RACF ID is used to retrieve the applicable CloudProvider definitions, as well as retrieve the appropriate Security Credentials for the User/Cloud Provider/Resource combination from the gdkkeyf.json document.
- This API is supported on z/OS 2.4 and higher.

Restrictions

None.

Authorization

User.

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDRV.H.

Table 150. Return and reason codes for the GDKMSGTR service		
Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
113	GDK_BUFFER_TOO_SMALL	When attempting to fill the message buffer, the length of the text is longer than the specified message buffer length.
120	GDK_INVALID_RC_VALUE	Either a negative number, or greater than RC 1000, was passed.
121	GDK_UNKNOWN_RC	A return code value was passed that is currently unused.

GDKGETP – List cloud providers

The GDKGETP API is used to retrieve a list of cloud providers into the location specified by cloudProviderBufAddr.

Description

The GDKGETP API is used to retrieve a list of known cloud providers available. When called, a buffer is passed, and the unique names of cloud providers found in the current user's ~/gdk/providers and the /usr/lpp/dfsms/gdk/providers/ directories are returned.

The GDKGETP API processing can additionally be modified through the optionalParmStruct.

Syntax

```
gdkgetp (retCodeAddr,  
         cloudProviderBufAddr,  
         cloudProviderBufLen,  
         optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProviderBufAddr

Specifies the address of a pointer to a buffer where names should be placed. The buffer is formatted as described by the GDK_PROVIDER_NAME_LIST_TYPE structure as detailed below. When retrieving the cloud provider names, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If the directory exists, the names of the files with the .json suffix will be copied into the character arrays in the GDK_PROVIDER_NAME_LIST_TYPE structure. Additionally, the CDA system default directory of /usr/lpp/dfsms/gdk/providers/ will be used and any additional unique names found there will be copied into the character arrays in the GDK_PROVIDER_NAME_LIST_TYPE structure.

Table 151. GDK_PROVIDER_NAME_TYPE				
GDK_PROVIDER_NAME_TYPE				
Offset	Len	Type	Name	Description
0		STRUCTURE	GDK_PROVIDER_NAME_TYPE	Structure of one provider name.
0	20	CHARACTER	GDK_providerName	Null-terminated provider name.

Table 152. GDK_PROVIDER_NAME_LIST_TYPE				
GDK_PROVIDER_NAME_LIST_TYPE				
Offset	Len	Type	Name	Description
0		STRUCTURE	GDK_PROVIDER_NAME_LIST_TYPE	Structure of one optional parameter.
0	4	FIXED	GDK_NumberofProviders	Count the number of names returned.
4	*	ARRAY	GDK_ProviderArray	Array of provider names.

cloudProviderBufLen

Specifies the address of a 4-byte field holding the length of the storage pointed to by cloudProviderBufAddr. If all the provider names found will not fit in the provided storage, an error will be returned, and the buffer will be filled as much as possible.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	RACF UserID used to find <provider>.json files in the ~/gdk/providers/ directory.
Use-Config-File	Character	false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.

Optional Parameters		
NAME	Type	Description
log-level	Character	<p>The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stdout. The levels in order of low to high severity are listed:</p> <ul style="list-style-type: none"> • DEBUG means all logging messages are written to. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.
CDA_Session	Address	<p>Specifies the address of a 12-byte field that was filled in as part of a gdkinit() CPI call. This indicates a session is current and the config file, keyfile, and provider file will not be read again during this API call.</p>
logOutput	Address	<p>Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.</p>
providerFile	Address	<p>Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.</p>

Optional Parameters		
NAME	Type	Description
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.

Related services

- [“GDKGET — Retrieve a cloud object” on page 716](#)
- [“GDKWRITE — Write an object to cloud storage” on page 728](#)
- [“GDKDEL — Delete an object from cloud storage” on page 742](#)
- [“GDKLIST — List cloud objects” on page 748](#)

Usage notes

- When the API is invoked, if UserID wasn't provided in the OptionalParms, the current user's RACF ID is used to retrieve the CloudProvider names.
- When retrieving the CloudProvider definition names, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment in order to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists, the <cloudProvider>.json is used as the template to return the names. Additionally, the CDA system default directory of /usr/lpp/dfsms/gdk/providers/ is examined for non-duplicate names to be added to the list returned.
- This API is supported on z/OS 2.4 and higher.

Restrictions

None.

Authorization

User.

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 153. Return and reason codes for the GDKGETP service		
Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
113	GDK_BUFFER_TOO_SMALL	When decrypting the security credentials, the specified buffer was too small to hold the decrypted credential.
141	GDK_PARAMETER_ERROR	One of the parameters on the API call was incorrect. Examine the ERROR log message for details on the parameter in error.
142	GDK_EMPTY_PROVIDER_FILE	The specified <cloudProvider>.json file is empty. <cloudProvider> is the provider name specified on the API parameter list.

Table 153. Return and reason codes for the GDKGETP service (continued)

Return code	Constant Name	Explanation
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.

GDKKEYSR – Retrieve cloud credentials

The GDKKEYSR API is used to retrieve the stored cloud credentials for a user associated with a cloud provider and resource.

Description

The GDKKEYSR API is used to retrieve the cloud credentials for the specified cloud provider and resource. When called, the key file for the user will be read and the encrypted cloud credentials matching the request will be decrypted and returned.

The GDKKEYSR API processing can additionally be modified through the optionalParmStruct.

Syntax

```
gdkkeysr (retCodeAddr,
          cloudProvider,
          resource,
          keyValue,
          keyValueBufLen,
          secretKeyValue,
          secretKeyValueBufLen,
          tenantValue,
          tenantValueBufLen,
          useridValue,
          useridValueBufLen,
          passwordValue,
          passwordValueBufLen,
          accessUrlValue,
          accessUrlValueBufLen,
          optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProviderBufAddr

Specifies the address of a pointer to a buffer where names should be placed. The buffer is formatted as described by the GDK_PROVIDER_NAME_LIST_TYPE structure as detailed below. When retrieving the cloud provider names, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If the directory exists, the names of the files with the .json suffix will be copied into the character arrays in the GDK_PROVIDER_NAME_LIST_TYPE structure. Additionally, the CDA System Default directory of /usr/lpp/dfsms/gdk/providers/ will be used and any additional unique names found there will be copied into the character arrays in the GDK_PROVIDER_NAME_LIST_TYPE structure.

resource

Specifies the address of a pointer to the resource or bucket associated with the saved cloud credentials. The generic resource is the forward slash character (/). It's possible that specific bucket or resource names have their own cloud credentials.

keyValue

Specifies a pointer to a buffer that will be filled with the decrypted access key ID if one exists in the keyfile. The access key will be null-terminated.

keyValueBufLen

Specifies a 4-byte integer value that indicates the amount of buffer space that is available for storing the keyValue.

secretKeyValue

Specifies a pointer to a buffer that will be filled with the decrypted secret access key ID if one exists in the keyfile. The secret access key will be null-terminated.

secretKeyValueBufLen

Specifies a 4-byte integer value that indicates the amount of buffer space that is available for storing the secretKeyValue.

tenantValue

Specifies a pointer to a buffer that will be filled with the decrypted tenant if one exists in the keyfile. The tenant name will be null-terminated.

tenantValueBufLen

Specified a 4-byte integer value that indicates the amount of buffer space that is available for storing the tenantValue.

useridValue

Specifies a pointer to a buffer that will be filled with the decrypted userid name if one exists in the keyfile. The userid name will be null-terminated.

useridValueBufLen

Specifies a 4-byte integer value that indicates the amount of buffer space that is available for storing the useridValue.

passwordValue

Specifies a pointer to a buffer that will be filled with the decrypted password value if one exists in the keyfile. The password value will be null-terminated.

passwordValueBufLen

Specifies a 4-byte integer value that indicates the amount of buffer space that is available for storing the passwordValue.

accessUrlValue

Specifies a pointer to a buffer that will be filled with the access URL value if one exists in the keyfile. The access URL text will be null-terminated.

accessUrlValueBufLen

Specifies a 4-byte integer value that indicates the amount of buffer space that is available for storing the accessUrlValue.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	RACF UserID used to retrieve cloud security credentials for the GET request.
Use-Config-File	Character	false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.

Optional Parameters		
NAME	Type	Description
log-level	Character	<p>The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stdout. The levels in order of low to high severity are listed:</p> <ul style="list-style-type: none"> • DEBUG means all logging messages are written to. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.
logOutput	Address	<p>Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.</p>
providerFile	Address	<p>Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.</p>
keyFile	Address	<p>Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.</p>

Optional Parameters		
NAME	Type	Description
CDA_Session	Address	Specifies the address of a 12-byte field that was filled in as part of a gdkinit() CPI call. This indicates a session is current and the config file, keyfile, and provider file will not be read again during this API call.
Check-refresh-keyfile	Character	String value “true” indicates DFSMSdfp CDA should compare the modification time of the user’s gdkkeyf.json file with the saved time of keyfile when it was last loaded for the CDA session. If the times do not match, the current gdkkeyf.json file will be re-loaded.

Related services

- [“GDKKEYAD — Store cloud credentials” on page 773](#)
- [“GDKKEYGR — List resources for provider” on page 781](#)
- [“GDKKEYDL — Delete cloud credentials” on page 778](#)

Usage notes

- When the API is invoked, if UserID wasn’t provided in the OptionalParms, the current user's RACF ID is used to retrieve the appropriate security credentials for the User/Cloud Provider/Resource combination from the gdkkeyf.json document.
- The current user (invoking the API) must have READ access to the gdkkeyf.json file. Additionally, the current RACF user must have READ access to the keylabel (GDK.<userid>.<provider>.*) in the CSFKEYS class. This applies even when UserID is different from the current user's RACF ID, which would happen if UserID is provided in the OptionalParms.
- This API is supported on z/OS 2.4 and higher.

Restrictions

None.

Authorization

User.

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 154. Return and reason codes for the GDKKEYSR service		
Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
100	GDK_UNABLE_TO_READ_KEYF ILE	The gdkkeyf.json document was not found in the CDA System directory.
102	GDK_UNABLE_TO_PARSE_KEY FILE	When parsing the gdkkeyf.json document, it is not a valid JSON document.
103	GDK_MALFORMED_KEYFILE	The object in the keyfile for a particular cloud provider is not a JSON array.

Table 154. Return and reason codes for the GDKKEYSR service (continued)

Return code	Constant Name	Explanation
104	GDK_CLOUD_PROVIDER_NOT_FOUND	When reading the gdkkeyf.json document, for the requested user, the specified cloudProvider was not found.
105	GDK_NO_RESOURCE_FOUND_IN_KEYFILE	For the specified UserId, and cloudProvider, no credentials entry for / was found.
113	GDK_BUFFER_TOO_SMALL	When decrypting the security credentials, the specified buffer was too small to hold the decrypted credential. Contact IBM support.
114	GDK_AUTH_INIT_FAILURE	When using the AWS4 authentication model, the appropriate entry in the gdkkeyf.json document for the MVSUserID, cloud provider, and name (resource) is missing either the key, or the secretkey key-value pairs.
118	GDK_USER_INFO_NOT_FOUND	When reading the gdkkeyf.json document, the requested user entry was not found.
122	GDK_ICSF_ERROR	An error occurred in the ICSF API. With LOG(ERROR) in effect, more information is provided as to the specific ICSF error.
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.

GDKKEYAD – Store cloud credentials

The GDKKEYAD API is used to store cloud credentials securely for usage by other API calls.

Description

The GDKKEYAD API is used to store cloud credentials for the specified cloud provider and resource. When called, the passed cloud credentials are encrypted by ICSF, using a newly generated AES256 data key. The new data key is stored with ICSF under a keylabel of:

```
GDK.<userid>.<provider>.<identifier>
```

where:

- userid is the current RACF user's ID
- provider is the specified cloud provider in the user's ~/gdk/providers/ directory or system default directory off /usr/lpp/dfsms/gdk/providers/.
- identifier is a CDA used identifier used internally

For more information, see [Chapter 27, “Cloud Data Access cloud credential storage,”](#) on page 701.

Storing the encrypted credentials. The keyfile in the current user's ~/gdk/ directory is read, and a new entry is created, or an existing entry is overwritten for the specified cloud provider and resource.

The GDKKEYAD API processing can additionally be modified through the optionalParmStruct.

Syntax

```
gdkkeyad (retCodeAddr,  
          cloudProvider,  
          resource,  
          keyValue,  
          secretKeyValue,  
          tenantValue,  
          useridValue,  
          passwordValue,  
          accessUrlValue,  
          optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProvider

Specifies the address of a pointer to a name of the cloud provider that the associated cloud credentials are to be retrieved. The name must be null-terminated. When processing the request, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment to retrieve the home directory. That home directory is examined for a gdk/gdkkeyf.json key file. The keyfile will be searched for entries corresponding to the requested cloud provider name.

The application may request a list of all supported providers by using the **GETPROVIDERS** operation.

accessUrlValue

Specifies the address of a pointer to a buffer that will be filled with the decrypted access URL value if one exists in the keyfile. The access URL text will be null-terminated.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	RACF UserID used to retrieve cloud security credentials for the GET request.
Use-Config-File	Character	false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.

Optional Parameters		
NAME	Type	Description
log-level	Character	<p>The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stdout. The levels in order of low to high severity are listed:</p> <ul style="list-style-type: none"> • DEBUG means all logging messages are written to. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.
auth-type	Character	Authentication type used. OAUTH_2, AWS4, BASIC, TEMPAUTH, and KEYSTONE are reserved values.
credentials_file	Character	Absolute path to a credentials JSON file that contains information such as a private RSA key as well as issuer and authorization URL fields.
logOutput	Address	Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.
providerFile	Address	Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.

Optional Parameters		
NAME	Type	Description
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.

keyValue

Specifies the address of a pointer to a buffer that contains the null-terminated Access Key ID. This Access Key ID is used during requests to a cloud object store that uses the S3 API.

secretKeyValue

Specifies the address of a pointer to a buffer that contains the null-terminated Secret Access Key ID. This Secret Access Key ID is used during requests to a cloud object store that uses the S3 API.

tenantValue

Specifies the address of a pointer to a buffer that contains the null-terminated tenant name.

useridValue

Specifies the address of a pointer to a buffer that contains the null-terminated user ID value.

passwordValue

Specifies the address of a pointer to a buffer that contains the null-terminated password value.

Related services

- [“GDKKEYDL — Delete cloud credentials” on page 778](#)
- [“GDKKEYSR — Retrieve cloud credentials” on page 769](#)
- [“GDKKEYGR — List resources for provider” on page 781](#)

Usage notes

- When the API is invoked, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment in order to retrieve the home directory. That home directory is examined for a gdk/ sub-directory. The gdkkeyf.json file must first be located.
- Once the keyfile has been found, it is parsed, looking for an entry associated with the user's RACF ID, or UserID from the optionalParms. The most specific bucket/container name (Resource) is searched for first. If no entry is found, then the generic / Resource name is searched for.
- With the most appropriate entry found, the security credentials will be decrypted using ICSF. The decryption key is expected to be found using a keylabel of GDK.<cloudProvider>.<UserID/RACF ID>.<sequence_number>. If RACF protection of the ICSF key labels is being used, the invoker of the API must have authority to the key label.
- This API is supported on z/OS 2.4 and higher.

Authentication methods and associated parameter

The authentication object in the cloud provider json file determines which parameter values are used in an operation.

AWS4

- accessKeyValue
- secretKeyValue

AZURE

- accessKeyValue
- secretKeyValue

BASIC

- accessKeyValue
- secretKeyValue

KEYSTONE

- userIDValue
- passwordValue
- tenant
- accessUrl

OAUTH_2

- tenant_holds (Access_Token, Client_Secret, Credentials_file)
- When Access_Token, secretKeyValue is used to hold the Access token.
- When Client_Secret, AccessKeyValue, and SecretKeyValue is used.
- When Credentials_file, accessUrl holds absolute path to credentials file.

TEMPAUTH

- userIDValue
- passwordValue
- **Optional:** accessUrl (Used, if it exists)

Restrictions

None.

Authorization

User.

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 155. Return and reason codes for the GDKKEYAD service		
Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
100	GDK_UNABLE_TO_READ_KEYFILE	The gdkkeyf.json document was not found in the CDA System directory.
102	GDK_UNABLE_TO_PARSE_KEYFILE	When parsing the gdkkeyf.json document, it is not a valid JSON document.
103	GDK_MALFORMED_KEYFILE	The object in the keyfile for a particular cloud provider is not a JSON array.
104	GDK_CLOUD_PROVIDER_NOT_FOUND	When reading the gdkkeyf.json document, for the requested user, the specified cloudProvider was not found.
105	GDK_NO_RESOURCE_FOUND_IN_KEYFILE	For the specified UserId, and cloudProvider, no credentials entry for / was found.
113	GDK_BUFFER_TOO_SMALL	When decrypting the security credentials, the specified buffer was too small to hold the decrypted credential.

Table 155. Return and reason codes for the GDKKEYAD service (continued)

Return code	Constant Name	Explanation
118	GDK_USER_INFO_NOT_FOUND	When reading the gdkkeyf.json document, the requested user entry was not found.
119	GDK_DECRYPTION_ERROR	An error occurred when decrypting the cloud credentials in the keyfile.
122	GDK_ICSF_ERROR	An error occurred in the ICSF API. With LOG(ERROR) in effect, more information is provided as to the specific ICSF error.
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.

GDKKEYDL — Delete cloud credentials

The GDKKEYDL API is used to remove cloud credentials for the specified cloudProvider and resourceName with additional cleanup.

Description

The GDKKEYDL API is used to remove stored cloud credentials from the keyfile. When called, keyfile for the current user is read and the stored credentials for the specified cloud provider and resource name combination are removed. Additionally, the data encryption key associated with those credentials is also removed from the ICSF CKDS.

The GDKKEYDL API processing can additionally be modified through the optionalParmStruct.

Syntax

