z/OS
2.5

# *Validated Boot for z/OS*

**Note**

Before using this information and the product it supports, read the information in “Notices” on page 143.

This edition applies to Version 2 Release 5 of z/OS® (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2023-05-15

# Contents

# Figures

# Tables

# About Validated Boot for z/OS

**Purpose of this information** This is a collection of the information needed to understand and use Validated Boot for z/OS. Some of the information contained in this collection also exists elsewhere in the z/OS library.

**Who should read this information** This information is intended for system programmers who are responsible for installing, configuring, and maintaining the Validated Boot for z/OS.

## Related information

To find the complete z/OS library, go to IBM Documentation (www.ibm.com/docs/en/zos).

## How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

**Important:** If your comment regards a technical question or problem, see instead "If you have a technical problem" on page xiii.

Submit your feedback by using the appropriate method for your type of comment or question:

**Feedback on z/OS function**
: If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

**Feedback on IBM® Documentation function**
: If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at ibmdocs@us.ibm.com.

**Feedback on the z/OS product documentation and content**
: If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

  To help us better process your submission, include the following information:

  - Your name, company/university/institution name, and email address
  - The section title of the specific information to which your comment relates
  - The comprehensive content collection title: Validated Boot for z/OS
  - The text of your comment.

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

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

## If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

# Chapter 1. Introduction to this content solution

This comprehensive content collection describes Validated Boot for z/OS and includes information from the following z/OS publications:

- *z/OS Introduction and Release Guide*
- *Device Support Facilities (ICKDSF) User's Guide and Reference*
- *z/OS MVS Initialization and Tuning Reference*
- *z/OS MVS Program Management: Advanced Facilities*
- *z/OS MVS Diagnosis: Tools and Service Aids*
- *z/OS Security Server RACF Security Administrator's Guide*
- *z/OS Security Server RACF Command Language Reference*
- *z/OS Security Server RACF Callable Services*
- *z/OS MVS System Commands*
- *z/OS MVS IPCS Commands*
- *z/OS MVS System Management Facilities (SMF)*
- *z/OS MVS System Messages, Vol 4 (CBD-DMO)*
- *z/OS MVS System Messages, Vol 6 (GOS-IEA)*
- *z/OS MVS System Messages, Vol 7 (IEB-IEE)*
- *z/OS MVS System Messages, Vol 9 (IGF-IWM)*
- *z/OS MVS System Codes*
- *z/OS MVS Data Areas Volume 1 (ABE - IAR)*
- *z/OS Security Server RACF Data Areas*

This document is part of the Validated Boot for z/OS (www.ibm.com/support/z-content-solutions/validated-boot-for-zos/) content solution.

For additional information about setting up Validated Boot for z/OS, see the white paper, z/OS Validated Boot (ibm.biz/zosValidatedBoot).

# Chapter 2. What is Validated Boot for z/OS?

Validated Boot for z/OS is a solution that uses digital signatures to provide an initial program load (IPL)-time check that validates that IPL data is intact, not tampered with, and originated from a trusted source. It also enables detection of unauthorized changes to software executables.

By using Validated Boot for z/OS, you have the ability to meet regulatory compliance standards, including National Information Assurance Partnership (NIAP) certification, that are required for certain secure software deployment scenarios. Additionally, you can detect both accidental and malicious IPL data changes earlier, thus reducing the impact of outages and stopping certain kinds of attacks.

Validated Boot for z/OS is implemented with updates to several elements, including MVS™, IPCS, Program Management (Binder), and RACF®.

Validated Boot for z/OS requires an IBM z16, with z/OS 2.5 or later for the target system.

# Chapter 3. Preparing your DASD

## LDIRTS parameter: specify which user-supplied IPL program record number to sign when doing List Directed IPL

| Parameter/ Abbreviations | Description |
|---|---|
| LDIRTS(*record number*) | Specifies the record number for the user-supplied IPL program that is to be signed and placed on the volume when doing List Directed IPL.<br>For *record number* specify a value between 1 to 10 to identify which record passed in with the IPLDD parameter is to be signed. |
| **Default:** | 4 |
| **Restrictions:** | Valid only in the MVS version of ICKDSF.<br>Valid only when the LDIPL parameter and the IPLDD parameter with the ABSFORMAT sub parameter are specified. |

## LDIPL parameter: write List Directed IPL records and a signed user-supplied IPL program record on the volume

| Parameter/ Abbreviations | Description |
|---|---|
| LDIPL(STANDARD\|DUMP) | |
| LDIPL | Write List Directed IPL records and a signed user-supplied IPL program record on the volume.. |
| STANDARD | Indicates that the user-supplied IPL program is standard IPL text. |
| DUMP | Indicates that the user-supplied IPL program is for stand-alone dump. |
| **Default:** | None. |
| **Restrictions:**<br>CMS SA | Valid only in the MVS version of ICKDSF<br>Valid only when the IPLDD parameter is specified.<br>The SYSIN option for passing in the user-supplied IPL program in the IPLDD parameter is not supported. |

# Chapter 4. Preparing your Software

## AUTOIPL parameter of DIAGxx

**AUTOIPL SADMP(sadmp info) MVS(mvs info) [NUCLABEL {ENABLE|DISABLE}(BLWRSTO2)]**

**(sadmp info)**

Either `(device,loadparm)` or `(NONE)`.

- For SADMP(device,loadparm), if the system is about to enter a wait state, SADMP is loaded from this volume with this load parameter.
- For SADMP(NONE), if the system is about to enter a wait state, SADMP is not loaded.