```
gdkkeydl (retCodeAddr,
          cloudProvider,
          resourceName,
          optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProvider

Specifies the address of a pointer to a name of the cloud provider that this request is associated with. The name must be null-terminated. When processing the request, the name of the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment to retrieve the home directory. That home directory is examined for a gdk/gdkkeyf.json file.

resourceName

Specifies the address of a pointer to the resource name associated with the cloud credentials to be deleted. The name passed must be null-terminated. The resource name is usually just a forward slash character (/), indicating reference to all buckets should use these cloud credentials. It may be a specific bucket name as well, which would start with the forward slash character.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or

additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	RACF UserID used to retrieve cloud security credentials for the GET request.
Use-Config-File	Character	false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.
log-level	Character	The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stdout. The levels in order of low to high severity are listed: <ul style="list-style-type: none"> • DEBUG means all logging messages are written to. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.
logOutput	Address	Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.

Optional Parameters		
NAME	Type	Description
providerFile	Address	Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.

Related services

- [“GDKKEYAD — Store cloud credentials” on page 773](#)
- [“GDKKEYSR — Retrieve cloud credentials” on page 769](#)
- [“GDKKEYGR — List resources for provider” on page 781](#)

Usage notes

- When removing the Cloud Credentials from the keyfile, the gdkkeyf.json file must first be located. The user's RACF ID, or UserID from the optionalParms, is used to find the home directory. That home directory is examined for a gdk/gdkkeyf.json document that can be accessed for WRITE.
- Once the keyfile has been found, it is parsed, looking for an entry associated with the user's RACF ID, or UserID from the optionalParms. The entry matching the Cloud Provider name, and resource name is found and deleted. Additionally, the encryption key and associated keylabel are deleted from ICSF.
- The GDKKEYGR API may be used to list the resources associated with a particular cloud provider that are found in the gdkkeyf.json file. If a resource is listed, that means that there are cloud credentials stored for that cloud provider and Resource combination.
- This API is supported on z/OS 2.4 and higher.

Restrictions

None.

Authorization

User.

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 156. Return and reason codes for the GDKKEYDL service

Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
100	GDK_UNABLE_TO_READ_KEYFILE	The gdkkeyf.json document was not found in the CDA System directory.
102	GDK_UNABLE_TO_PARSE_KEYFILE	When parsing the gdkkeyf.json document, it is not a valid JSON document.
103	GDK_MALFORMED_KEYFILE	The object in the keyfile for a particular cloud provider is not a JSON array.
104	GDK_CLOUD_PROVIDER_NOT_FOUND	When reading the gdkkeyf.json document, for the requested user, the specified cloudProvider was not found.
105	GDK_NO_RESOURCE_FOUND_IN_KEYFILE	For the specified UserId, and cloudProvider, no credentials entry for / was found.
118	GDK_USER_INFO_NOT_FOUND	When reading the gdkkeyf.json document, the requested user entry was not found.
119	GDK_DECRYPTION_ERROR	An error occurred when decrypting the cloud credentials in the keyfile.

GDKKEYGR – List resources for provider

The GDKKEYGR API is used to retrieve a list of resource names for the requested cloud provider.

Description

The GDKKEYGR API is used to retrieve a list of resource names found in the gdkkeyf.json keyfile for a particular cloud provider. If a resource name is returned, it means the gdkkeyf.json file contains encrypted cloud credentials associated with that cloud provider and resource name.

Syntax

```
gdkkeygr (retCodeAddr,
          cloudProvider,
          resultsBuffer,
          optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProvider

Specifies the address of a pointer to a name of the cloud provider that this request should contact. The name must be null-terminated.

The application may request a list of all supported providers by using the **GETPROVIDERS** operation.

resultsBuffer

Specifies the address of a pointer to a buffer of space that the resource names will be placed into. The buffer contents is mapped by the SEARCH_RESULTS_STRUCT_TYPE and an array of SEARCH_RESULT_ELEMENT_TYPE entries as defined in the gdkic.h header file found in SYS1.SIEAHDR.V.

The API invoker will allocate the buffer, and set the maxResults field to the maximum number of SEARCH_RESULT_ELEMENT_TYPE entries that the buffer can contain. If the size of the allocated space for the buffer is too small for the number of results according to the maxResults field, then an exception may occur and the calling program ended abnormally.

Table 157. SEARCH_RESULT_ELEMENT_TYPE				
SEARCH_RESULT_ELEMENT_TYPE				
Offset	Len	Type	Name	Description
0	100	CHARACTER	resourceName	Resource name.
100	20	CHARACTER	providerName	Cloud provider name as found in gdk/providers/directory (without .json suffix).

Table 158. SEARCH_RESULT_STRUCT_TYPE				
SEARCH_RESULT_STRUCT_TYPE				
Offset	Len	Type	Name	Description
0	4	FIXED	numberOfResults	The number of SEARCH_RESULT_ELEMENT_TYPE entries returned.
4	4	FIXED	macResults	The maximum number of results that will fit in this buffer.
8	*	STRUCTURE	resultElement	Array of SEARCH_RESULT_ELEMENT_TYPE entries to be filled in by the API.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	RACF UserID used to retrieve cloud security credentials for the GETT request.
Use-Config-File	Character	false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.

Optional Parameters		
NAME	Type	Description
log-level	Character	<p>The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stdout. The levels in order of low to high severity are listed:</p> <ul style="list-style-type: none"> • DEBUG means all logging messages are written to. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.
logOutput	Address	<p>Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.</p>
providerFile	Address	<p>Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.</p>
keyFile	Address	<p>Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.</p>

Optional Parameters		
NAME	Type	Description
provAltValue	Character	String indicating the field name from each credential entry whose contents should be returned within the providerName character array within a SEARCH_RESULT_ELEMENT_TYPE. Only the first 19 bytes of the value will be returned.

Related services

- [“GDKWRITE — Write an object to cloud storage” on page 728](#)
- [“GDKDEL — Delete an object from cloud storage” on page 742](#)
- [“GDKLIST — List cloud objects” on page 748](#)

Usage notes

- When the API is invoked, the gdkkeyf.json file must first be located. The user's RACF ID, or UserID from the optionalParms, is used to find the home directory. That home directory is examined for a gdk/gdkkeyf.json document that can be accessed for READ.
- Once the keyfile has been found, it is parsed, looking for all entries for the requested cloud provider name.
- If the keyfile is not found, return code 100, GDK_UNABLE_TO_READ_KEYFILE is returned.
- This API is supported on z/OS 2.4 and higher.

Restrictions

None.

Authorization

User.

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 159. Return and reason codes for the GDKKEYGR service		
Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
100	GDK_UNABLE_TO_READ_KEYFILE	The gdkkeyf.json document was not found in the CDA System directory.
102	GDK_UNABLE_TO_PARSE_KEYFILE	When parsing the gdkkeyf.json document, it is not a valid JSON document.
104	GDK_CLOUD_PROVIDER_NOT_FOUND	When reading the gdkkeyf.json document, for the requested user, the specified cloudProvider was not found.
113	GDK_BUFFER_TOO_SMALL	When filling the resultsBuffer area, there was not enough room to include all resource names Partial results are returned.
118	GDK_USER_INFO_NOT_FOUND	When reading the gdkkeyf.json document, the requested user entry was not found.
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.

GDKINIT – Initialize a Session

Description

The GDKINIT API allows callers to start a 'CDA session' to be used with the other CDA APIs. A session can cross multiple API calls. Once a session has been started, some things will be skipper for subsequent API calls. Those are:

- The config.json file is not re-read.
- The provider file for the session is not re-read.
- The key file for the session is not re-read.
- Once a REST API call has been made, the SSL connection is kept open to be used for subsequent REST API calls to the same cloud.

The GDKTERM API must be called to terminate and cleanup the session once complete.

Even though the key file is not re-read, the ICSF APIs are still used to decrypt the cloud credentials when they are needed to ensure that the cloud credentials are not kept in the clear longer than necessary to perform the requested action.

The GDKINIT API processing can additionally be modified through the optionalParmStruct.

Purpose (or Function): Initialize a Cloud Data Access session that will consist of 0 or more DFSMSdfp CDA API calls.

Requirements: When using the Branch Entry, a Language Environment must be established before the gdkinit() API is invoked. When called from a Language Environment C program, the environment is already created and available. Otherwise a PreInitialization environment can be used to create a Language Environment (CEEPIPI). For more information, see [Using preinitialization services](#) in *z/OS Language Environment Programming Guide*

When linking with SYS1.CSSLIB(GDKCSS), then a Language Environment should already be established. There is no entry in the SYS1.CSSLIB(GDKCSSNL) for GDKINIT.

Syntax

```
gdkinit (retCodeAddr,  
        cloudProvider,  
        session_handle,  
        optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProvider

Specifies the address of a pointer to a name of the cloud provider that all requests for this session will use. The name must be null-terminated. When retrieving the CloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParmStruct, is used to examine the associated OMVS segment to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory does not exists or the <cloudProvider>.json file is not found there, then the CDA system Default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

Subsequent calls to the CDA APIs that specify a Cloud Provider name that communicate with the Cloud Server will ignore the specified name and instead use the Cloud Provider for the session. The Cloud Provider JSON file is only read once per session.

The application may request a list of all support providers by using the **GETPROVIDERS** operation.

session_handle

Specifies the address of a 12 byte field where DFSMSdfp CDA will store a session handle which uniquely identifies the session.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	RACF UserID used to retrieve cloud security credentials for the session
AutoConvert	Character	"true" means that ASCII to EBCDIC translation should be performed on the data being sent or retrieved from the cloud. "false" means that no translation should be performed. This optional parameter will override any default, or value in the config.json file. If this value is passed, this sets the new default for the session. The default may be overridden for individual subsequent API calls, but the override only lasts for that call. If not passed, the default for the session is no translation of data.
Use-Config-File	Character	false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used. On subsequent API calls, the "Use-Config-File" parm is ignored because the default config has been set by the gdkinit() call.

Optional Parameters		
NAME	Type	Description
log-level	Character	<p>This setting sets the default logging level for all subsequent API calls. The default logging level may be overridden for specific API calls via the optionalParms on that call. The default logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stderr. The levels in order of low to high severity are listed:</p> <ul style="list-style-type: none"> • DEBUG means all logging messages are written. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.
web-toolkit-logging	Character	<p>This parameter sets the default value of the web-toolkit-logging for the session. The value for a specific API call may be overridden by the optionalParms of that API call.</p> <p>"false" means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should not be written.</p> <p>"true" means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should be written.</p> <p>This value will override any default or value in the config.json file. Logging messages are written to stdout.</p>

Optional Parameters		
NAME	Type	Description
logOutput	Address	Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.
providerFile	Address	Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.
multi_thread	Address	Specifies the address of a null terminated string that indicates that this CDA session may be performed in parallel with other CDA sessions. Specifying this to be "true" will cause CDA to use the ICSF APIs CSNPTRC, CSNPHMG, and CSNPTRD to compute the SHA-256 HMAC instead of the ICSF PKCS#11 APIs.

Usage notes

When the API is invoked, if UserID wasn't provided in the OptionalParms, the current user's RACF ID is used to retrieve the applicable CloudProvider definitions, as well as read the gdkkeyf.json document for decryption by subsequent API calls. The gdkkeyf.json document will not be re-read for any subsequent API calls that interact with the Cloud Storage server. It may be re-read as needed for the subsequent Key Management API calls.

When retrieving the CloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment in order to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to

communicate with the cloud provider. If the gdk/providers/ directory does not exist or the provider file is not found there, then the CDA System Default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

When retrieving the Security Credentials to use to communicate with the cloudProvider specified on the API call, the gdkkeyf.json file must first be located. The user's RACF ID, or UserID from the optionalParms, is used to find the home directory. That home directory is examined for a gdk/gdkkeyf.json document that can be accessed for READ. If not found, then the CDA API call fails.

Restrictions

None.

Authorization

User.

Related services

- “GDKGET — Retrieve a cloud object” on page 716
- “GDKWRITE — Write an object to cloud storage” on page 728
- “GDKDEL — Delete an object from cloud storage” on page 742
- “GDKLIST — List cloud objects” on page 748

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 160. Return and reason codes for the GDKKEYGR service		
Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
100	GDK_UNABLE_TO_READ_KEYFILE	The gdkkeyf.json document was not found in the CDA System directory.
104	GDK_CLOUD_PROVIDER_NOT_FOUND	When reading the gdkkeyf.json document, for the requested user, the specified cloudProvider was not found.
110	GDK_PROVIDER_OPEN_FAILURE	The JSON document for the requested cloudProvider does not exist, or was not found, but cannot be opened for READ.
141	GDK_PARAMETER_ERROR	One of the parameters on the API call was incorrect. Examine the ERROR log message for details on the parameter in error.
142	GDK_EMPTY_PROVIDER_FILE	The specified <cloudProvider>.json file is empty. <cloudProvider> is the provider name specified on the API parameter list.
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.

GDKTERM — Terminate a Session

Description

The GDKTERM API is called to indicate that a 'CDA session' is complete, and any cleanup should be performed. Any open connection is closed and cleaned up. Any storage obtained across the session is released.

Purpose (or Function): Terminate a Cloud Data Access session that was established via a GDKINIT API call.

Requirements: When using the Branch Entry, a Language Environment must be established before the `gdkterm()` API is invoked. When called from a Language Environment C program, the environment is already created and available. Otherwise a PreInitialization environment can be used to create a Language Environment (CEEPIPI). For more information, see [Using preinitialization services](#) in *z/OS Language Environment Programming Guide*

When linking with `SYS1.CSSLIB(GDKCSS)`, then a Language Environment should already be established. There is no entry in the `SYS1.CSSLIB(GDKCSSNL)` for `GDKTERM`.