**(mvs info)**

Either `(device,loadparm,loadtype)` or `(LAST,loadtype)` or `(NONE)`

**loadtype**

Either CLEAR or NORMAL.

If `,loadtype` is not specified, it defaults to `,CLEAR`.

**Notes:**

1. SADMP specified with a device and load parameter and MVS specified with NONE causes the AutoIPL function to IPL SADMP only.
2. SADMP specified with NONE and MVS specified with a device and load parameter or with LAST causes the AutoIPL function to re-IPL MVS immediately, with no SADMP taken.
3. SADMP with a device and load parameter and MVS with a device and load parameter or with LAST causes SADMP to be IPLed, followed by MVS.
4. Any valid specification of AUTOIPL causes any prior AutoIPL information to be replaced.
5. SADMP (NONE) and MVS (NONE) both specified effectively deactivates the AutoIPL function.
6. The SADMP *loadparm* value can be specified so that SADMP executes without prompts to the operator. For information on coding the SADMP load parameter, see Procedure A: Initialize and run stand-alone dump in *z/OS MVS Diagnosis: Tools and Service Aids*.
7. The MVS with LAST parameters do not support HyperSwaps in which the IPL, IODF, and SADMP secondary volumes are not part of an alternate subchannel set (that is, the primary and secondary device pairs do not have identical device numbers and all reside in a single subchannel set). Therefore, if MVS with LAST was specified in that environment and a HyperSwap® swapped different device numberHyperSwap within subchannel set zero and no intervening IPL occurred, the MVS with LAST function will still attempt to use the original IPL, IODF, and SADMP volumes before the swap.
8. *device* is a 4-digit device number that can be prefixed by `*`. The asterisk prefix denotes that the device in the currently active subchannel set should be used. If an asterisk does not prefix the device number, the device in subchannel set 0 is used.

   This function is intended for HyperSwap environments that use alternate subchannel sets. Environments without HyperSwap or HyperSwap environments that do not use an alternate subchannel set will not benefit from using `*`.
9. Only in a HyperSwap environment that uses an alternate subchannel set will the MVS with LAST parameters ensure that AUTOIPL uses the device number from the appropriate subchannel set in the event of a HyperSwap.
10. AUTOIPL, when requested to IPL an MVS system, will, by default, specify the `CLEAR` option that can lead to an elongated response time for the function. The request can be made to perform better by specifying the `CLEAR` suboption for AUTOIPL. The result of requesting this option is that the AUTOIPL initiated IPL will be done using the `NORMAL` option.

**Note:** This processing can result in the unavailability of reconfigurable storage when the IPL is done so the specification of this option should only be used when necessary.

11. An IPL of MVS or SADMP that is initiated via AUTOIPL will be the same type of IPL (CCW or List-Directed) as the last IPL that was initiated by an operator. If that IPL was List-Directed, the Secure or not Secure setting for AUTOIPL will also be the same as the last IPL that was initiated by an operator.

For more information about AutoIPL, see Using the automatic IPL function in *z/OS MVS Planning: Operations*.

## CLPA parameter of IEASYSxx

**CLPA**

(See also the MLPA parameter for temporary additions to the LPA, and the CVIO parameter for the deletion of VIO data sets.) This parameter causes NIP to load the LPA with all modules contained in the LPALST concatenation. Modules that are listed in the specified LPA pack list member (IEAPAKxx) are packed together, preferably in one-page groups. (See description of IEAPAKxx.) Modules not in the pack list are loaded in size order, large modules first, then smaller modules to fill unused space.

PLPA pages are written to auxiliary storage. Only one set of PLPA pages can exist in paging space. Modules in the LPALST concatenation must be reenterable and refreshable because the system uses the processor's page protection facility, which enforces read-only access to each PLPA page.

CLPA should be specified after the installation has modified a data set in the LPALST concatenation and wants to reload the PLPA with new or changed modules.

**Note:** CLPA also implies CVIO, so that VIO data set pages on local page data sets are automatically purged. (See the description of CVIO for further information.)

The CLPA parameter is not needed at the first IPL. NIP detects the cold start condition internally, noting that the PLPA has not been loaded.

**Note:** CLPA is always enforced for Validated Boot for z/OS. No message is issued about this being done.

If CLPA is not specified, NIP tries to find a usable PLPA in the existing page data sets. If NIP is successful, a quick start or a warm start occurs, and the auxiliary storage manager (ASM) obtains the records that specify where the PLPA pages reside on auxiliary storage. It then reestablishes the previous set of PLPA pages. The old PLPA can be reused for any number of system initializations, if CLPA is not specified. However, page data sets that contain the last used set of PLPA pages must be mounted. If they are not, the operator is asked to mount them. If the operator bypasses mounting, ASM initialization requests a different page data set and forces a *cold* start. NIP then reestablishes the PLPA as it does when CLPA is specified. In this cold start, both the previously established PLPA and existing VIO data set pages are logically deleted from paging space.

The fixed LPA and the modified LPA, however, are not automatically reused in a quick start or a warm start. They must be respecified. Existing VIO data set pages on local page data sets are retained in a warm start, unless the CVIO or CLPA parameter is forced. Such pages are not retained in a quick start or a cold start. (See the description of the CVIO parameter.)

If CLPA is specified and a set of PLPA pages already exists on a paging data set, NIP frees the existing PLPA and updates the appropriate records to reflect the new PLPA pages on auxiliary storage. NIP loads the LPA from the LPALST concatenation, as previously described.

IBM suggests that you have an IEASYSxx member that does not specify CLPA. This permits you to load the initial program without rebuilding the PLPA if it is not possible to access your LPA data sets.

**Value range:** Not applicable

**Default:** Not applicable

**Associated parmlib member:** None

# Chapter 5. Utilities

## IEWSIGN: Sign, unsign, and report load modules

IEWSIGN, the signing utility on z/OS, provides the following functions:

- Sign a load module in a partitioned data set (PDS). The utility adds a signature to the load module and marks it as signed.
- Unsign a load module in a PDS. The utility removes the signature from the load module and marks it as un-signed.
- Report on a load module in a PDS. The utility reports signing-related information, such as the time the load module was signed, the signing algorithm, and the certificate fingerprint.

IEWSIGN must be invoked by either JCL that uses EXEC PGM=xxx, call, or LINK.

It resides in SYS1.SIEAMIGE.

### Usage notes

The following list contains usage notes for the signed load module:

- To copy a signed load module from one PDS to another PDS, IEBCOPY with the control statement COPY should be used. Do not use IEBCOPY with the control statement COPYMOD, which causes the signature in the destination load module to become invalid. Do not use the z/OS UNIX command **cp**, which causes the destination load module to become unsigned.
- Copying a signed load module from a PDS to a PDSE causes the destination program to become unsigned.
- Relinking a signed load module causes the resultant program to become unsigned.
- Reprocessing a signed load module with the binder API SAVEW causes the resultant program to become unsigned.
- Renaming a signed load module causes its signature to become invalid.

### Parameters of IEWSIGN

Parameters of IEWSIGN have the following rules:

- For JCL, parameters can be provided by `PARM='..'` or by `PARMDD=xx` with the data within the `xx` DD.
- For call or LINK, parameters consist of a halfword length followed by the parameter string, as the first parameter pointed to by the parameter list located by the PARAM keyword (and thus located by register 1 on entry to the target routine).
- Parameters are of the form keyword=value, separated from one another by a single comma.
- Parameters can be in any order.
- Optional parameters need not be provided.

The following table lists the parameters of IEWSIGN along with their descriptions:

*Table 1. Parameters of IEWSIGN*

| Parameter name | Description |
|---|---|
| Action | Specifies the action to be run. There is no default value. It must be specified explicitly.<br>Its eligible values are as follows:<br>• Sign<br>• Unsign<br>• Report |
| State | For Action=Sign and Action=Report, it determines which name-matched members will be processed. For Action=Unsign, this parameter is ignored, since only signed load modules are processed.<br>Its eligible values are as follows:<br>**Unsigned**<br>Only unsigned name-matched members will be processed.<br>**Signed**<br>Only signed name-matched members will be processed.<br>**All**<br>All name-matched members will be processed. This is the default value.<br>When Action=Sign and State is Signed or All, existing signed modules will be re-signed. |
| RC4LIM/ RC8LIM | This integer specifies the limits for return code 4 / return code 8.<br>Valid range is from 1 to 2147483647.<br>If any limit is reached, this utility will terminate.<br>The default value of RC4LIM is 2147483647.<br>The default value of RC8LIM is 1 for ACTION=SIGN and ACTION=UNSIGN, and 2147483647 for ACTION=REPORT.<br>This utility will terminate immediately if a return code 12 or higher occurs. |
| Verbose | Specifies the content that is provided in SYSPRINT.<br>**Yes**<br>Specifying this option provides more detailed content.<br>**No**<br>Specifying this option provides less detailed content.<br>. |

*Table 1. Parameters of IEWSIGN (continued)*

| Parameter name | Description |
|---|---|
| ReportLevel | Specifies the level of checking/printed information when ACTION=REPORT.<br>**1**<br>Only check whether a load module is signed. This is the default value.<br>**2**<br>Includes all reports for ReportLevel=1, and the text size, link time, signing time, binder version, hash algorithm, signing algorithm, certificate fingerprint.<br>**3**<br>Includes all reports for ReportLevel=2, and whether the hash in signature is valid. |

The utility's processing of all parameters is not case-sensitive.

Parameters in the PARM string are separated by commas.

## Data definitions (DD) of IEWSIGN

The following table lists the IEWSIGN data definitions and their descriptions:

*Table 2. IEWSIGN data definitions*

| DD name | Description |
|---|---|
| SYSPRINT | This is a required DD. It is valid for all Action values.<br>This data set contains the IEWSIGN processing messages or reports. The specification for the data set is as follows:<br>`//SYSPRINT DD SYSOUT=*`<br>If DCB attributes are expected, the following recommendation is provided:<br>`LRECL   RECFM`<br>`121     FBA` |
| INFILE | This is a required DD. It is valid for all Action values.<br>This DD specifies the source PDS library where load modules are read from.<br>Concatenation of multiple data sets is not allowed. IEWSIGN will fail if the data set is not a PDS. |