Syntax

```
gdkterm (retCodeAddr,  
        cloudProvider,  
        session_handle,  
        optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

session_handle

Specifies the address of a 12 byte field where DFSMSdfp CDA will store a session handle uniquely identifies the session.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
log-level	Character	<p>This setting sets the default logging level for all subsequent API calls. The default logging level may be overridden for specific API calls via the optional Parms on that call. The default logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stderr. The levels in order of low to high severity are listed:</p> <ul style="list-style-type: none"> • DEBUG means all logging messages are written. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.
logOutput	Address	<p>Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.</p>
providerFile	Address	<p>Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.</p>

Optional Parameters		
NAME	Type	Description
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.

Usage notes

None.

Restrictions

None.

Authorization

User.

Related services

- [“GDKWRITE — Write an object to cloud storage” on page 728](#)
- [“GDKDEL — Delete an object from cloud storage” on page 742](#)
- [“GDKLIST — List cloud objects” on page 748](#)

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 161. Return and reason codes for the GDKKEYGR service		
Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
141	GDK_PARAMETER_ERROR	One of the parameters on the API call was incorrect. Examine the ERROR log message for details on the parameter in error.
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.

GDKGEN — Perform a Configurable request

Description

The GDKGEN API will allow callers to request that CDA perform an operation found in the supportedOperations array in the specified provider file. Having a generic (configurable) API will allow CDA to perform other operations such as CreateBucket, DeleteBucket, GetObjectACL, etc.

When called, the provider definition for the requested cloud provider will be retrieved and the security credentials for the invoking user will be retrieved. Those security credentials will be used to send an HTTP request as described by the named Operation.

The GDKGEN API processing can additionally be modified through the optionalParmStruct.

Purpose (or Function): Perform a named operation found in the provider file supportedOperations array. The API allows for invocation of a configurable request, where the programmer knows the name of the operation in the provider file, as well as what inputs and outputs are expected for that particular operation.

Requirements: When using the Branch Entry, a Language Environment must be established before the gdkterm() API is invoked. When called from a Language Environment C program, the environment is already created and available. Otherwise a PreInitialization environment can be used to create a Language Environment (CEEPIPI). For more information, see [Using preinitialization services](#) in *z/OS Language Environment Programming Guide*

When linking with SYS1.CSSLIB(GDKCSS), then a Language Environment should already be established. When linking with SYS1.CSSLIB(GDKCSSNL), then a Language Environment does not need to be established as it will be created on entry, and terminated at exit.

Syntax

```
gdkgen (retCodeAddr,
        cloudProvider,
        objectName
        operation,
        genParmStructPtr,
        optionalParmStructPtr);
```

When using Branch Entry, and a Language Environment is not established, the GDKGENN API may be used. This will create a Language Environment upon entry and terminate the Language Environment upon exit.

```
gdkgenn (retCodeAddr,
          cloudProvider,
          objectName
          operation,
          genParmStructPtr,
          optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProvider

Specifies the address of a pointer to a name of the cloud provider that this request should contact. The name must be null-terminated. When retrieving the CloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory does not exist or the <cloudProvider>.json file is not found there, then the CDA System Default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

The application may request a list of all supported providers by using the **GETPROVIDERS** API.

objectName

Specifies the address of a pointer to the remote object name to be used during processing. The remote object name is referenced as a whole as GDK_OBJECT in the provider file. It is further broken down into to variables referenced as GDK_OBJECT_PART and GDK_BUCKET_PART, where GDK_BUCKET_PART is the characters between the first forward slash, and the second forward slash.

This value must specify the complete remote cloud provider location description including bucket/container and/or path and/or file name that is the target of this request (e.g /bucket2/dir1/CDA.pdf). The name must start with a forward-slash (/), and the bucket/container name is the characters up to the next forward-slash (/). The name passed must be null-terminated. The bucket/container name is folded to lowercase. The name is not checked in general for valid URL characters. see: [RFC 3986 \(www.ietf.org/rfc/rfc3986.txt\)](http://www.ietf.org/rfc/rfc3986.txt) When interacting with a provider using the AWS4 authentication model, some characters are encoded as follows:

Table 162.	
Character	Encoded Version
/	%2F
"encode": "special" in provider file	
!	%21
*	%2A
(%28
"encodeURIComponent" in provider file will cause the specified characters to be URL encoded	

operation

Specifies the address of a pointer to the name of the marching operation from the supportedOperations array in the provider file. The name is folded to uppercase before attempting to match with an operation in the supportedOperations array. The name must be null-terminated.

genParmStructPtr

Specifies additional information or data that is intended to be used by DFSMSdfp CDA during the processing of the requested operation. The operation as defined in the provider file may require some data as identified by the <var_name> syntax in the provider file. e.g. <GDK_DATA> identifies a pointer to a buffer of data to be sent to the Cloud Server.

In a single GDK_GEN_TYPE entry, the field are as follows:

- GDK_genKey is a pointer to a null terminated string of the variable name that this entry describes
- GDK_genValue is a pointer to storage that is defined via the GDK_gen Type and GDK_genLen values.
- GDK_genType is 2-byte unsigned number that indicates what GDK_genValue points to. I.e a 4 byte integer, or a character array of EBCDIC data.
- GDK_genLen is a 2-byte unsigned number that indicates the length of the storage that GDK_genValue points to. Before calling the GDKGEN() API, this field should be initialized to the amount of storage available. After the GDKGEN() API call returns, it will contain the amount of storage used for the return value. For example: A GDK_genLen of 4 paired with GDK_genType of GDK_GEN_INT means that GDK_genValue points to a 4-byte integer value.

genericParmsStruct Control Block Mapping

```

/* ----- */
/* genericParmsStruct Type Definition (GDKGEN service) */
/* - maps the data pointed to by the genParmStructPtr */
/* parameter */
/* ----- */
typedef void * GDK_GEN_VALUE_TYPE; /* Generic pointer to value */

typedef struct {
    GDK_OPTION_KEY_TYPE GDK_genKey; /* Key name pointer */
    GDK_GEN_VALUE_TYPE GDK_genValue; /* Value pointer */
    unsigned short GDK_genType; /* Type of value */
    unsigned short GDK_genLen; /* Length of value area */
} GDK_GEN_TYPE; /* Single Generic Entry */

typedef struct {
    uint32_t GDK_NumOfGenTypes; /* Count of Generic Entries */
    GDK_GEN_TYPE GDK_GenArray[]; /* Array of GDK_GEN-TYPES */
}

```

```

} GDK_GEN_PARMS_TYPE:          /* Header of structure          */
#define GDK_GEN_STRING 0;      /* String type (default)    */
#define GDK_GEN_INT 1;         /* Integer type             */
#define GDK_GEN_ADDR 2;        /* Address type             */
#define GDK_GEN_FLOAT 3;       /* Floating point type       */

```

The name of the GDK_Gen_Key may be selected as preferred. When following the rules defined in the provider file, the key name will be used. There are some CDA internal variables that are used internally and if specified, may not be honored if passed via the genericParmsStruct.

Table 163.	
CDA Variable Name	Description
GDK_DATA_LOCATION	Address of the thing where the data is read from or written to. Requires GDK_DATA_LOCATION_TYPE to also be passed to describe what is pointed to.
GDK_DATA_LOCATION_TYPE	<p>Integer. Describes the thing that GDK_DATA_LOCATION points to.</p> <ol style="list-style-type: none"> 1. Buffer area. GDK_DATA_LOCATION is a 31-bit address of an area to read data from, or write data to. GDK_DATA_LOCATION_LEN is the length of data in the buffer when reading, or the maximum size of the buffer. 2. UNIX file Path or Data Set name. GDK_DATA_LOCATION is a 31-bit address that points to the name of a z/OS UNIX data set, or data set name that contains the data to read, or should be written to with data received. GDK_DATA_LOCATION_LEN is the length of the null-terminated path name. Name must conform to conditions described in the z/OS XL C/C++ Programming Guide Input and Output chapter, performing OS I/O operations → Opening Files → Using fopen() or freopen(). <ul style="list-style-type: none"> • Unix file absolute path. Begins with /, and includes all directories between root and the file name. i.e. /u/user1/file.txt • Data Set name. Begins with //'DATASET.NAME' 3. Streaming Exit pointer. GDK_DATA_LOCATION is a 31-bit address that points to executable code that conforms to the z/OS Client for Web Enablement Toolkit streaming exit expectations. GDK_DATA_LOCATION_LEN must be the value 4, indicating it is a 31-bit address.
GDK_DATA_LOCATION_LEN	4 byte Unsigned integer. Describes the length of the data that GDK_DATA_LOCATION points to.
GDK_READ_DATA_LEN_PTR	Address of a 4-byte unsigned integer. On an operation that has read data from the Cloud Object Store, the amount of data read will be placed in this integer.

Table 164.

CDA Variable Name	Description
GDK_DATA	Pointer to data buffer used in operation
GDK_DATA_LEN	Length of data in buffer for WRITE type operations
DATA_ISO_8601	GMT current timestamp in ISO 8601 format
AZURE_ACCOUNT	Account name from saved Cloud Credentials
DATE_GMT	GMT current timestamp
GDK_OBJECT_NAME	The remote object name is referenced as a whole. (Bucket name and object name.)
GDK_PART	Part number from a multipart upload
GDK_GMT_UNIX_TIME	GMT current timestamp in number of seconds since January 1, 1970
GDK_JWT_EXP	JWT Expiration time calculated from the GDK_GMT_UNIX_TIME + JWT_JWT_DURATION value from provider file
HOST	Host value from provider file
PARAMETER_SET	Used internally by DFSMSdfp CDA for collecting requestParameter from the provider file
UPLOADID	Used internally by DFSMSdfp CDA for tracking the Upload ID during a Multi-Part upload
GDK_ETAG	Used internally by DFSMSdfp CDA for tracking the returned e Tag during a multi-part upload.
GDK_PREFIX	Set from the "prefix" optional parm, and used in URL_PARM request parameters.
GDK_DELIMITER	Set from the "delimiter" optional parm, and used in URL_PARM request parameters.
GDK_MARKER	Set from the "marker" optional parm, or set internally to the last object in a list when GDK_IST is set, and used in URL_PARM request parameters.
GDK_IST	Indicates that a truncated list of objects was found.
GDK_VERSION_ID	Set from the "prefix" optional parm, and used in URL_PARM request parameters.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	RACF User ID used to retrieve Cloud security credentials for the GET request.
Content-Length	Integer	Integer value to place in the HTTP header of the request indicating the amount sent.
Content-LengthE	8-byte-Integer	Address of an 8-byte Integer value to place in the HTTP header of the request indicating amount set.
Range-Start	Integer	Specifies the ending byte number for the request. If not specified (and Range-Start is specified), then read to end. If Range-End is specified (or is 0), but Range-Start is not specified, then the last number of bytes, inclusive, are returned from the object. If invalid, may be ignored by Cloud Server.
Range-End	Integer	Specifies the ending byte number for the request. If not specified (and Range-Start is specified), then read to end. If Range-End is specified (or is 0), but Range-Start is not specified, then the last number of bytes, inclusive, are returned from the object. If invalid, may be ignored by Cloud Server.
Get-HWTH-Code	Address	Specifies the address of a 4-byte Integer field where the resulting z/OS Client Web Enablement Toolkit return code should be stored. (See HWTRQST documentation in the z/OS Client Web Enablement Toolkit HTTP Enabler section of the z/OS MVS Programming Callable Services for High Level Languages manual for more details.)
Get-HWTH-Diag	Address	Specifies the address of an HWTH_DIAGAREA_TYPE area where the resulting z/OS Client Web Enablement Toolkit diagnostic information should be stored. (See HWTRQST documentation for more details.)
AutoConvert	Character	"true" means that ASCII to EBCDIC translation should be performed on the data being retrieved from or sent to the cloud "false" means that no translation should be performed. This optional parameter will override any default, session value, or value in the config.json file. If not passed, the default is no translation of data, or whatever the session value is.
Set-Header-Buffer	Character	String of custom headers that should be included on the HTTP GET request. Each header is newline (\n) separated.

Optional Parameters		
NAME	Type	Description
Get-Header-Buffer	Address	Pointer to storage area to place the headers from the HTTP Response.
Get-Header-Buffer-Size	Address	Pointer to a 4-byte signed number field that is the size of the provided buffer to contain the headers. Must be provided if Get-Header-Buffer is specified.
Get-Body-Buffer	Address	Pointer to 4-byte area to place the HTTP status code from the HTTP GET request.
Get-Body-Buffer-Size	Address	Pointer to a 4-byte signed number field that is the size of the provided body buffer. Must be provided if Get-Body-Buffer is specified.
Get-Response-Code	Address	Pointer to 4-byte area to place the HTTP status code from the HTTP Get request.
Use-Config-File	Character	"false" means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.
log-level	Character	<p>This setting sets the default logging level for all subsequent API calls. The default logging level may be overridden for specific API calls via the optionalParms on that call. The default logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stderr. The levels in order of low to high severity are listed:</p> <ul style="list-style-type: none"> • DEBUG means all logging messages are written. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.

Optional Parameters		
NAME	Type	Description
web-toolkit-logging	Character	<p>"false" means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should not be written.</p> <p>"true" means that additional logging messages from the z/OS Client Web Enablement Toolkit processing should be written.</p> <p>This value will override any default or value in the config.json file. Logging messages are written to stdout.</p>
CDA_Session	Address	Specifies the address of a 12-byte field that was filled in as part of a gdkinit() CPI call. This indicates a session is current and the config file, keyfile, and provider file will not be read again during this API call.
prefix	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_PREFIX substitution text in a URL_PARM request parameter.
delimiter	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_DELIMITER substitution text in a URL_PARM request parameter.
versionID	Address	Specifies the address of a null-terminated string value that should be stored in the GDK_VERSION_ID substitution text in a URL_PARM request parameter.
logOutput	Address	Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.

Optional Parameters		
NAME	Type	Description
providerFile	Address	Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.

Usage notes

When the API is invoked without the session handle, if UserID wasn't provided in the OptionalParms, the current user's RACF ID is used to retrieve the applicable CloudProvider definitions, as well as retrieve the appropriate Security Credentials for the User/Cloud Provider/Resource combination from the gdkkeyf.json document.

When retrieving the CloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment in order to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory does not exist or the provider file is not found there, then the CDA System Default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

When retrieving the Security Credentials to use to communicate with the cloudProvider specified on the API call, the gdkkeyf.json file must first be located. The user's RACF ID, or UserID from the optionalParms, is used to find the home directory. That home directory is examined for a gdk/gdkkeyf.json document that can be accessed for READ. If not found, then the CDA API call fails.

Once the keyfile has been found, it is parsed, looking for an entry associated with the user's RACF ID, or UserID from the optionalParms. The most specific bucket/container name (Resource) is searched for first. A bucket name issue the first characters surrounded by a forward slash character. I.e /bucket_name/ if no entry is found, then the generic / Resource name is searched for.

With the most appropriate entry found, the Security Credentials will be decrypted using ICSF. Thai decryption key is expected to be found using a keylabel of GDK. <cloudProvider>.<UserID/RACFID>.<sequence_number>. If RACF protection of the ICSF key labels is being used, the invoker of the API must have authority to the key label.

If a valid session handle is passed via the optional parameters, then the config.json file, keyfile, and provider file are not re-read.

Ensure that the GDK variables that are specified in the provider file for the requested operation are completely satisfied. GDK variables are in the format <var_name> within the provider file. When specifying in the genericParmsStruct, the key should be *var_name*.

The requested operation may direct the details of the request via the requestParameter object. Additionally, the response, including the headers and body, may be directed via the responseResults object.

The requestParameter object may hold an entry with "mechanism": "HEADER" to add the header described by the "descriptor" key. A mechanism of "MESSAGE_BODY" may be used to request that the body to be sent come from the <GDK_DATA> descriptor, as well. A mechanism of "URL_PARM" may be used to request that the url be modified with a query as defined by the "descriptor" key.

The responseResults object may hold an entry with a mechanism of "HEADER" to indicate that a specific response header name be processed according to the "descriptor" key.

Restrictions

None.

Authorization

User.

Related services

- [“GDKGET — Retrieve a cloud object” on page 716](#)
- [“GDKWRITE — Write an object to cloud storage” on page 728](#)
- [“GDKDEL — Delete an object from cloud storage” on page 742](#)
- [“GDKLIST — List cloud objects” on page 748](#)

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 165. Return and reason codes for the GDKKEYGR service		
Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
100	GDK_UNABLE_TO_READ_KEYFILE	The gdkkeyf.json document was not found in the CDA System directory.
102	GDK_UNABLE_TO_PARSE_FILE	When parsing the gdkkeyf.json locum.net, it is not a valid JSON document
104	GDK_CLOUD_PROVIDER_NOT_FOUND	When reading the gdkkeyf.json document, for the requested user, the specified cloudProvider was not found.
105	GDK_NO_RESOURCE_FOUND_IN_KEYFILE	For the specified UserID, and cloudProvider, no credentials entry for / was found.
110	GDK_PROVIDER_OPEN_FAILURE	The JSON document for the requested cloudProvider does not exist, or was not found, but cannot be opened for READ.
111	GDK_PROVIDER_SPECIFICATION_INVALID	When parsing the JSON document, it was found to be invalid.
112	GDK_FEATURE_UNSUPPORTED	An unknown content Type was found in the parameterSet.
113	GDK_BUFFER_TOO_SMALL	When attempting to fill the passed buffer, the amount of data is too big to fit in the buffer.
114	GDK_AUTH_INIT_FAILURE	When using the AWS4 authentication model, the appropriate entry in the gdkkeyf.json document for the "UserID", "cloud provider", and "name" (resource) is missing either the "key", or the "secretkey" key-value pairs.
116	GDK_AUTH_APPLY_FAILURE	An error occurred while applying the authorization parameters.

Table 165. Return and reason codes for the GDKKEYGR service (continued)

Return code	Constant Name	Explanation
118	GDK_USER_INFO_NOT_FOUND	When reading the gdkkeyf.json document, the requested user entry was not found.
123	GDK_CONN_NOT_HTTPS	HTTPS was not specified for httpMethod in the provider file.
124	GDK_ICSF_ERROR	ICSF returned an error during processing. Refer to the accompanying ICSF return code and reason code.
141	GDK_PARAMETER_ERROR	One of the parameters on the API call was incorrect. Examine the ERROR log message for details on the parameter in error.
142	GDK_EMPTY_PROVIDER_FILE	The specified <cloudProvider>.json file is empty. <cloudProvider> is the provider name specified on the API parameter list.
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.
800	GDK_CONNECTION_FAILED	Unable to connect to the host set in the cloud provider JSON definition.
801	GDK_TOOLKIT_FAILED	Error occurred in the z/OS Client Web Enablement Toolkit HWTHRQST call.
900	GDL_OBJECT_NOT_FOUND	HTTP status 404 was returned indicating the object was not found.
901	GDK_ACCESS_DENIED	HTTP status 403 was returned indicating the user is not authorized to access the resource in the desired manner.
902	GDK_OPERATION_NOT_SUPPORTED	When parsing the cloud provider JSON document, the GETOBJECT description was not found in the "supportedOperations" array.
903	GDK_RESPONSE_FORMAT_MISMATCH	The request was successful, but the data returned was in a format that did not match what is expected according to the contentType specified in the responseResults for the action in the provider file.
904	GDK_REQUEST_FAILED	A bad HTTP status (4xx or 5xx) was returned.

GDKVALD – Validate Provider File

Description

The GDKVALD API will allow callers to request that CDA examine a named provider json file. The caller may pass operation names as well as specific key names on input. The API will verify that the requested operation name exists, and may return the value associated with the requested keys. The intended use of this API is to allow callers to validate that the requested provider file meets the requirements of their expected processing. Additionally, it allows retrieval of values from the provider file.

The GDKVALD API processing can additionally be modified through the optionalParmStruct.

Purpose (or Function): Perform validation of operations in a provider file, and additionally return values for requested key names.

Requirements: When using the Branch Entry for gdkvald(), a Language Environment must be established before this API is invoked. When called from a Language Environment C program, the environment is already created and available. Otherwise a PreInitialization environment can be used to create a Language Environment (CEEPIPI). For more information, see [Using preinitialization services in z/OS Language Environment Programming Guide](#)

When linking with SYS1.CSSLIB(GDKCSS), then a Language Environment should already be established. When linking with SYS1.CSSLIB(GDKCSSNL), then a Language Environment does not need to be established as it will be created on entry, and terminated at exit.

When using Branch Entry, and a Language Environment is not established, the GDKGENN API may be used. This will create a Language Environment upon entry and terminate the Language Environment upon exit.

Syntax

```
gdkvald (retCodeAddr,
        cloudProvider,
        valParmStructPtr,
        optionalParmStructPtr);
```

When using Branch Entry, and a Language Environment is not established, the GDKVALDN API may be used. This will create a Language Environment upon entry and terminate the Language Environment upon exit.

```
gdkvaldn (retCodeAddr,
          cloudProvider,
          valParmStructPtr,
          optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

cloudProvider

Specifies the address of a pointer to a name of the cloud provider that this request should contact. The name must be null-terminated. When retrieving the CloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory does not exist or the <cloudProvider>.json file is not found there, then the CDA System Default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

The application may request a list of all supported providers by using the **GETPROVIDERS** API.

valParmStructPtr

Specifies the address of the validateParmsStruct block which contains key names to be validated as well as pointers to buffers to hold the returned values if requested.

validateParmsStruct Control Block Mapping

```
/* ----- */
/* validateParmsStruct Type definitions (GDKVALDservice) */
/* - maps the data pointed to by the valParmsStructPtr */
/* parameter */
/* ----- */
typedef void * GDK_VAL_VALUE_TYPE; /* Generic pointer to value */

typedef struct {
    GDK_OPTION_KEY_TYPE GDK_valOp; /* Operation name */
    GDK_OPTION_KEY_TYPE GDK_valKey; /* Key name pointer */
    GDK_VAL_VALUE_TYPE GDK_valValue; /* Value buffer pointer */
    unsigned short GDK_valLen; /* Buffer size/Length used */
    unsigned char GDK_valRqst; /* Request options */
    unsigned char _GDK_rsrv2; /* Reserved */
    unsigned char GDK_valExists; /* 1 = Key exists */
    unsigned char GDK_valType; /* Type of value */
}
```

```

    unsigned short      GDK_valErr; /* Entry in error with RC      */
} GDK_VAL_TYPE;          /* Single Validation Entry    */

typedef struct {
    uint32_t GDK_NumOfValTypes; /* Count of Validation Ents  */
    GDK_VAL_TYPE GDK_ValArray[]; /* Array of GDK_VAL_TYPES    */
} GDK_VAL_PARMS_TYPE; /* Header of structure      */

#define GDK_VAL_INVALID 0 /* Invalid type              */
#define GDK_VAL_STRING 3 /* String type               */
#define GDK_VAL_INT 4 /* Integer type              */
#define GDK_VAL_BOOL 5 /* Boolean type: 0 is false   */
#define GDK_VAL_ARRAY 6 /* Array of Entries          */
#define GDK_VALERR_SIMPLE 8 /* Key is not simple type    */

#define GDK_MAX_HOST_NAME_LEN 508 /* Longest host name 0-term */
#define GDK_VR_ARR_OK 5 /* Return Array of Values OK */

typedef struct {
    unsigned short GDK_vType; /* Type of Entry              */
    unsigned short GDK_vLen; /* Length of Entry            */
    char GDK_vStr[GDK_MAX_HOST_NAME_LEN]; /* Array for string          */
} GDK_VAL_ARR_ENT; /* Array entry description    */

#define GDK_VA_PREF 0x0097 /* Preferred hostname entry  */
#define GDK_VA_BACK 0x0082 /* Backup hostname entry     */

```

Within a single GDK_VAL_TYPE entry, the fields have the following meanings and uses:

- GDK_valOp field points to a null terminated string that is the name operation to be searched for in the supportedOperations array. Set to NULL if the first instance of GDK_valKey should be returned. Set to the name of the operation that should be searched inside.
- GDK_valKey field points to a null terminated string that is the key name that should be validated. If only wanting to verify the existence of an operation, this should be set to NULL.
- GDK_valValue should point to a buffer large enough to hold the returned value. When GDK_valKey is NULL, the buffer this points to will not be filled.
- GDK_valLen should be set to maximum size of the buffer. After the call, this will be set to the actual used size if no error, or zero if an error occurred.
- GDK_valRqst is used to indicate the types of values acceptable when multiple types may be returned. Default is 0, indicating only the default type should be returned. Set to GDK_VR_ARR_OK to indicate that an array of GDK_VAL_ARR_ENT entries is acceptable.
- GDK_valExists – On output is set to 1 to indicate the GDK_valKey was found as expected. It is set to 0 to indicate the GDK_valKey was not found as expected. If GDK_valKey was NULL, it is set to 1 to indicate that the operation object was found in the supportedOperations array. It is set to 0 if the operation name was not found in the supportedOperations array.
- GDK_valErr – Output field regarding result of processing
 - 0 if no error occurred.
 - 8 if the key was found, but was not a simple type (String, Number, Boolean, Null).
 - 113 if the buffer is too small. GDK_valLen will be set to the needed size.
 - 902 if operation is not found
- GDK_valType – On Output is set to a value to indicate the content of GDK_valValue.
 - 0 – Value is invalid. (It doesn't point to anything.)
 - 3 – Value is a null-terminated string.
 - 4 – Value is a number. (GDK_valLen is 4 for a 4-byte integer)
 - 5 – Value is a 4-byte Boolean value. 0 means false. 1 means true.
 - 6 – Value is an array of GDK_VAL_ARR_ENT entries.

When GDK_valType indicates an array of entries is placed in the buffer that GDK_valVal points to, each entry in the array is laid out in the fixed format described by the GDK_VAL_ARR_ENT mapping:

- GDK_vType – A 2-byte field indicating the type of entry. Values are:

- 0x0097 – Preferred host name from host array
- 0x0082 – Backup host name from host array
- GDK_vLen – A 2-byte field containing the length of string following.
- GDK_vStr – A 508-byte character area containing the null-terminated host name which may be of the format: https://<host_name><:port><,sslKey_ring>.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
UserID	8-byte char	RACF User ID used to retrieve Cloud security credentials for the GET request.
Use-Config-File	Character	"false" means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.
log-level	Character	<p>This setting sets the default logging level for all subsequent API calls. The default logging level may be overridden for specific API calls via the optional Parms on that call. The default logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stderr. The levels in order of low to high severity are listed:</p> <ul style="list-style-type: none"> • DEBUG means all logging messages are written. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.
CDA_Session	Address	Specifies the address of a 12-byte field that was filled in as part of a gdkinit() CPI call. This indicates a session is current and the config file, keyfile, and provider file will not be read again during this API call.

Optional Parameters		
NAME	Type	Description
logOutput	Address	Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.
providerFile	Address	Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.

Usage notes

When the API is invoked without the session handle, if UserID wasn't provided in the OptionalParms, the current user's RACF ID is used to retrieve the applicable CloudProvider definitions, as well as retrieve the appropriate Security Credentials for the User/Cloud Provider/Resource combination from the gdkkeyf.json document.

When retrieving the CloudProvider definition corresponding to the cloudProvider specified on the API call, the user's RACF ID, or UserID from the optionalParms, is used to examine the associated OMVS segment in order to retrieve the home directory. That home directory is examined for a gdk/providers/ sub-directory. If it exists and the <cloudProvider>.json file is found, it will be used as the template to communicate with the cloud provider. If the gdk/providers/ directory does not exist or the provider file is not found there, then the CDA System Default directory of /usr/lpp/dfsms/gdk/providers/ will be used.

If a session handle is passed via the optional parameters, then the config.json file, keyfile, and provider file are not re-read.

If the "host" key is requested, and the host key in the provider has a value that is a JSON object containing a "preferred" list and optionally a "backup" list and GDK_valRqst is not GDK_VR_ARR_OK, then the return value will be a randomly chosen host value from the preferred array. If the size of the return buffer is

not big enough to contain all of the values, as many complete values that can fit will be returned and the GDK_valErr will be set to RC 113 indicate there was more data.

Restrictions

None.

Authorization

User.

Related services

None.

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDRV.H.

Table 166. Return and reason codes for the GDKKEYGR service		
Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
104	GDK_CLOUD_PROVIDER_NOT_FOUND	When reading the gdkkeyf.json document, for the requested user, the specified cloudProvider was not found.
110	GDK_PROVIDER_OPEN_FAILURE	The JSON document for the requested cloudProvider does not exist, or was not found, but cannot be opened for READ.
111	GDK_PROVIDER_SPECIFICATION_INVALID	When parsing the JSON document, it was found to be invalid.
113	GDK_BUFFER_TOO_SMALL	When attempting to fill the passed buffer, the amount of data is too big to fit in the buffer.
141	GDK_PARAMETER_ERROR	One of the parameters on the API call was incorrect. Examine the ERROR log message for details on the parameter in error.
142	GDK_EMPTY_PROVIDER_FILE	The specified <cloudProvider>.json file is empty. <cloudProvider> is the provider name specified on the API parameter list.
902	GDK_OPERATION_NOT_SUPPORTED	When parsing the cloud provider JSON document, the GETOBJECT description was not found in the "supportedOperations" array.

GDKQUERY – Query available CDA functions

Description

Purpose (or Function): Retrieve information about the supported CDA functions on the current system.

Requirements: When using the Branch Entry, a Language Environment must be established before the gdkquery() API is invoked. When called from a Language Environment C program, the environment is already created and available. Otherwise, a PreInitialization environment can be used to create a Language Environment (CEEPIPI). The *z/OS Language Environment Programming Guide* has more details in the [Using preinitialization services](#) section. When using the Branch Entry interface, if no Language Environment exists, the alternate gdkqryn() API may be invoked. This will create and tear down a Language Environment for the duration of the Call.

When linking with SYS1.CSSLIB(GDKCSS), then a Language Environment should already be established.

When linking with the SYS1.CSSLIB(GDKCSSNL), then a Language Environment will be established then torn down once processing is completed.

Syntax

```
gdkquery (retCodeAddr,
          queryBuffer,
          queryBufferLen,
          optionalParmStructPtr);
```

When using Branch Entry, and a Language Environment is not established, the GDKQRYN API may be used. This will create a Language Environment upon entry and terminate the Language Environment upon exit.

```
gdkqryn (retCodeAddr,
          queryBuffer,
          queryBufferLen,
          optionalParmStructPtr);
```

Parameters

The parameters are explained as follows:

retCodeAddr

Specifies the address of a 4-byte field that the API will place the return code into.

queryBuffer

Specifies the address of a pointer to a buffer that is large enough to hold one of the versions of the GDK_FUNCTION_MAP_Vx control blocks. The GDK_FUNCTION_MAP_V1 block is 1024 bytes in size.

GDK_FUNCTION_MAP_V1 Control Block

```
/******
/* GDK_FUNCTION_MAP_V1 control block mapping
/******
typedef struct {
    char gdkf1_eyec[8];           /* Eye catcher for control block */
    unsigned short gdkfg_ver;     /* Version of the Function Map */
    char _rsrvd1 char[2];        /* Reserved space */
    struct{
        int gdkf1_gdkdel :1,      /* Flags */
        gdkf1_gdkget :1,         /* GDKDEL API */
        gdkf1_gdkwrite :1,       /* GDKGET API */
        gdkf1_gdklist :1,        /* GDKWRITE API */
        gdkf1_gdkkeysr :1,       /* GDKLIST API */
        gdkf1_gdkkeyad :1,       /* GDKKEYSR API */
        gdkf1_gdkkeygr :1,       /* GDKKEYAD API */
        gdkf1_gdkkeydl :1,       /* GDKKEYGR API */
        gdkf1_gdkinit :1,        /* GDKKEYDL API */
        gdkf1_gdkterm :1,        /* GDKINIT API */
        gdkf1_gdkgen :1,         /* GDKTERM API */
        gdkf1_gdkvald :1,        /* GDKGEN API */
        gdkf1_gdkquery :1,       /* GDKVALD API */
        :18;                     /* GDKQUERY API */
    } api_flags1;                /* Reserved bits */
    struct{
        int gdkf1_fmata :1,       /* Supported API flags */
        gdkf1_exit_multipart :1, /* Flags */
        gdkf1_list_prefix :1,    /* Object metadata */
        gdkf1_lilst_delim :1,    /* Multipart upload for EXIT */
        :28;                     /* LIST PREFIX support */
    } func_flags1;               /* LIST Delimiter support */
    /* Reserved bits */
    /* Supported function flags */
}
```

```

char _rsrvdEnd[1004];          /* Reserved space          */
} GDK_FUNCTION_MAP_V1;        /* Version 1 function map  */