*Table 2. IEWSIGN data definitions (continued)*

| DD name | Description |
|---|---|
| OUTFILE | When Action=Sign or Action=Unsign, this DD is required. When Action=Report, this DD is not required.<br><br>This DD specifies the destination PDS library where load modules are written to.<br><br>Concatenation of multiple data sets is not allowed. The signing utility will fail if the data set is not a PDS.<br><br>In-place signing is supported by specifying the same DSN in the INFILE and OUTFILE.<br><br>If the blocksize of OUTFILE is not 0, it must be equal to or larger than the blocksize of INFILE. Additionally, it must be equal to or larger than 1024. The maximum value can be 32760.<br><br>If the blocksize of OUTFILE is 0, the IEWSIGN utility will set it to the blocksize of the INFILE data set. |
| INCLUDE | This is an optional DD. It is valid for all Action values.<br><br>This DD specifies members to be included from INFILE. If this DD is unspecified, all members in INFILE will be included.<br><br>The standard specification for this data set is as follows:<br><br>`//INCLUDE DD *`<br><br>If a regular data set is provided, it needs to be RECFM=FB and LRECL=80. |
| EXCLUDE | This is an optional DD. It is valid for all Action values.<br><br>This DD specifies members to be excluded from INFILE. If this DD is unspecified, no members in INFILE will be excluded.<br><br>The standard specification for this data set is as follows:<br><br>`//EXCLUDE DD *`<br><br>If a regular data set is provided, it needs to be RECFM=FB and LRECL=80. |

Both INFILE and OUTFILE must be a PDS whose record format is U. In addition, the IEWSIGN utility will not process a member in INFILE if any of the following are true:

- It is not a load module.
- It is a load module with an overlay attribute.
- It is a load module without TEXT.

## Rules for INCLUDE and EXCLUDE

Both INCLUDE and EXCLUDE specify one or multiple names, which are used to match primary members and aliases in INFILE. The following rules apply:

- A name must obey PDS member naming rules.
- Multiple names are not allowed on a single line.
- A name must not be continued to the next line.
- Blank characters are allowed before and after a name.
- Comments are supported with the following conditions:

- A line will be treated as a comment if the first nonblank character is “#”.
- A blank line, where all characters are space (its hex value is 0x40), is acceptable.

- Wildcards are supported. Both “*” and “?” can be used in a name. . “*” matches “0 or more characters” and “?” matches “exactly one character”. The following are some examples:
  - `*`
  - `ABC*`
  - `ABC??DEF`
  - `X*YYY`
- Only the first 72 characters in a line are parsed.

IEWSIGN uses the following steps to determine which members should be processed (for example, signed, unsigned or reported).

*Table 3. Steps for IEWSIGN member processing*

| **Step number** | **Step instructions** |
|---|---|
| Step 1 | Retrieve a list of names of all primary members of INFILE whose signing state is the one required by parameter STATE. |
| Step 2 | If INCLUDE is not specified, skip this step.<br>If INCLUDE is specified, remove a primary member from this list if both of the following are true:<br>1. The name of the primary member does not match any name specified in INCLUDE, and<br>2. None of its alias names matches any name specified in INCLUDE. |
| Step 3 | If EXCLUDE is not specified, skip this step.<br>If EXCLUDE is specified, remove a primary member from this list if either of the following are true:<br>1. The name of the primary member matches any name specified in EXCLUDE, or<br>2. One of its alias names matches any name specified in EXCLUDE. |
| Step 4 | All primary members remaining in the list are processed. |

**Note:** For Action=Sign, when a primary member is being signed, in addition to adding load module signing records, IEWSIGN will update the directory entries of the primary member and all its aliases. Therefore, when a primary member becomes signed, all its aliases also become signed. For Action=Unsign, analogous processing is performed.

A detailed example of INCLUDE/EXCLUDE is provided in next section as Example 1.

## Examples

1. The following provides an example of INCLUDE/EXCLUDE processing. The conditions for this example are as such:
   - A PDS has 4 primary members: M1, M2, M3 and M4.
     - M1 has one alias A11;
     - M2 has two aliases A21 and A22;
     - M3 and M4 have no alias.
     - M1, M2 and M3 are unsigned.
     - M4 is signed.

- IEWSIGN is called with parameter Action=Sign,State=Unsigned.
- INCLUDE has two lines

  ```
  M1
  A21
  ```

- EXCLUDE has one line:

  ```
  A11
  ```

Results of the processing are as follows:

**Step 1 results**
: IEWSIGN receives a name list of all unsigned primary members of INFILE. At the end of step 1, M1, M2, M3 are in the list.

**Step 2 results**
: M1 is kept, as its name matches a filter specified at the first line of INCLUDE. M2 is kept, as its alias A21 matches a filter specified on the second line of INCLUDE. M3 is removed, as it and all its aliases don't match any filters in INCLUDE. At the end of step 2, M1,M2 are in the list.

**Step 3 results**
: M1 is removed, as its alias A11 matches a filter specified at the first line of EXCLUDE. M2 is kept, as it and all its aliases do not match any filters in EXCLUDE. At the end of step 3, M2 is in the list.

**Step 4 results**
: IEWSIGN begins to sign M2. In this step, a signature is added to the load module records of M2. In addition, the directory entries of M2, A21 and A22 are all updated together. Therefore, M2, A21 and A22 all become signed.

2. The following is a JCL example that signs all load modules in place. In this example, both INFILE and OUTFILE use the same DSN. As a result, signed modules will be saved into its original PDS.

   ```
   //SIGN EXEC PGM=IEWSIGN,PARM='Action=Sign'
   //STEPLIB  DD DISP=SHR,DSN=SYS1.SIEAMIGE
   //SYSPRINT DD SYSOUT=*
   //INFILE   DD DSN=SYS1.LPALIB,DISP=SHR
   //OUTFILE  DD DSN=SYS1.LPALIB,DISP=SHR
   ```

   **Note:** In-place signing requires the OUTFILE data set to have room for both the original modules and for the updated versions of those modules, since the storage space of the original modules cannot be reused during the processing.