#define GDKF_V1 1;             /* Version 1 function map  */

```

queryBufferLen

Specifies the address of a 4-byte unsigned number that is the size of the queryBuffer storage. It must be at least 1024.

optionalParmStructPtr

Specifies an optional method for a user of this API to provide customized processing not provided by default by the CDA API. The API will specify a pointer to a structure as mapped by the data structure **GDK_OPTIONAL_PARMS_TYPE**. This data structure will contain one or more customized overrides or additions. In general, all string values, including keys and values, must be null-terminated when being passed to the APIs.

Optional Parameters		
NAME	Type	Description
Use-Config-File	Character	"false" means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.
log-level	Character	<p>This setting sets the default logging level for all subsequent API calls. The default logging level may be overridden for specific API calls via the optional Parms on that call. The default logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stderr. The levels in order of low to high severity are listed:</p> <ul style="list-style-type: none"> • DEBUG means all logging messages are written. • INFO means only INFO and higher severity logging messages are written. • NOTICE means only NOTICE and higher severity logging messages are written. • WARNING means only WARNING and higher severity logging messages are written. • ERROR means only ERROR logging messages are written. • NONE means no messages are written regardless of severity.

Optional Parameters		
NAME	Type	Description
logOutput	Address	Specifies the address of a null terminated string that is the name of the output for CDA log messages. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it defaults to stderr, which in a JCL environment is directed to a DD named SYSOUT.
providerFile	Address	Specifies the address of a null terminated string containing a valid JSON document with all of the provider file contents describing how to communicate with the target cloud object server. When this optional parameter is specified, CDA will copy the contents and use that as the provider specification and not attempt to read the provider file from the user's provider directory or the system default provider directory.
keyFile	Address	Specifies the address of a null terminated string containing a valid JSON document consisting of the contents of a CDA keyfile for the user. When this optional parameter is specified, CDA will copy the contents and use that as the keyfile contents to be used when authenticating with the target cloud object server.

Usage notes

When examining the GDK_FUNCTION_MAP, the version field should be examined to understand which version of the map is being returned.

Restrictions

None.

Authorization

User.

Related services

- [“GDKMSGTR — Translate a CDA return code into text” on page 762](#)

Return and reason codes

The various return code constants are documented in the gdkic header file, found in SYS1.SIEAHDV.H.

Table 167. Return and reason codes for the GDKKEYGR service

Return code	Constant Name	Explanation
0	GDK_OK	Processing was successful.
113	GDK_BUFFER_TOO_SMALL	When attempting to fill the passed buffer, the amount of data is too big to fit in the buffer.
799	GDK_UNEXPECTED_ERROR	An unexpected error occurred. Contact IBM Level 2, and provide the logging output.

gdkkeysrJ/gdkkeysr64J – Retrieve cloud credentials JSON API

The gdkkeysrJ C API is used to retrieve the stored cloud credentials for a user associated with a cloud provider and resource with the input and output being JSON.

Description

The gdkkeysrJ C API is used to retrieve the cloud credentials for the specified cloud provider and resource. When called, the key file for the user will be read and the encrypted cloud credentials matching the request will be decrypted and returned. An AMODE64 version of the C API is available on z/OS 3.1 and called with gdkkeysr64J. The gdkkeysrJ C API processing can additionally be modified through optional parameters.

Syntax

```
char* gdkkeysrJ(char *__JSON_in)
char* gdkkeysr64J(char *__JSON_in)
```

Input Parameter

The input parameter is a null-terminated character string JSON document containing the following key/value pairs:

provider

Specifies the name of the cloud provider the credentials are associated with. Required.

resource

Specifies the bucket name the credentials are associated with. Optional. If not specified, the credentials associated with the default bucket / are returned.

maxKeyLen

Specifies a number that is the maximum length of each credentials returned. Optional. Defaults to 8192 bytes.

optionalParms

Specifies a JSON object containing key/value pairs used to modify processing of the API.

UserID - RACF UserID used to retrieve cloud credentials.

Use-Config-File - false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.

log-level - The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stderr unless overridden by the logOutput optional parameter. The levels in order of low to high severity are listed:

- **DEBUG** means all logging messages are written.
- **INFO** means only INFO and higher severity logging messages are written.
- **NOTICE** means only NOTICE and higher severity logging messages are written.

- **WARNING** means only WARNING and higher severity logging messages are written.
- **ERROR** means only ERROR logging messages are written.
- **NONE** means no messages are written regardless of severity.

logOutput Specifies the name of a location where the logging output messages should be written. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, / UNIX/file/name.text, or //"IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it default to stderr, which in a JCL environment is directed to a DD named SYSOUT.

```
Example input JSON to retrieve the default credentials for the IBMCOS cloud provider
{
  "provider": "IBMCOS",
  "optionalParms": {
    "Use-Config-file": "false",
    "log-level": "DEBUG"
  }
}
```

Return JSON

The return string is a JSON containing the following fields as appropriate. It is the responsibility of the caller of the API to free the returned memory.

rc

CDA return code from processing.

provider

The name of the cloud provider the credentials are associated with.

resource

The bucket name the credentials are associated with.

accessKey

Access Key ID value if present in the key file.

secretAccessKey

Secret Access Key value if present in the key file.

tenant

Tenant value if present in the key file.

userid

Userid value if present in the key file.

password

Password value if present in the key file.

accessURL

Access URL value if present in the key file.

maxKeyLen

Specifies the maximum key buffer length to use when retrieving the credentials. If not specified, the default size is 8192 bytes.

messages

JSON array of output messages from processing.

```
Example output JSON
{
  "rc": 0,
  "provider": "IBMCOS",
  "resource": "/bucket1",
  "accessKey": "myExampleAccessKey1D1",
  "secretAccessKey": "myExampleSecretAccessKey1",
  "accessURL": "https://loc.site.org/auth/v1.0",
  "messages": [
    "Success"
  ]
}
```



```
}  
}
```

gdkkeyadJ/gdkkeyad64J – Store cloud credentials JSON API

The gdkkeyadJ C API is used to store cloud credentials for a user associated with a cloud provider and resource with the input and output being JSON.

Description

The gdkkeyadJ C API is used to store the cloud credentials for the specified cloud provider and resource. When called, the passed cloud credentials are encrypted by ICSF, using a newly generated AES256 data key. The new data key is stored with ICSF under a keylabel of:

```
GDK.<userid>.<provider>.<identifier>
```

where:

- **userid** is the current RACF user's ID
- **provider** is the specified cloud provider in the user's ~/gdk/providers/ directory or system default directory off /usr/lpp/dfsms/gdk/providers/.
- **identifier** is a CDA used identifier used internally

For more information, see Cloud Data Access cloud credential storage in *z/OS MVS Programming: Callable Services for High-Level Languages*. Storing the encrypted credentials. The keyfile in the current user's (or UserID optional parm_ ~/gdk/ directory is read, and a new entry is created, or an existing entry is overwritten for the specified cloud provider and resource. An AMODE64 version of the C API is available on z/OS 3.1 and is called with gdkkeyad64J C API processing can additionally be modified through optional parameters.

Syntax

```
char* gdkkeyadJ(char *__JSON_in)  
char* gdkkeyad64J(char *__JSON_in)
```

Input Parameter

The input parameter is a null-terminated character string JSON document containing the following key/value pairs:

provider

Specifies the name of the cloud provider the credentials are associated with. Required.

resource

Specifies the bucket name the credentials are associated with. Optional. If not specified, the credentials associated with the default bucket / are returned.

accessKey

Access Key ID value for the credentials. Optional.

secretAccessKey

Secret Access Key value for the credentials. Optional.

tenant

Tenant value for the credentials. Optional.

userid

Userid value for the credentials. Optional.

password

Password value for the credentials. Optional.

accessURL

Access URL value for the credentials. Optional.

Note: Authentication methods and associated parameters

The authentication object in the cloud provider json file determines which parameter values are used in an operation.

AWS4

- keyValue
- secretKeyValue

AZURE

- keyValue
- secretKeyValue

BASIC

- keyValue
- secretKeyValue

KEYSTONE

- userIDValue
- passwordValue
- tenant
- accessUrl

OAUTH_2

- tenant_holds (Access_Token, Client_Secret, Credentials_file)
 - When Access_Token, secretKeyValue is used to hold the Access token.
 - When Client_Secret, keyValue, and SecretKeyValue is used.
 - When Credentials_file, accessUrl holds absolute path to credentials file.

TEMPAUTH

- userIDValue
- passwordValue
- Optional: accessUrl (Used, if it exists)

optionalParms

Specifies a JSON object containing key/value pairs used to modify processing of the API.

UserID - RACF UserID used to retrieve cloud credentials.

Use-Config-File - false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.

log-level - The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stderr unless overridden by the logOutput optional parameter. The levels in order of low to high severity are listed:

- **DEBUG** means all logging messages are written.
- **INFO** means only INFO and higher severity logging messages are written.
- **NOTICE** means only NOTICE and higher severity logging messages are written.
- **WARNING** means only WARNING and higher severity logging messages are written.
- **ERROR** means only ERROR logging messages are written.

- **NONE** means no messages are written regardless of severity.

logOutput Specifies the name of a location where the logging output messages should be written. It must conform to the naming convention documented for the `open()` function. e.g. DD:OUTDD, /UNIX/file/name.text, or `/"IBMUSER.CDATALOG.OUTPUT'`. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it default to `stderr`, which in a JCL environment is directed to a DD named `SYSOUT`.

```
Example input JSON to store the default credentials for an S3 cloud provider named IBMCOS
{
  "provider": "IBMCOS",
  "accessKey": "myAccessKey",
  "secretAccessKey": "mySecretKeyID",
  "optionalParms": {
    "Use-Config-file": "false",
    "log-level": "DEBUG"
  }
}
```

Return JSON

The return string is a JSON containing the following fields as appropriate. It is the responsibility of the caller of the API to free the returned memory.

rc

CDA return code from processing.

provider

The name of the cloud provider the credentials are associated with.

resource

The bucket name the credentials are associated with.

messages

JSON array of output messages from processing.

```
Example output JSON
{
  "rc": 0,
  "provider": "IBMCOS",
  "resource": "/",
}
```

gdkkeydlJ/gdkkeydl64J — Delete cloud credentials JSON API

The `gdkkeydlJ` C API is used to delete the stored cloud credentials for a user associated with a cloud provider and resource with the input and output being JSON.

Description

The `gdkkeydlJ` C API is used to delete the cloud credentials for the specified cloud provider and resource. When called, the key file for the user will be read and the encrypted cloud credentials matching the request will be decrypted and returned. An AMODE64 version of the C API is available on z/OS 3.1 and is called with `gdkkeydl64J`. The `gdkkeydlJ` C API processing can additionally be modified through optional parameters.

Syntax

```
char* gdkkeydlJ(char *__JSON_in)
char* gdkkeydl64J(char *__JSON_in)
```

Input Parameter

The input parameter is a null-terminated character string JSON document containing the following key/value pairs:

provider

Specifies the name of the cloud provider the credentials are associated with. Required.

resource

Specifies the bucket name the credentials are associated with. Optional. If not specified, the credentials associated with the default bucket / are returned.

optionalParms

Specifies a JSON object containing key/value pairs used to modify processing of the API.

UserID - RACF UserID used to retrieve cloud credentials.

Use-Config-File - false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.

log-level - The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stderr unless overridden by the logOutput optional parameter. The levels in order of low to high severity are listed:

- **DEBUG** means all logging messages are written.
- **INFO** means only INFO and higher severity logging messages are written.
- **NOTICE** means only NOTICE and higher severity logging messages are written.
- **WARNING** means only WARNING and higher severity logging messages are written.
- **ERROR** means only ERROR logging messages are written.
- **NONE** means no messages are written regardless of severity.

logOutput Specifies the name of a location where the logging output messages should be written. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //"IBMUSER.CDALOG.OUTPUT". When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it default to stderr, which in a JCL environment is directed to a DD named SYSOUT.

```
Example input JSON to delete the credentials for the IBMCOS provider
{
  "provider": "IBMCOS",
  "optionalParms": {
    "Use-Config-file": "false",
    "log-level": "DEBUG"
  }
}
```

Return JSON

The return string is a JSON containing the following fields as appropriate. It is the responsibility of the caller of the API to free the returned memory.

rc

CDA return code from processing.

provider

The name of the cloud provider the credentials are associated with.

resource

The bucket name the credentials are associated with.

accessKey

Access Key ID value if present in the key file.

secretAccessKey

Secret Access Key value if present in the key file.

tenant

Tenant value if present in the key file.

userid

Userid value if present in the key file.

password

Password value if present in the key file.

accessURL

Access URL value if present in the key file.

maxKeyLen

Specifies the maximum key buffer length to use when retrieving the credentials. If not specified, the default size is 8192 bytes.

messages

JSON array of output messages from processing.

Example output JSON

```
{
  "rc": 0,
  "provider": "IBMCOS",
  "resource": "/bucket1",
  "accessKey": "myExampleAccessKey1D1",
  "secretAccessKey": "myExampleSecretAccessKey1",
  "accessURL": "https://loc.site.org/auth/v1.0",
  "messages": [
    "Success"
  ]
}
```

gdkkeygrJ/gdkkeygr64J — List cloud credentials resources JSON API

The gdkkeygrJ C API is used to retrieve a list of cloud credentials identified by their bucket names or resources for a user associated with a cloud provider with the input and output being JSON.

Description

The gdkkeygrJ C API is used to retrieve a list of the cloud credentials for the specified cloud provider. When called, the key file for the user will be read and a list containing the bucket name or resources will be returned. This API only identifies which credentials exist. No credentials are returned in. An AMODE64 version of the C API is available on z/OS 3.1 and is called with gdkkeygr64J. The gdkkeygrJ C API processing can additionally be modified through optional parameters.

Syntax

```
char* gdkkeygrJ(char *__JSON_in)
char* gdkkeygr64J(char *__JSON_in)
```

Input Parameter

The input parameter is a null-terminated character string JSON document containing the following key/value pairs:

provider

Specifies the name of the cloud provider the credentials are associated with. Required.

optionalParms

Specifies a JSON object containing key/value pairs used to modify processing of the API.

UserID - RACF UserID used to retrieve cloud credentials.

Use-Config-File - false means that the config.json file should not be read for default configuration values. Any other values mean that the config.json should be used.

log-level - The logging level can be set as desired. This value will override any default, or value in the config.json file. Logging messages are written to stderr unless overridden by the logOutput optional parameter. The levels in order of low to high severity are listed:

- **DEBUG** means all logging messages are written.
- **INFO** means only INFO and higher severity logging messages are written.
- **NOTICE** means only NOTICE and higher severity logging messages are written.
- **WARNING** means only WARNING and higher severity logging messages are written.
- **ERROR** means only ERROR logging messages are written.
- **NONE** means no messages are written regardless of severity.

logOutput Specifies the name of a location where the logging output messages should be written. It must conform to the naming convention documented for the fopen() function. e.g. DD:OUTDD, /UNIX/file/name.text, or //'IBMUSER.CDALOG.OUTPUT'. When the log-level is not NONE, CDA logging messages will be written to the requested output name, which can be a DD, z/OS UNIX file, or data set. The userid invoking the API must have enough permission to append data to the output target. If not specified, it default to stderr, which in a JCL environment is directed to a DD named SYSOUT.

```
Example input JSON to retrieve the default cloud credentials for the IBMCOS cloud provider
{
  "provider": "IBMCOS",
  "optionalParms": {
    "Use-Config-file": "false",
    "log-level": "DEBUG"
  }
}
```

Return JSON

The return string is a JSON containing the following fields as appropriate. It is the responsibility of the caller of the API to free the returned memory.

rc

CDA return code from processing.

provider

The name of the cloud provider the credentials are associated with.

resource

The bucket name the credentials are associated with.

timestamp

The date and time the credentials were saved in the key file.

messages

JSON array of output messages from processing.

```
Example output JSON
{
  "rc": 0,
  "provider": "IBMCOS",
  "results": [
    { "resource": "/bucket", "timestamp": "2024-09-26 14:43:17" },
    { "resource": "/", "timestamp": "2024-09-27 17:07:14" },
    { "resource": "/testbucket", "timestamp": "2024-09-27 17:57:40" }
  ]
}
```

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, the Diag_Commerr field in the DiagArea contains a more descriptive return code from the BCPii communications transport to help pinpoint the cause of the failure.

The following table provides a partial list of the descriptive communication transport error return codes, along with a suggested action to take.

Return code, in hexadecimal (in decimal)	Description / suggested action
0-63 (0-99)	<p>These return codes are documented in Appendix C (API Return Codes) in <i>IBM z SNMP Application Programming Interfaces</i> (SB10-7171-06).</p> <p>For a Diag_Commerr value of X'15' (21 decimal), this may signify a possible busy condition on the targeted SE. An application may choose to retry the request. Persistent failures with this return code should be reported to the IBM Support Center.</p> <p>For a Diag_Commerr value of X'4A' (74 decimal), the z14 or higher CPC rejected communication from BCPii because the BCPii firmware security settings have not been granted BCPii access to the target CPC or LPAR.</p>
64-76 (100-118)	An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.
77 (119)	The BCPii transport rejected the particular request. Activate CTRACE with CTRACE option "ALL" and reissue the request. If the request failed again, turn off CTRACE, collect the SVCDUMP, and contact the IBM Support Center.
78-81 (120-129)	An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.
82 (130)	<p>The support element fails to return the required information needed for BCPii address space to come up.</p> <p>Action:</p> <ol style="list-style-type: none"> 1. If this error occurs during BCPii initialization, restart BCPii manually (S HWISTART) 2. If restarting BCPii manually fails, issue the following to re-drive the SE recovery process to return the required information: <ol style="list-style-type: none"> a. Issue the command VARY CN(*),ACTIVATE from Operating System Messages b. Issue a command (any command) from Operating System Messages c. Manually restart BCPii (S HWISTART) 3. If the above suggested actions still fail, IPL is required to restart BCPii.

Return code, in hexadecimal (in decimal)	Description / suggested action
83-CF (131-207)	An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.
D0 (208)	The support element rejected the particular request. This could occur for any number of reasons including: the SE is busy, the SE is rebooting, etc. Consider retrying the request one or more times. If the problem persists, activate CTRACE with CTRACE option "ALL" and reissue the request. Then turn off CTRACE, collect the SVCDUMP, and contact the IBM Support Center.
D1-D3 (209-211)	An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.
D4 (212)	The support element rejected communication from BCPii, likely for one of the following reasons: <ul style="list-style-type: none"> • If targeting a z13 or lower CPC, the Cross partition authority was not granted on this support element. • If targeting a z14 or higher CPC, the SEND BCPii permission was not granted to the LPAR on this support element.
E0 (224)	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.
Greater than E0 (>224)	An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.

Appendix B. BCPii summary tables

The following summary tables show the objects that can be targeted for the BCPii functions:

- “HWIQUERY and HWISET / HWISET2 attributes” on page 825
- “HWICMD / HWICMD2” on page 821
- “HWIEVENT” on page 823

For complete details of the BCPii APIs, see [Chapter 19, “Base Control Program internal interface \(BCPii\),”](#) on page 245.

BCPii configuration considerations

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.SCEERUN2* and *high-level-qualifier.SCEERUN* 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. BCPii also requires the *high-level-qualifier.SCEERUN2* and *high-level-qualifier.SCEERUN* data sets to be APF authorized. Failure to have these two data sets in the link list or APF authorized 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 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](#) for further information regarding CTRACE settings in BCPii.

BCPii writes SMF record 106 (X'6A') for certain API invocations. An SMFPRMxx parmlib member must be configured and activated in order to capture these records. See [“SMF recording in BCPii”](#) on page 262 or [z/OS MVS System Management Facilities \(SMF\)](#) for more information about how BCPii uses SMF.

HWICMD / HWICMD2

This table shows the BCPii HWICMD and HWICMD2 types and the objects that can be targeted for each command.

Table 168. HWICMD types					
Command type / Constant with hexadecimal and (decimal) values	Description	Starting z/OS release	CPC	Image	User-defined Image Group
HWI_CMD_ACTIVATE 1 (1)	Activate target object	<ul style="list-style-type: none"> • CPC and image: V1R10 • User-defined image group: V1R13 	X	X	X

Table 168. HWICMD types (continued)

Command type / Constant with hexadecimal and (decimal) values	Description	Starting z/OS release	CPC	Image	User-defined Image Group
HWI_CMD_DEACTIVATE 2 (2)	Deactivate target object	<ul style="list-style-type: none"> CPC and image: V1R10 User-defined image group: V1R13 	X	X	X
HWI_CMD_HWMSG 3 (3)	Resend all hardware messages or delete one hardware message	V1R10	X		
HWI_CMD_CBU 4 (4)	Activate or deactivate capacity backup	V1R10	X		
HWI_CMD_OOCOD 5 (5)	Activate or deactivate On/Off Capacity on Demand	V1R10	X		
HWI_CMD_PROFILE 6 (6)	Import or export activation profiles	V1R10	X		
HWI_CMD_RESERVE 7 (7)	Add or delete a reserve for an application	V1R10	X		
HWI_CMD_SYSRESET 8 (8)	Reset target object	<ul style="list-style-type: none"> Image: V1R10 User-defined image group: V1R13 		X	X
HWI_CMD_START 9 (9)	Start all CPs on target object	<ul style="list-style-type: none"> Image: V1R10 User-defined image group: V1R13 		X	X
HWI_CMD_STOP A (10)	Stop all CPs on target object	<ul style="list-style-type: none"> Image: V1R10 User-defined image group: V1R13 		X	X
HWI_CMD_PSWRESTART B (11)	Restart one CP	<ul style="list-style-type: none"> Image: V1R10 User-defined image group: V1R13 		X	X
HWI_CMD_OSCMD C (12)	Issue an operating system command	V1R10		X	
HWI_CMD_LOAD D (13)	IPL operating system or systems	<ul style="list-style-type: none"> Image: V1R10 User-defined image group: V1R13 		X	X
HWI_CMD_TEMPCAP E (14)	Add or remove temporary capacity. For more information see Writing XML for use with the temporary capacity SNMP APIs (www-01.ibm.com/servers/resourcelink/lib03011.nsf/pages/zCoDXMLforCoDCommands?OpenDocument) .	V1R10	X		
HWI_CMD_SYSRESET_IPLT F (15)	Reset an image if the IPL token matches the specified IPLT	V1R11		X	
HWI_CMD_ACTIVATE_WITH_ACTPROF 10 (16)	Activate using the specified activation profile	V1R11	X	X	

Table 168. HWICMD types (continued)					
Command type / Constant with hexadecimal and (decimal) values	Description	Starting z/OS release	CPC	Image	User-defined Image Group
HWI_CMD_POWER_CONTROL 11 (17)	Specify power control characteristics	V1R10	X		
HWI_CMD_SCSI_LOAD 12 (18)	IPL Linux operating system or systems	<ul style="list-style-type: none"> Image: V1R12 User-defined image group: V1R13 		X	X
HWI_CMD_SCSI_DUMP 13 (19)	Dump a Linux operating system	V1R12		X	
HWI_CMD_SYSPLEX_TIME_SWAP_CTS 14 (20)	Swap the role of current time server (CTS) in a configured STP-only coordinated timing network (CTN) from preferred time server to backup time server or vice versa	V1R13	X		
HWI_CMD_SYSPLEX_TIME_SET_STP_CONFIG 15 (21)	Set the configuration for an STP-only coordinated timing network (CTN)	V1R13	X		
HWI_CMD_SYSPLEX_TIME_CHANGE_STP_ONLY_CTN 16 (22)	Change the STP_ID portion of the CTN ID for an entire STP-only coordinated timing network (CTN)	V1R13	X		
HWI_CMD_SYSPLEX_TIME_JOIN_STP_ONLY_CTN 17 (23)	Allow a CPC to join an STP-only coordinated timing network (CTN)	V1R13	X		
HWI_CMD_SYSPLEX_TIME_LEAVE_STP_ONLY_CTN 18 (24)	Remove a CPC from an STP-only coordinated timing network (CTN)	V1R13	X		

HWIEVENT

This table shows the BCPii HWIEVENT types and the objects that can be registered or unregistered for each event.

Table 169. HWIEVENT types					
Event ID / Bit position in structure specified (non-REXX)	Description	Starting z/OS release	CPC	Image	
Hwi_Event_CmdResp 97	Notice of command completion from the SE	V1R10	X	X	
Hwi_Event_StatusChg 98	Object status change	V1R10	X	X	
Hwi_Event_NameChg 99	Object name change	V1R10	X	X	
Hwi_Event_ActProfChg 100	Object has changed associated activation profile	V1R10	X	X	

Table 169. HWIEVENT types (continued)

Event ID / Bit position in structure specified (non-REXX)	Description	Starting z/OS release	CPC	Image
Hwi_Event_ObjCreate 101	New object has been defined	V1R10	X	X
Hwi_Event_ObjDestroy 102	Object has been undefined	V1R10	X	X
Hwi_Event_ObjException 103	Object has entered into or out of an exception state	V1R10	X	X
Hwi_Event_ApplStarted 104	Console application has started	V1R10	X	
Hwi_Event_ApplEnded 105	Console application is ending	V1R10	X	
Hwi_Event_HwMsg 106	Hardware message associated has been issued	V1R10	X	
Hwi_Event_HwMsgDel 107	Hardware message has been deleted	V1R10	X	
Hwi_Event_SecurityEvent 108	Security event has been logged	V1R10	X	
Hwi_Event_CapacityChg 109	Processing capacity has changed in some manner	V1R10	X	
Hwi_Event_CapacityRecord 110	A change has occurred to a temporary capacity record	V1R10	X	
Hwi_Event_OpSysMsg 111	Operating system message has been issued	V1R10		X
Hwi_Event_HwCommError 112	Hardware communication error received	V1R10	X	
Hwi_Event_BCPIIStatus 113	BCPii address space has stopped or started	V1R10		X
Hwi_Event_DisabledWait 114	An image has entered a disabled wait state	V1R10		X
Hwi_Event_PowerChange 115	Power characteristic or characteristics have changed	V1R10	X	
Hwi_Event_Shutdown 116	SHUTDOWN event has been issued. The shutdown event was introduced in z16 and indicates that the Support Element associated with the target CPC will be shutting down or restarting soon.	z/OS 2.5	X	

HWIQUERY and HWISET / HWISET2 attributes

This table shows the BCPii HWIQUERY and HWISET / HWISET2 attributes and the objects that can be targeted for each function. Note: The HWMCA attribute suffix refers to the 'HWMCA Object Attribute ID suffix' documented in *IBM z SNMP Application Programming Interfaces* (SB10-7171-06).

Table 170. HWIQUERY and HWISET / HWISET2 attributes													
Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_NAME 1 (1)	Name	V1R10		X	X		X	X	X	X	X	X	1.0
HWI_ERRSTAT 2 (2)	Status error (Y/N)	V1R10		X	X					X			7.0
HWI_BUSYSTAT 3 (3)	Busy status (Y/N)	V1R10		X	X					X			8.0
HWI_MSGSTAT 4 (4)	Messages present (Y/N)	V1R10		X	X								9.0
HWI_OPERSTAT 5 (5)	Current status	V1R10		X	X								10.0
HWI_ACCEPTAT 6 (6)	Acceptable status values	V1R10	X	X	X								11.0
HWI_APROF 7 (7)	Next reset activation profile name	V1R10	X	X	X								13.0
HWI_LUAPROF 8 (8)	Last used activation profile name	V1R10		X	X								14.0
HWI_OBJTYPE 9 (9)	Object type	V1R10		X	X	X	X	X	X	X	X	X	22.0
HWI_IMLMODE A (10)	IML mode	V1R10		X	X								12.0
HWI_IPADDR 17 (23)	Internet address (IPv4 format)	V1R10		X									15.0
HWI_SNAADDR 18 (24)	SNA address (netid.nau)	V1R10		X									16.0
HWI_MMODEL 19 (25)	Machine model	V1R10		X									17.0
HWI_MTYPE 1A (26)	Machine type	V1R10		X									18.0

HWIQUERY and HWISET attributes

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_MSERIAL 1B (27)	Machine serial	V1R10		X									19.0
HWI_CPSERIAL 1C (28)	CPC serial number	V1R10		X									20.0
HWI_CPCID 1D (29)	CPC identifier	V1R10		X									21.0
HWI_RESERVEID 1E (30)	Name of application holding reserve	V1R10		X									44.0
HWI_SVCEREQD 1F (31)	Service required (Y/N)	V1R10		X									46.0
HWI_CBUINST 20 (32)	CBU installed (Y/N)	V1R10		X									32.0
HWI_CBUE NABLD 21 (33)	CBU enabled (Y/N)	V1R10		X									48.0
HWI_CBUACTIVE 22 (34)	CBU activated (Y/N)	V1R10		X									33.0
HWI_CBUACTDT 23 (35)	CBU activation date	V1R10		X									34.0
HWI_CBUEXPDT 24 (36)	CBU expiration date	V1R10		X									35.0
HWI_CBUESTAR 25 (37)	CBU test activations remaining	V1R10		X									36.0
HWI_CBUREALAV 26 (38)	Real CBU activation available (Y/N)	V1R10		X									37.0
HWI_PRUNTYPE 27 (39)	Processor running time type	V1R10	X	X			X						78.0
HWI_PRUNTIME 28 (40)	Processor running time	V1R10	X	X			X						79.0
HWI_PRUNTSEW 29 (41)	Processor loses its running time slice when in wait state (Y/N)	V1R10	X	X			X						80.0

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Startin g z/OS release	Settabl e using HWISET or HWISET2	CPC	Image	CapRe c	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_OOCI NST 2A (42)	On/Off on Demand installed (Y/N)	V1R10		X									87.0
HWI_OOC ACT 2B (43)	On/Off on Demand activated (Y/N)	V1R10		X									88.0
HWI_OOC ENAB 2C (44)	On/Off on Demand enabled (Y/N)	V1R10		X									89.0
HWI_OOC ADT 2D (45)	On/Off on Demand activation date	V1R10		X									90.0
HWI_PCPC SWM 2E (46)	Permanent CPC software model	V1R10		X									120.0
HWI_PPBP SWM 2F (47)	Permanent plus billable processor software model	V1R10		X									121.0
HWI_PPTP SWM 30 (48)	Permanent plus (all) temporary processor software model	V1R10		X									122.0
HWI_PCPC MSU 31 (49)	CPC millions of service units (MSU) value	V1R10		X									123.0
HWI_PPBP MSU 32 (50)	Permanent plus billable processor MSU value	V1R10		X									124.0
HWI_PPTP MSU 33 (51)	Permanent plus (all) temporary processor MSU value	V1R10		X									125.0
HWI_NUM GPP 34 (52)	Number of general purpose processors	V1R10		X									126.0
HWI_NUM SAP 35 (53)	Number of service assist processors	V1R10		X									127.0
HWI_NUM IFAP 36 (54)	Number of integrated facility for applications (IFA) processors	V1R10		X									128.0
HWI_NUM IFLP 37 (55)	Number of integrated facility for Linux (IFL) processors	V1R10		X									129.0
HWI_NUM ICFP 38 (56)	Number of internal coupling facility (ICF) processors	V1R10		X									130.0

HWIQUERY and HWISET attributes

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_NUM IIPP 39 (57)	Number of integrated information (IIP) processors	V1R10		X									131.0
HWI_NUM FLTYP 3A (58)	Number of defective (faulty) processors	V1R10		X									132.0
HWI_NUM SPARE 3B (59)	Number of spare processors	V1R10		X									133.0
HWI_NUM PENDP 3C (60)	Number of pending (activation) processors	V1R10		X									134.0
HWI_ CAPCHGAL LWD 3D (61)	Allow temporary capacity change (Y/N)	V1R10		X									149.0
HWI_DGR STAT 3E (62)	Degraded status	V1R10		X									47.0
HWI_ CURRPPO WERMODE 3F (63)	Current processor power savings mode activated	V1R10		X									190.0
HWI_ SUPPPPO WERMODE 40 (64)	Supported processor power savings modes available	V1R10		X									191.0
HWI_STPC ONFIG 41 (65)	Server Timer Protocol (STP) configuration data	V1R12		X									165.0
HWI_NUM PGPP 42 (66)	Number of pending general purpose processors	V1R12		X									175.0