3. The following is a JCL example that signs all unsigned load modules specified by INCLUDE/EXCLUDE:

   ```
   //SIGN EXEC PGM=IEWSIGN,PARM='Action=Sign,State=Unsigned'
   //STEPLIB  DD DISP=SHR,DSN=SYS1.SIEAMIGE
   //SYSPRINT DD SYSOUT=*
   //INFILE    DD DSN=SYS1.LINKLIB,DISP=SHR
   //OUTFILE   DD DSN=SYS1.SIGN.LIB,DISP=(NEW,PASS),DSNTYPE=PDS,
   //             SPACE=(CYL,(500,500,5)),UNIT=3390
   //INCLUDE   DD *
   # this is a comment line
   AMBLIST
   IEHMVE*
   //EXCLUDE    DD *
   IEHMVE2
   ```

   For example, if INFILE has the following six unsigned members: AMBLIST, AMBLIST2, IEHMVE1, IEHMVE2, IEHMVE3, IEHMVE4, then the matched members are as follows: AMBLIST, IEHMVE1, IEHMVE3, IEHMVE4.

4. The following is a JCL example that unsigns all signed load modules in-place:

   ```
   //UNSIGN EXEC PGM=IEWSIGN,PARM='Action=Unsign'
   //STEPLIB  DD DISP=SHR,DSN=SYS1.SIEAMIGE
   //SYSPRINT DD SYSOUT=*
   ```

```
//INFILE    DD DSN=SYS1.LINKLIB,DISP=SHR
//OUTFILE   DD DSN=SYS1.LINKLIB,DISP=SHR
```

5. The following is a JCL example that reports all load modules:

```
//REPORT EXEC PGM=IEWSIGN,PARM='Action=Report'
//STEPLIB  DD DISP=SHR,DSN=SYS1.SIEAMIGE
//SYSPRINT DD SYSOUT=*
//INFILE   DD DSN=SYS1.LINKLIB,DISP=SHR
```

6. The following is a JCL example that reports load modules specified by INCLUDE or EXCLUDE:

```
//REPORT EXEC PGM=IEWSIGN,PARM='Action=Report,Verbose=YES'
//STEPLIB  DD DISP=SHR,DSN=SYS1.SIEAMIGE
//SYSPRINT DD SYSOUT=*
//INFILE   DD DSN=SYS1.LINKLIB,DISP=SHR
//INCLUDE    DD *
AMBLIST
IEHMVE*
//EXCLUDE    DD *
IEHMVE2
```

7. The following two examples show the use of SYSPRINT with Action=Sign. To better understand these examples, know that the contents of SYSPRINT consists of two parts:

   - A selection part, which indicates which members will be selected by INCLUDE or EXCLUDE from INFILE. This part is the same for all ACTION values.
   - A processing part, which displays the results of actions such as signing, unsigning, and reporting. This part is different for each ACTION value.

   a. The following is an example of the selection part for Action=Sign:

      **Note:** In the selection part of the example, report lines in italic type are printed only when Verbose=Yes. (in the example, the italicized section starts with the line "`Member/Alias(es) in INFILE with STATE=UNSIGNED`" and goes to the end of the example)

```
Invocation parameters: ACTION=SIGN,STATE=UNSIGNED,VERBOSE=YES
Execution  Parameters:
ACTION=SIGN,STATE=UNSIGNED,VERBOSE=YES,RC4LIM=2147483647,RC8LIM=1,REPORTLEVEL=1

DD        Data Set Name                               Volume        Block Size
INFILE    SYS1.LPALIB                                 BPX111        32760
OUTFILE   SYS1.LPALIB                                 BPX222        32760

INFILE
summary:

          Unsigned primary members
1
          Unsigned aliases
2
          Signed   primary members
0
          Signed   aliases
0
          Non-LM   members
1
          Overlay       LM
0
          Zero-TEXT     LM              0

Member/Alias(es) in INFILE with STATE=UNSIGNED
Member      Alias(es)
ASM         AL1      AL2

Including members specified in INCLUDE ...
<NONE>

Member/Alias(es) selected after INCLUDing
Member      Alias(es)
ASM         AL1      AL2

Excluding members specified in EXCLUDE ...
<NONE>

Member/Alias(es) selected after EXCLUDing
```

```
Member      Alias(es)
ASM         AL1      AL2
```

Each line in SYSPRINT is one of the two types:

**Report line**
This line has no message ID. The text is provided for informational purposes.

**Message line**
This line has a message ID. These lines are only for warning and error messages. These messages are documented in *z/OS MVS System Messages, Vol 8 (IEF-IGD)*.

b. The following is an example of the processing part for Action=Sign:

```
Signing results:
ASM      Successful

OUTFILE
summary:

          Unsigned primary members
0
          Unsigned aliases
0
          Signed   primary members
1
          Signed   aliases
2
          Non-LM   members
0
          Overlay       LM
0
          Zero-TEXT     LM              0

Processing summary of selected primary members:
          Selected                      1
          Processed                     1
          Processed successfully        1
          Processed with error          0

IEW6007W SYSCATLG in INFILE is excluded. It is not a load module.

Task completed with RC=4.
```

8. The following example shows the result of SYSPRINT for Action=Report,Verbose=No,ReporLevel=1:

**Note:** The selection part is not provided here, as it is the same as in Example 7a.

```
Name      Signed
BPXMIDMX  No
M1        Yes
M2        Yes
M3        Yes
M41ST     Yes
M4111     Yes
M4112     Yes
YM1       Yes
YM2       Yes
ZM1       Yes

Processing summary of selected primary members:
          Selected                      10
          Processed                     10
          Processed successfully        10
          Processed with error          0

IEW6007W SYSCATLG in INFILE is excluded. It is not a load module.

IEWSIGN exits with return code 4.
```

9. The following example shows the result of SYSPRINT for Action=Report,Verbose=No,ReporLevel=3:

**Note:** The selection part is not provided here, as it is the same as in Example 7a. Also, since the report formats for ReporLevel=2 and ReporLevel=3 are identical, only an example for ReporLevel=3 is provided.

```
Name      Size     Link date/time      Rel  Signed ErrorID Sign date/time      ALG  Cert-
Index
BPXMIDMX  00002218 2021-08-02 14:03:50 0205 No
M1        00000008 2022-09-14 08:29:45 0205 Yes    ERR01
M2        00000008 2022-09-14 08:29:45 0205 Yes    ERR01
M3        00000008 2022-09-14 08:29:45 0205 Yes    ERR01
M41ST     00000008 2022-09-14 08:29:45 0205 Yes    ERR01
M4111     00000008 2022-09-14 08:29:45 0205 Yes    ERR01
M4112     00000008 2022-09-14 08:29:45 0205 Yes    ERR01
YM1       00000008 2022-09-30 22:19:28 0205 Yes    ERR09
YM2       00000008 2022-10-02 21:19:09 0205 Yes    ERR09
ZM1       00000008 2022-10-13 15:11:32 0205 Yes           2022-10-13 15:11:32 0202 INDEX001


ErrorID   Number    Error explanations
ERR01          6    Signing records lost or incomplete.
ERR09          2    Signature length is invalid.

Algorithm ID        Hash algorithm                Sign algorithm
0202                SHA2-512                      ECDSA-P521

Certificate summary:
Cert-Index:         INDEX001
Subject KeyID:      21CC95D0 8A12F9FE 5AA01598 430EF6A0 8D58DFDE
Cert Fingerprint:   0CED78C4 802B2B9A 3D190F75 8A79F005 87EF2294 69D680A0 C63B2FEE D3120D83

Processing summary of selected primary members:
          Selected                     10
          Processed                    10
          Processed successfully       2
          Processed with error         8

IEW6007W SYSCATLG in INFILE is excluded. It is not a load module.
IEW6027E 8 reported load modules have errors.

IEWSIGN exits with return code 8.
```

The following is an explanation of the contents in this example.

The terms in the first table of the example are described as follows:

**Size**
: The size of the load module's code in bytes.

**Link date/time**
: When the load module was built. It will be blank if the link time is unavailable.

**Rel**
: The binder version that linked the load module.

**Signed**
: Indicates whether the module is signed.

**ErrorID**
: Indicates whether there is a signature error. Blank means no error. Use this error ID to find an error description.

**Sign date/time**
: Indicates when the load module was signed.

**ALG**
: The algorithm of hash and sign. Use this ID to find an algorithm description.

**Cert-Index**
: An index assigned by IEWSIGN. Use this index to find a certificate displayed in “Certificate summary”, which signed this load module.

The table with heading “ErrorID Number Error explanations” provides the number of each error, along with a brief explanation for the error.

The following table provides a brief explanation and a detailed explanation for each ErrorID:

*Table 4. ErrorID explanations*

| ErrorID | Brief explanation | Detailed explanation |
|---|---|---|
| ERR01 | Signing records lost or incomplete. | The sign flag in the directory is on, but the signing records are lost or incomplete. This error is usually caused by some tools that are able to read and write regular records in load modules, but will discard the new signing records. For example, IEBCOPY with COPYMOD will cause this error. |
| ERR02 | Subtype of signing record is invalid. | A field in the signing record is invalid. The signing record has been modified by unknown tools. |
| ERR03 | Version of signing record is invalid. | A field in the signing record is invalid. The signing record has been modified by unknown tools. |
| ERR04 | Flags of the signing record are invalid. | A field in the signing record is invalid. The signing record has been modified by unknown tools. |
| ERR05 | Length of signing record is invalid. | A field in the signing record is invalid. The signing record has been modified by unknown tools. |
| ERR06 | Reserved field of signing record is invalid. | A field in the signing record is invalid. The signing record has been modified by unknown tools. |
| ERR07 | Signature type is invalid. | A field in the signing record is invalid. The signing record has been modified by unknown tools. |
| ERR08 | Signature version is invalid. | A field in the signing record is invalid. The signing record has been modified by unknown tools. |
| ERR09 | Signature length is invalid. | A field in the signing record is invalid. The signing record has been modified by unknown tools. |
| ERR10 | Signature reserved bytes are invalid. | A field in the signing record is invalid. The signing record has been modified by unknown tools. |

*Table 4. ErrorID explanations (continued)*

| ErrorID | Brief explanation | Detailed explanation |
|---|---|---|
| ERR11 | Signature algorithm is invalid. | A field in the signing record is invalid. The signing record has been modified by unknown tools. |
| ERR12 | Signature hash is invalid. Load module has been modified. | The hash calculated at the reporting time does not match the hash calculated at the signing time. This error is usually caused by tools that are able to modify load modules. For example, SPZAP will cause this error. |
| ERR13 | Directory entry error. Check the error message. | The directory entry of this module at the time of reporting is different from the directory entry of this module at the time of signing. More information is provided by one or more of the following error messages: IEW6022E, IEW6023E or IEW6024E |

The table with the heading "Algorithm ID Hash algorithm Sign algorithm" provides the names of the algorithms used.