HWI_NUM PSAP 43 (67)	Number of pending service assist processors	V1R12		X									176.0
HWI_NUM PAAP 44 (68)	Number of pending Application Assist (AAP) processors	V1R12		X									177.0
HWI_NUM PIFLP 45 (69)	Number of pending Integrated Facility for Linux (IFL) processors	V1R12		X									178.0
HWI_NUM PICFP 46 (70)	Number of pending Internal Coupling Facility (ICF) processors	V1R12		X									179.0

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_NUMPIIPP 47 (71)	Number of pending Integrated Information (IIP) processors	V1R12		X									180.0
HWI_POWERMODE ALLOWED 48 (72)	Processor power savings mode allowed (Y/N)	V1R10		X									193.0
HWI_VERSION 49 (73)	CPC version number	V1R13		X									151.0
HWI_EC_MCL_INFO 4A (74)	XML string that describes the Engineering Change (EC) and Microcode Level (MCL) levels	V1R13		X									162.0
HWI_LIST_IP_ADDRESSES 4B (75)	All the IP addresses (in IPv4 and/or IPv6 format)	V1R13		X									161.0
HWI_AUTO_SWITCH_ENABLED 4C (76)	Automatic switching between primary and alternate support elements enabled (Y/N)	V1R13		X									163.0
HWI_CPCNAME 69 (105)	Parent (CPC) name	V1R10			X								2.0
HWI_OSNAME 6A (106)	Operating system name	V1R10			X								3.0
HWI_OSTYPE 6B (107)	SW operating system type (CFCC, MVS, VM, LINUX, VSE, Z TPF EE)	V1R10			X								4.0
HWI_OSLEVEL 6C (108)	SW operating system level	V1R10			X								5.0
HWI_SYSPLEX 6D (109)	SW sysplex name	V1R10			X								6.0
HWI_CLUSTER 6E (110)	LPAR cluster name	V1R10			X								49.0
HWI_PARTITIONID 6F (111)	Partition ID	V1R10			X			X					51.0

HWIQUERY and HWISET attributes

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_DEFCAP 70 (112)	Current defined	V1R10	X		X			X					43.0
HWI_SGPIPW 71 (113)	Shared general processor initial processing weight	V1R10	X		X			X					30.0
HWI_SGPIPWCAP 72 (114)	SGPIPW capped (Y/N)	V1R10	X		X			X					31.0
HWI_SGPPWMIN 73 (115)	Minimum SGPPW value	V1R10	X		X			X					38.0
HWI_SGPPWMAX 74 (116)	Maximum SGPPW value	V1R10	X		X			X					39.0
HWI_SGPPW 75 (117)	Current SGPPW value	V1R10			X								41.0
HWI_SGPPWCAP 76 (118)	SGPPW capped (Y/N)	V1R10			X								42.0
HWI_WLM 77 (119)	WLM allowed to change processing weight related attributes (Y/N)	V1R10	X		X			X					40.0
HWI_IFAIPW 78 (120)	Integrated facility for applications initial processing weight	V1R10	X		X			X					60.0
HWI_IFAIPWCAP 79 (121)	IFAIPW capped (Y/N)	V1R10	X		X			X					61.0
HWI_IFAPWMIN 7A (122)	Minimum IFAPW value	V1R10	X		X			X					62.0
HWI_IFAPWMAX 7B (123)	Maximum IFAPW value	V1R10	X		X			X					63.0
HWI_IFAPW 7C (124)	Current IFAPW value	V1R10			X								64.0
HWI_IFAPWCAP 7D (125)	IFAPW capped (Y/N)	V1R10			X								65.0
HWI_IFLIPW 7E (126)	Integrated facility for Linux initial processing weight	V1R10	X		X			X					66.0

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_IFLPWCAP 7F (127)	IFLPW capped (Y/N)	V1R10	X		X			X					67.0
HWI_IFLPWMIN 80 (128)	Minimum IFLPW value	V1R10	X		X			X					68.0
HWI_IFLPWMAX 81 (129)	Maximum IFLPW value	V1R10	X		X			X					69.0
HWI_IFLPW 82 (130)	Current IFLPW value	V1R10			X								70.0
HWI_IFLPWCAP 83 (131)	IFLPW capped (Y/N)	V1R10			X								71.0
HWI_ICFIPW 84 (132)	Internal coupling facility initial processing weight	V1R10	X		X			X					72.0
HWI_ICFIPWCAP 85 (133)	ICFIPW capped (Y/N)	V1R10	X		X			X					73.0
HWI_ICFPWMIN 86 (134)	Minimum ICFPW value	V1R10	X		X			X					74.0
HWI_ICFPWMAX 87 (135)	Maximum ICFPW value	V1R10	X		X			X					75.0
HWI_ICFPW 88 (136)	Current ICFPW value	V1R10			X								76.0
HWI_ICFPWCAP 89 (137)	ICFPW capped (Y/N)	V1R10			X								77.0
HWI_IIPIPW 8A (138)	Integrated information processors initial processing weight	V1R10	X		X			X					81.0
HWI_IIPIPWCAP 8B (139)	IIPIPW capped (Y/N)	V1R10	X		X			X					82.0
HWI_IIPPWMIN 8C (140)	Minimum IIPPW value	V1R10	X		X			X					83.0
HWI_IIPPWMAX 8D (141)	Maximum IIPPW value	V1R10	X		X			X					84.0

HWIQUERY and HWISET attributes

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)													
Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_IIPPW 8E (142)	Current IIPPW value	V1R10			X								85.0
HWI_IIPPWCAP 8F (143)	IIPPW capped (Y/N)	V1R10			X								86.0
HWI_IPLTOKEN 90 (144)	IPL token associated with the current IPL of the image	V1R11			X								164.0
HWI_PSW 91 (145)	PSW for each CP associated with the image	V1R11			X								150.0
HWI_GROUP_PROFILE_CAPACITY 92 (146)	Workload unit for the group profile associated with an image	V1R13	X		X						X	X	1
HWI_LAST_USED_LOADADDR 93 (147)	Last-used load address	V1R13			X								201.0
HWI_LAST_USED_LOADPARAM 94 (148)	Last-used load parameters	V1R13			X								202.0
HWI_ABSCAP 95 (149)	Absolute capping enablement (GPP) (Y/N)	V2R1	X		X			X					217.0
HWI_ABSCAPVAL 96 (150)	Absolute capping value (GPP)	V2R1	X		X			X					218.0
HWI_IFAABSCAP 97 (151)	Absolute capping enablement (AAP) (Y/N)	V2R1	X		X			X					219.0
HWI_IFAABSCAPVAL 98 (152)	Absolute capping value (AAP)	V2R1	X		X			X					220.0
HWI_IFLABSCAP 99 (153)	Absolute capping enablement (IFL) (Y/N)	V2R1	X		X			X					221.0
HWI_IFLABSCAPVAL 9A (154)	Absolute capping value (IFL)	V2R1	X		X			X					222.0

¹ The HWMCA attribute suffix is 92.0 for Group profile connection and it is 192.0 for other applicable connections

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_ICFABSCAP 9B (155)	Absolute capping enablement (ICF) (Y/N)	V2R1	X		X			X					223.0
HWI_ICFABSCAPVAL 9C (156)	Absolute capping value (ICF)	V2R1	X		X			X					224.0
HWI_IIPABSCAP 9D (157)	Absolute capping enablement (IIP) (Y/N)	V2R1	X		X			X					225.0
HWI_IIPABSCAPVAL 9E (158)	Absolute capping value (IIP)	V2R1	X		X			X					226.0
HWI_GROUP_PROF_ABSCAP 9F (159)	Absolute capping enablement (GPP) (Y/N)	V2R3	X		X						X	X	227.0
HWI_GROUP_PROF_ABSCAPVAL A0 (160)	Absolute capping value (GPP)	V2R3	X		X						X	X	228.0
HWI_GROUP_PROF_ICFABSCAP A1 (161)	Absolute capping enablement (ICF) (Y/N)	V2R3	X		X						X	X	229.0
HWI_GROUP_PROF_ICFABSCAPVAL A2 (162)	Absolute capping value (ICF)	V2R3	X		X						X	X	230.0
HWI_GROUP_PROF_IFLABSCAP A3 (163)	Absolute capping enablement (IFL) (Y/N)	V2R3	X		X						X	X	231.0
HWI_GROUP_PROF_IFLABSCAPVAL A4 (164)	Absolute capping value (IFL)	V2R3	X		X						X	X	232.0
HWI_GROUP_PROF_IIPABSCAP A5 (165)	Absolute capping enablement (IIP) (Y/N)	V2R3	X		X						X	X	233.0
HWI_GROUP_PROF_IIPABSCAPVAL A6 (166)	Absolute capping value (IIP)	V2R3	X		X						X	X	234.0

HWIQUERY and HWISET attributes

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_RECORD B7 (183)	Record ID	V1R10				X							135.0
HWI_RECORD TYPE B8 (184)	Record type	V1R10				X							136.0
HWI_ACTIVATION STATUS B9 (185)	Record activation status	V1R10				X							137.0
HWI_ACTIVATION DATE BA (186)	Record activation date	V1R10				X							138.0
HWI_EXPIRATION DATE BB (187)	Record expiration date	V1R10				X							139.0
HWI_ACTIVATION EXPIRATION DATE BC (188)	Record activation expiration date	V1R10				X							140.0
HWI_MAXIMUM REAL ACTIVATION DAYS BD (189)	Maximum real activation days	V1R10				X							141.0
HWI_MAXIMUM TEST ACTIVATION DAYS BE (190)	Maximum test activation days	V1R10				X							142.0
HWI_REMAINING REAL ACTIVATION DAYS BF (191)	Remaining real activation days	V1R10				X							143.0
HWI_REMAINING TEST ACTIVATION DAYS C0 (192)	Remaining test activation days	V1R10				X							144.0
HWI_CAPACITY RECORD IN XML FORMAT C1 (193)	Capacity record in XML format	V1R10				X							N/A
HWI_IOCD S C9 (201)	IOCD S	V1R11	X				X						27.0
HWI_IPL_ ADDRESS CA (202)	IPL address	V1R11	X					X	X				28.0
HWI_IPL_ PARAM CB (203)	IPL parameter	V1R11	X					X	X				29.0
HWI_IPL_ TYPE CC (204)	IPL type	V1R11	X					X	X				52.0

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_WW_PORTNAME CD (205)	Worldwide port name	V1R11	X					X	X				53.0
HWI_BOOT_PGM_SELECTOR CE (206)	Boot program selector	V1R11	X					X	X				54.0
HWI_LU_NUMBER CF (207)	Logical unit number value	V1R11	X					X	X				55.0
HWI_BOOTREC_BLK_ADDRESS D0 (208)	Boot record logical block address	V1R11	X					X	X				56.0
HWI_OPSYS_LOADPARAM D1 (209)	Operating system specific load parameter	V1R11	X					X	X				57.0
HWI_GROUP_PROFILE_NAME D2 (210)	Name of group profile to be used for image	V1R11	X		X			X					93.0
HWI_LOAD_ACTIVATION D3 (211)	Image loaded (IPLed) after activation (Y/N)	V1R11	X					X					94.0
HWI_CENTRAL_STORAGE D4 (212)	Initial amount of central storage (in MB) for image	V1R11	X					X					95.0
HWI_RESERVED_CENTRAL_STORAGE D5 (213)	Reserved amount of central storage (in MB) for image	V1R11	X					X					96.0
HWI_EXPANDED_STORAGE D6 (214)	Initial amount of expanded storage (in MB) for image	V1R11	X					X					97.0
HWI_RESERVED_EXPANDED_STORAGE D7 (215)	Reserved amount of expanded storage (in MB) for image	V1R11	X					X					98.0
HWI_NUM_GPP D8 (216)	Number of dedicated general purpose processors for image	V1R11	X					X					99.0

HWIQUERY and HWISET attributes

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)													
Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_NUM_RESGPPD9 (217)	Number of reserved dedicated general purpose processors for image	V1R11	X					X					100.0
HWI_NUM_IFADA (218)	Number of dedicated integrated facility for applications (IFA) processors for image	V1R11	X					X					101.0
HWI_NUM_RESIFADB (219)	Number of reserved dedicated integrated facility for applications (IFA) processors for image	V1R11	X					X					102.0
HWI_NUM_IFLDC (220)	Number of dedicated integrated facility for Linux (IFL) processors for image	V1R11	X					X					103.0
HWI_NUM_RESIFLDD (221)	Number of reserved dedicated integrated facility for Linux (IFL) processors for image	V1R11	X					X					104.0
HWI_NUM_ICFDE (222)	Number of dedicated internal coupling facility (ICF) processors for image	V1R11	X					X					105.0
HWI_NUM_RESICFDF (223)	Number of reserved dedicated internal coupling facility (ICF) processors for image	V1R11	X					X					106.0
HWI_NUM_ZIIP E0 (224)	Number of dedicated z System integration information processors (zIIPs) for image	V1R11	X					X					107.0
HWI_NUM_RESZIIP E1 (225)	Number of reserved dedicated z System integration information processors (zIIPs) for image	V1R11	X					X					108.0
HWI_NUM_SHARED_GPPE2 (226)	Number of shared general purpose processors for image	V1R11	X					X					109.0

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_NUM_RES_SHARED_GPP E3 (227)	Number of reserved shared general purpose processors for image	V1R11	X					X					110.0
HWI_NUM_SHARED_IFA E4 (228)	Number of shared integrated facility for applications (IFA) processors for image	V1R11	X					X					111.0
HWI_NUM_RES_SHARED_IFA E5 (229)	Number of reserved shared integrated facility for applications (IFA) processors for image	V1R11	X					X					112.0
HWI_NUM_SHARED_IFL E6 (230)	Number of shared integrated facility for Linux (IFL) processors for image	V1R11	X					X					113.0
HWI_NUM_RES_SHARED_IFL E7 (231)	Number of reserved shared integrated facility for Linux (IFL) processors for image	V1R11	X					X					114.0
HWI_NUM_SHARED_ICF E8 (232)	Number of shared internal coupling facility (ICF) processors for image	V1R11	X					X					115.0
HWI_NUM_RES_SHARED_ICF E9 (233)	Number of reserved shared internal coupling facility (ICF) processors for image	V1R11	X					X					116.0
HWI_NUM_SHARED_ZIIP EA (234)	Number of shared z System integrated information processors (zIIPs) for image	V1R11	X					X					117.0
HWI_NUM_RES_SHARED_ZIIP EB (235)	Number of reserved shared z System integrated information processors (zIIPs) for image	V1R11	X					X					118.0
HWI_BASIC_CPU_AUTH_COUNT_CNTL EC (236)	Basic CPU counter facility for the image enabled (Y/N)	V1R12	X					X					168.0

HWIQUERY and HWISET attributes

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_PROBSTATE_CPU_AUTH_COUNT_CNTL ED (237)	Problem state CPU counter facility for the image enabled (Y/N)	V1R12	X					X					169.0
HWI_CRYPTOCOUNTER_ACTIVITY_CPU_AUTH_COUNT_CNTL EE (238)	Crypto activity CPU counter facility for the image enabled (Y/N)	V1R12	X					X					170.0
HWI_EXTENDED_CPU_AUTH_COUNT_CNTL EF (239)	Extended CPU counter facility for the image enabled (Y/N)	V1R12	X					X					171.0
HWI_COPROCESSOR_CPU_AUTH_COUNT_CNTL FO (240)	Coprocessor group CPU counter facility for the image enabled (Y/N)	V1R12	X					X					172.0
HWI_BASIC_CPU_SAMPLING_AUTH_CNTL F1 (241)	Basic CP CPU sampling facility for the image enabled (Y/N)	V1R12	X					X					173.0
HWI_APROF_STORE_STATUS F2 (242)	Store status selected (Y/N)	V1R11	X						X				166.0
HWI_APROF_LOADTYPE F3 (243)	Type of load requested	V1R11	X						X		X	X	167.0
HWI_PROFILE_DESCRIPTION F4 (244)	Activation profile description	V1R13	X				X	X	X		X	²	203.0
HWI_PARTITION_ID F5 (245)	Partition identifier for AProf	V1R13	X					X					51.0

² HWI_PROFILE_DESCRIPTION is NOT settable for LPAR Capacity Group

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_OPERATING_MODE F6 (246)	Operating mode value for AProf	V1R13	X					X					204.0
HWI_CLOCK_TYPE F7 (247)	Clock type assignment (time source setting)	V1R13	X					X					205.0
HWI_TIME_OFFSET_DAYS F8 (248)	Number of days currently set as offset from external time source's TOD	V1R13	X					X					206.0
HWI_TIME_OFFSET_HOURS F9 (249)	Number of hours currently set as offset from external time source's TOD	V1R13	X					X					207.0
HWI_TIME_OFFSET_MINUTES FA (250)	Number of minutes currently set as offset from external time source's TOD	V1R13	X					X					208.0
HWI_TIME_OFFSET_INCREASE FB (251)	Local time zone: TRUE means east of GMT; FALSE means west of GMT	V1R13	X					X					209.0
HWI_LICCV_VALIDATION_ENABLED FC (252)	Activation profile must conform to the current LICCV configuration (Y/N)	V1R13	X					X					210.0
HWI_GLOBAL_PERFORMANCE_DATA_CONTROL FD (253)	LPAR can be used to view processing unit activity data for all other LPARs on the same CPC (Y/N)	V1R13	X					X					211.0
HWI_IO_CONFIGURATION_CONTROL FE (254)	LPAR can be used to read and write any IOCDS (Y/N)	V1R13	X					X					212.0
HWI_CROSS_PARTITION_AUTHORITY FF (255)	LPAR can be used to issue instructions that reset or deactivate other LPARs (Y/N)	V1R13						X					213.0

HWIREST attributes

Table 170. HWIQUERY and HWISET / HWISET2 attributes (continued)													
Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET or HWISET2	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	Group profile	LPAR Capacity group	HWMCA attribute suffix
HWI_LOGICAL_PARTITION_ISOLATION 100 (256)	Re-configurable channel paths assigned to LPAR are reserved for its exclusive use (Y/N)	V1R13	X					X					214.0

HWIREST attributes

For more information on HWIREST attributes, see [Record type 106 \(X'6A'\) – BCPii activity in z/OS MVS System Management Facilities](#).

Appendix C. General use C/C++ header files

C/C++ header files are shipped in z/OS V2R3 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* in the *z/OS Internet library* (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Member	Description
BLSCADPL	Describes same data areas as assembler macro BLSABDPL. Depends on BLSCDESC.
BLSCADSY	Describes same data areas as assembler macro BLSADSY.
BLSCCBSP	Describes same data areas as assembler macro BLSACBSP. Depends on BLSCDESC.