"Certificate summary" lists all certificates used to sign the reported load modules.

## Return codes for the IEWSIGN utility

The following table lists the return codes, conditions of each return code, and the corresponding message that is issued for each condition.

*Table 5. Return codes for the IEWSIGN utility*

| Return code (decimal) | Conditions of the return code |
|---|---|
| 0 | 1. Successful completion. No error or warning messages are issued. |
| 4 | 1. One or more non-load modules are found in INFILE. Message number: IEW6007W<br>2. One or more overlay load modules are found in INFILE. Message number: IEW6008W<br>3. One or more zero-text load modules are found in INFILE. Message number: IEW6009W<br>4. The primary member has been renamed. Message number: IEW6011W |
| 8 | 1. One or more directory entry errors have been found in INFILE. Message number: IEW6010E<br>2. When Action=Report, an invalid signing record has been found. Message number: IEW6022E, IEW6023E, IEW6024E, IEW6025E, IEW6026E, IEW6027E<br>3. RC4LIM or RC8LIM reached. Message number: IEW6014E, IEW6015E<br>4. A load module has no CESD record or TEXT record, Message number: IEW6031E |

Table 5. Return codes for the IEWSIGN utility (continued)

| Return code (decimal) | Conditions of the return code |
|---|---|
| 12 | 1. Invalid parameters. Message number: IEW6001S, IEW6002S. IEW6003S, IEW6028S<br>2. Necessary DD missed. Message number: IEW6004S<br>3. Incorrect data set attributes. Message number: IEW6005S, IEW6006S, IEW6017S, IEW6018S, IEW6034S, IEW6035S<br>4. Syntax error in INCLUDE or EXCLUDE. Message number: IEW6012S<br>5. No load modules have been selected for processing. Message number: IEW6013S<br>6. Output error of OUTFILE. Message number: IEW6019S, IEW6020S<br>7. RACF has not been configured correctly to sign a load module. Message number: IEW6016S, IEW6033S<br>8. IEWSIGN cannot allocate enough memory. Message number: IEW6021S<br>9. Invalid blocksize of OUTFILE. Message number: IEW6029S, IEW6030S<br>10. Language Environment callable service fails. Message number: IEW6032S |

Programming Interface Information

## Invoking the signing utility (IEWSIGN) from another program

In most cases, IEWSIGN is invoked by JCL. However, it can be invovked by another program. Furthermore, the program may specify alternative DDs to replace the default DDs (INFILE, OUTFILE, SYSPRINT, INCLUDE, and EXCLUDE) used by IEWSIGN.

**Restriction:**

- IEWSIGN must be invoked in AMODE 31.
- IEWSIGN must be invoked in user key, problem state, and the job step key must also be the user key.
- The calling program cannot be a Language Environment program.

To invoke IEWSIGN, the program must set register 1 (R1) to point to a pointer array.

The following two cases are supported:

**Case 1**

The pointer array has only one pointer.

The highest bit of this pointer must be ON. This points to an area that consists of a 2-byte length field followed by a parameter string of the length identified by the length field. The length of this string cannot be more than 1024.

**Case 2**

The pointer array has two pointers.

The highest bit of the first pointer must be OFF. This points to an area that consists of a 2-byte length field followed by a parameter string identified by the length field. The length of this string cannot be more than 1024.

The highest bit of the second pointer must be ON. This pointer points to the following structure:

Table 6. Structure of alternate DD list

| Offset | Length (in bytes) | Description |
|---|---|---|
| 0 | 2 | Length of the remainder of the alternate DD list, excluding this halfword. It must be one of the following values: 0,8,16,24,32 or 40. |

*Table 6. Structure of alternate DD list (continued)*

| **Offset** | **Length (in bytes)** | **Description** |
|---|---|---|
| 2 | 8 | Alternate DD for INFILE |
| 10 | 8 | Alternate DD for OUTFILE |
| 18 | 8 | Alternate DD for SYSPRINT |
| 26 | 8 | Alternate DD for INCLUDE |
| 34 | 8 | Alternate DD for EXCLUDE |

Each DD name that is shorter than eight characters must be padded with blanks. Each entry that you do not want to override must be binary zeroes, not blanks.