BLSCDESC	Describes same data areas as assembler macros BLSRDATC, BLSRDATS, BLSRDATT, BLSRESSY, and BLSRSASY. Many of the other members require that this header file be included before they are included.
BLSCDRPX	Describes same data areas as assembler macro BLSRDRPX. Depends on BLSCDESC.
BLSCNAMP	Describes same data areas as assembler macro BLSRNAMP. Depends on BLSCDESC.
BLSCPCQE	Describes same data areas as assembler macro BLSRPCQE. Depends on BLSCDESC.
BLSCPPR2	Describes same data areas as assembler macro BLSUPPR2.
BLSCPWHS	Describes same data areas as assembler macro BLSRPWHS. Depends on BLSCDESC.
BLSCXMSP	Describes same data areas as assembler macro BLSRXMSP. Depends on BLSCDESC.
BLSCXSSP	Describes same data areas as assembler macro BLSRXSSP. Depends on BLSCDESC.

Appendix D. Accessibility

Accessible publications for this product are offered through [IBM Documentation for z/OS \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS documentation see [How to Send Feedback to IBM](#) to leave documentation feedback.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan*

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.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who 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
Site Counsel
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 document 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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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 illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

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

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS \(www.ibm.com/software/support/systemsz/lifecycle\)](http://www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Additional notices

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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

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 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 information is intended to help the customer to write applications that use operating system services. This information documents general-use programming interface and associated guidance information provided by z/OS.

General-use programming interfaces allow the customer to write programs that obtain the services of z/OS.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.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 \(www.ibm.com/legal/copytrade.shtml\)](http://www.ibm.com/legal/copytrade.shtml).

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

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

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

Glossary

This glossary defines technical terms and abbreviations used in z/OS MVS documentation.

data object

A VSAM linear data set.

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.

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*.

hiperspace

A range of up to two gigabytes of virtual storage that a program can use like a buffer.

linear data set

A type of VSAM data set where data is stored as a linear string of bytes.

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.

object

See data object.

permanent data object

A virtual storage access method (VSAM) linear data set that resides on DASD (also called a data-in-virtual object).

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.

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.

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.

VSAM

Virtual storage access method.

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
 - temporary object [8](#)
- access to an object
 - terminating [18](#)
- accessibility
 - contact IBM [843](#)
- ADA programming language
 - example using window services [37](#)
- application
 - in resource recovery [107](#)
- application_backout_UR call
 - return and reason codes [113](#)
 - syntax [113](#)
- application_commit_UR call
 - return and reason codes [117](#)
 - syntax [116](#)
- assistive technologies [843](#)
- authentication types [704](#)
- authorized interfaces for zEDC [189](#), [196](#), [197](#), [212](#)

B

- back out changes to protected resources [111](#)
- BCPii REXX execs, setting up access to [259](#)
- BCPii REXX restrictions [265](#)
- BCPii REXX support [264](#), [265](#), [268](#), [269](#)
- blocks of an object
 - definition [3](#)
 - size [3](#)

C

- C programming language
 - call syntax for latch manager services [91](#)
 - example of reference pattern services [75](#)
 - example using window services [41](#)
- call statements for latch manager services [91](#)
- call statements for reference pattern services [71](#)
- call syntax
 - for latch manager service [91](#)
- CDA parser
 - linkage [709](#), [711](#)
 - syntax [709](#)
- CEA TSO/E address space services
 - CEATsoRequest API [129](#)
 - components [121](#)
 - diagnostic codes [158](#)
 - invoking [129](#), [143](#), [145](#), [146](#)
 - overview [121](#)
 - prerequisites [121](#)
 - reason codes [149](#)
 - request types
 - CeaTsoAttn [135](#)
 - CeaTsoEnd [136](#)
 - CeaTsoPing [137](#)

- CEA TSO/E address space services (*continued*)
 - request types (*continued*)
 - CeaTsoQuery [138](#)
 - CeaTsoQueryApp [141](#)
 - CeaTsoStart [134](#)
 - requirements for callers [134](#)
 - return codes [148](#)
 - TSO/E address spaces [121](#)
- changed data in an object
 - refreshing [16](#)
- client web enablement toolkit
 - HTTP/HTTPS protocol enabler [575](#)
 - JSON parser [467](#)
- Cloud Data Access (CDA) [685](#)
- Cloud Data Access (CDA) API basics [699](#)
- Cloud Data Access (CDA) Services [683](#)
- Cloud Data Access Cloud Credential storage [701](#)
- Cloud Data Access Configuration [687](#)
- Cloud Data Access files [691](#)
- COBOL programming language
 - call syntax for latch manager services [91](#)
 - example using reference pattern services [77](#)
 - example using window services [43](#)
- commit changes to protected resources [114](#)
- commit protocol, two-phase [108](#)
- compression service
 - memory registration [202](#)
 - Rendezvous [197](#), [200](#)
 - single compression request [206](#)
 - unregister memory [204](#)
 - unrendezvous [211](#)
- Config file [691](#)
- contact
 - z/OS [843](#)
- CPC names
 - dynamic modification of [257](#)
- Credential panels [704](#)
- Credential panels for different authentication types [704](#)
- CSRIDAC callable service [22](#)
- CSRIRP callable service
 - example [69](#)
- CSRL16J callable service
 - entry characteristics for the target routine [221](#)
 - freeing dynamic storage for the target routine [222](#)
 - programming requirements [222](#)
 - return codes [226](#)
- CSRL16J/CSRLJ1 callable service
 - parameter description [221](#)
 - syntax [221](#)
- CSRREFR callable service [26](#)
- CSRRRP callable service [73](#)
- CSRSAVE callable service [28](#)
- CSRSCOT callable service [30](#)
- CSRSIC include file [234](#)
- CSRVIEW callable service [32](#)

D

data compression [187–189](#)
data object
 mapping [3](#)
 obtaining access [10](#)
 structure [3](#)
data to be viewed
 identifying [13](#)
data-in-virtual object [3](#)
DFP requirement for window services [10](#)

E

Error conditions [703](#)
examples
 data object mapped to a window [3](#)
 structure of a data object [3](#)

F

FORTRAN programming language
 call syntax for latch manager services [91](#)
 example using reference pattern services [81](#)
 example using window services [46](#)
FPZ4ABC [206](#)
FPZ4DMR [204](#)
FPZ4PRB [200](#)
FPZ4RMR [202](#)
FPZ4RZV [197](#)
FPZ4URZ [211](#)

G

gap in reference pattern services
 defining [63](#)
 definition [63](#)
GDKDEL [742](#)
GDKDEL — Delete an object from cloud storage [742](#)
GDKGEN [792](#)
GDKGEN - Perform a Configurable request [792](#)
GDKGET [716](#)
GDKGET — Retrieve a cloud object [716](#)
GDKGETP [765](#)
GDKGETP — List cloud providers [765](#)
GDKINIT [785](#)
GDKINIT - Initialize a Session [785](#)
GDKKEYAD [773](#)
GDKKEYAD — Store cloud credentials [773](#)
gdkkeyadj [813](#)
gdkkeyadj — Store cloud credentials JSON API [813](#)
GDKKEYDL [778](#)
GDKKEYDL - Delete cloud credentials [778](#)
gdkkeydlj [815](#)
gdkkeydlj — delete cloud credentials json api [815](#)
GDKKEYGR [781](#)
GDKKEYGR - List resources for provider [781](#)
gdkkeygrj [817](#)
gdkkeygrj — list cloud credentials resources JSON API [817](#)
GDKKEYSR [769](#)
GDKKEYSR — Retrieve cloud credentials [769](#)
gdkkeysrj [811](#)
gdkkeysrj — Retrieve cloud credentials JSON API [811](#)

GDKLIST [748](#)
GDKLIST — List cloud objects [748](#)
GDKLISTL [755](#)
GDKLISTL — List cloud objects [755](#)
GDKMSGTR [762](#)
GDKMSGTR — Translate a CDA return code into text [762](#)
GDKQUERY [807](#)
GDKQUERY - Query available CDA functions [807](#)
GDKTERM [789](#)
GDKTERM - Terminate a Session [789](#)
GDKVALD [802](#)
GDKVALD - Validate Provider File [802](#)
GDKWRITE [728](#)
GDKWRITE — Write an object to cloud storage [728](#)
glossary of terms [849](#)

H

HTTP/HTTPS enabler
 Applications executed from a TSO/E operation environment [659](#)
 applications executed from z/OS UNIX operation environment [657](#)
 callable services [594](#)
 Capturing trace data through environment variables [657](#), [659](#)
 options [643](#)
 options for connections only [643](#)
 options for requests onlySure [653](#), [656](#)
HTTP/HTTPS enablerApplications executed from a batch job
 Capturing trace data through environment variables [657](#), [659](#)
HTTP/HTTPS protocol enabler
 availability [577](#)
 elements of [576](#)
 linkage [577](#), [578](#)
 programming considerations
 environment [579](#)
 problem determination [583](#)
 recovery [584](#)
 security [580](#)
 programming examples [589](#)
 syntax [577](#)
HWIREXX
 return codes [266](#)
 setting up access to [259](#)
HWTCONST [477](#), [514](#), [595](#)
HWTCONN [596](#)
HWTDISC [602](#)
HWTINIT [609](#)
HWTIRQST [613](#)
HWTIRSET [619](#)
HWTIRSET [624](#)
HWTIRSLST [631](#)
HWTIRTERM [638](#)
HWTJCREN [478](#)
HWTJDEL [490](#)
HWTJESCT [497](#)
HWTJGAEN [498](#)
HWTJGBOV [502](#)
HWTJGENC [506](#)
HWTJGJST [510](#)
HWTJGNUE [515](#)

[HWTJGNOV 519](#)
[HWTJGOEN 525](#)
[HWTJGVAL 531](#)
[HWTJINIT 535](#)
[HWTJOPTS 539](#)
[HWTJPARS 544](#)
[HWTJSENC 551](#)
[HWTJSERI 555](#)
[HWTJSRCH 561](#)
[HWTJTERM 569](#)

I

identifying data object [10](#)
IEAAFFN callable service
 parameter descriptions [219](#)
 purpose [219](#)
 requirements [220](#)
 restrictions and limitations [219](#)
 return codes [220](#)
 syntax [219](#)
IFAMCON [667](#)
IFAMDSC [670](#)
IFAMGET [673](#)
IFAMQRY [677](#)
interim changes to a permanent object
 saving [15](#)
Introduction to DFSMSdfp Cloud Data Access (CDA) [685](#)
ISGLCRT callable service
 syntax [91](#)
ISGLOBT callable service
 syntax [95](#)
ISGLPBA callable service
 syntax [102](#)
ISGLPRG callable service
 syntax [101](#)
ISGLREL callable service
 syntax [98](#)
ISV-provided REXX programming restrictions [269](#)
ISV-provided REXX support [268](#)

J

JavaScript Object Notation (JSON) [467](#)
JSON parser
 availability [469](#)
 callable services [475](#)
 elements of [468](#)
 linkage [470](#), [471](#)
 programming considerations [470](#), [471](#)
 syntax [470](#)

K

Key file [691](#)
keyboard
 navigation [843](#)
 PF keys [843](#)
 shortcut keys [843](#)

L

latch manager services

latch manager services (*continued*)
 ISGLCRT callable service
 syntax [91](#)
 ISGLOBT callable service
 syntax [95](#)
 ISGLPBA callable service
 syntax [102](#)
 ISGLPRG callable service
 syntax [101](#)
 ISGLREL callable service
 syntax [98](#)

M

multiple views of an object
 defining [14](#)

N

navigation
 keyboard [843](#)

P

Pascal programming language
 example using window services [50](#), [83](#)
permanent object
 definition [3](#)
 maximum size [3](#)
 relationship to a data-in-virtual object
 [3](#)
 structure [3](#)
PL/I programming language
 call syntax for latch manager services [91](#)
 example using window services [53](#)
processor affinity [219](#)
protected
 resource [107](#)
Provider file [692](#)

R

reference information [71](#), [91](#)
reference pattern services
 coding examples
 C programming language [75](#)
 COBOL programming language [77](#)
 FORTRAN programming language [81](#)
 Pascal programming language [83](#)
 overview [61](#)
 use with data window services [13](#)
 using [65](#)
reference unit in reference pattern services
 choosing [63](#)
 definition [63](#)
REPLACE option for a window [12](#)
resource
 process for protecting [108](#)
 protecting [107](#)
 protection on multiple systems [110](#)
 requesting protection [110](#)
resource manager
 in resource recovery [107](#)

- resource recovery
 - distributed [110](#)
 - process [108](#)
 - programs [107](#)
 - requesting [110](#)
 - service [111](#), [114](#)
- RETAIN option for a window [12](#)
- REXX programming language
 - call syntax for latch manager services [91](#)
- REXX restrictions [265](#)
- REXX support [264](#), [265](#)
- RRS
 - application_backout_UR call [111](#)
 - application_commit_UR call [114](#)
 - as sync-point manager [107](#)

S

- server identity [593](#)
- shortcut keys [843](#)
- size of an object
 - extending [14](#)
- SMF services
 - SMF real-time interface [667](#)
- SMS requirement for window services [10](#)
- structure of a data object [3](#)
- summary of changes [xxv](#)
- sync-point manager
 - in resource recovery [107](#)
- System Administrator Configuration Quick-Start [687](#)

T

- temporary object
 - definition [3](#)
 - functions supported [8](#)
 - maximum size [3](#)
 - overview of supported functions [8](#)
 - structure [3](#)
- terminology [849](#)
- transferring control with all registers intact
 - CSRL16J/CSRLJ1 [221](#)
- TSO/E REXX programming restrictions [268](#)
- TSO/E REXX support [268](#)
- two-phase commit protocol [108](#)

U

- UR (unit of recovery)
 - backing out [111](#)
 - committing [114](#)
- User Configuration Quick-Start [689](#)
- user interface
 - ISPF [843](#)
 - TSO/E [843](#)
- using protected resources [107](#)

V

- view of an object
 - terminating [17](#)

W

- ways that window services can map an object [4](#)
- what window services provides [4](#)
- window
 - definition [3](#)
 - use [3](#)
- window services
 - call statements [19](#)
 - COBOL programming language [43](#)
 - coding examples
 - ADA programming language [37](#)
 - C programming language [41](#)
 - FORTRAN programming language [46](#)
 - Pascal programming language [50](#)
 - PL/I programming language [53](#)
 - functions provided [4](#)
 - handling abends [18](#)
 - handling return codes [18](#)
 - reference information [19](#)
 - services provided [4](#)
 - ways to map an object [4](#)
- window services overview [3](#)

Z

- z/OS client web enablement toolkit, *See* client web enablement toolkit
- zEDC [187–189](#), [215](#)
- zEDC Express [187–189](#)
- zEnterprise Data Compression (zEDC)
 - requirements [188](#)
- zlib for zEDC [189](#), [191](#), [192](#), [194](#)



Product Number: 5655-ZOS

SA23-1377-70