The following is an example assembler program. The program calls IEWSIGN with these parameters: ACTION=REPORT,REPORTLEVEL=3 . It also specifies two alternate DDs: “SYS0001” for “INFILE”, and “PRT0001” for “SYSPRINT”.

```
//*  A  program invokes IEWSIGN with alternate DDs
//ASM      EXEC   PGM=ASMA90,PARM='GOFF,LIST(133)'
//SYSUT1   DD     UNIT=SYSDA,SPACE=(CYL,(5,2))
//SYSLIB   DD     DSN=SYS1.MACLIB,DISP=SHR
//SYSPRINT DD     SYSOUT=*
//SYSLIN   DD  DSN=&&OBJ1,DISP=(NEW,PASS),SPACE=(TRK,(4,1)),UNIT=SYSDA
//SYSIN    DD     *
CSECT1    CSECT
CSECT1    AMODE 31
CSECT1    RMODE 31
          SYSSTATE ARCHLVL=3
          STM   R14,R12,12(R13)
          BALR  R12,0
          USING *,R12
          STORAGE OBTAIN,LENGTH=DYNSIZE,ADDR=(R11)
          USING  DYNAREA,R11
          LA    R2,SAVEAREA
          ST    R2,8(,R13)
          ST    R13,SAVEAREA+4
          LR    R13,R2
**********************  BEGIN LOGIC  *********************************
          LINK EP=IEWSIGN,PARAM=(PARM1,PARM2+X'80000000')
          LR    R10,R15    * save return code from IEWSIGN
**********************  END LOGIC    *********************************
RETURN    EQU   *
          L     R13,SAVEAREA+4
          STORAGE RELEASE,LENGTH=DYNSIZE,ADDR=(R11)
          L     R14,12(,R13)
          LR    R15,R10    * restore return code from IEWSIGN
          LM    R2,R12,28(R13)
          BR    R14
**********************  DATA AREAS   *********************************
PARM1     DC    H'27',C'ACTION=REPORT,REPORTLEVEL=3'
PARM2     DC    H'40'
          DC    CL8'SYS0001 '           INFILE   changed to SYS0001
          DC    XL8'0000000000000000'   OUTFILE  unchanged
          DC    CL8'PRT0001 '           SYSPRINT changed to PRT0001
          DC    XL8'0000000000000000'   INCLUDE  unchanged
          DC    XL8'0000000000000000'   EXCLUDE  unchanged
DYNAREA   DSECT
SAVEAREA  DS   18F
DYNSIZE   EQU  *-DYNAREA
          YREGS ,
          END ,
//*******************************************************************
//LINK    EXEC PGM=IEWBLINK,PARM='RMODE=31'
//OBJ      DD DSN=&&OBJ1,DISP=(OLD,DELETE)
//SYSPRINT DD SYSOUT=*
//SYSLMOD  DD DSN=&&TOOLS,DISP=(NEW,PASS),DSNTYPE=LIBRARY,
//             SPACE=(TRK,(5,5,5)),UNIT=3390
//SYSLIN   DD *
   ENTRY   CSECT1
   INCLUDE OBJ
   NAME    CALLER(R)
//*******************************************************************
//*  A program that invokes IEWSIGN
```

```
//RUN      EXEC PGM=CALLER
//STEPLIB  DD DSN=&&TOOLS,DISP=(OLD,PASS)
//         DD DSN=SYS1.SIEAMIGE,DISP=SHR
//SYS0001  DD DSN=yourDSN,DISP=SHR
//PRT0001   DD SYSOUT=*
```

**Note:** Replace your DSN in the last JCL step with an actual data set name that you want to use for ACTION=REPORT.

End Programming Interface Information

## IEW messages

**IEW2950I SIGNING RECORDS HAVE BEEN DISCARDED.**

### Explanation

The input load module is a signed load module. It will become unsigned after the binder processing.

### System action

Processing continues.

**User response:**
None.

### Source

Binder

### Module

IEWBXCRW

**IEW6001S The length of the parameter string is *dddd*. It cannot be more than 1024.**

### Explanation

The length of the parameter string must be less than or equal to 1024. For JCL the parameter string is what is specified via PARM= or the PARMDD DD. For call and LINK, the parameter string is what is specified via the PARAM keyword.

### User response

Correct the parameter string.

### Source

Signing utility

**IEW6002S Invalid parameter: '*xxxxxxxx*'.**

### Explanation

The parameter string is split into multiple parts by a delimiter comma. Each part must be in the form of `keyString=valueString`. This message is issued for a part if:

- "=" is not found, or
- `keyString` is unrecognized, or
- `valueString` is unrecognized

### User response

Correct the parameters.

### Source

Signing utility

**IEW6003S Parameter ACTION not specified.**

### Explanation

Parameter ACTION must be specified.

### User response

Add the parameter ACTION.

### Source

Signing utility

**IEW6004S Cannot open DD *ddName*.**

### Explanation

*ddName* (such as INFILE, OUTFILE) is not defined, or the user does not have access to read or write.

### User response

Provide a definition for *ddName*.

### Source

Signing utility

**IEW6005S** **Cannot close DD *ddName*.**

**Explanation:**
The DFSMS service CLOSE cannot close the DCB opened for this ddName.

### User response

Attempt again.

### Source

Signing utility

**IEW6006S** **DD *ddName* does not specify a RECFM=U PDS.**

### Explanation

*ddName* (such as INFILE or OUTFILE) must be a PDS with RECFM=U.

### User response

Check the *ddName*.

### Source

Signing utility

**IEW6007W** **Member *member* in DD INFILE is excluded. It is not a load module.**

### Explanation

This module is not a load module.

### User response

Not applicable.

### Source

Signing utility

**IEW6008W** **Member *member* in DD INFILE is excluded. It has the overlay attribute.**

### Explanation

This module is an overlay load module, and cannot be signed.

### User response

Not applicable.

### Source

Signing utility

**IEW6009W** **Member *member* in DD INFILE is excluded. It has no text.**

### Explanation

This module has no text and cannot be signed.

### User response

Not applicable.

### Source

Signing utility

**IEW6010E** **Alias *alias* is an orphan in DD INFILE.**

### Explanation

There are two possibilities for this error:

1. Its primary member has been deleted.
2. Its primary member has been linked again without specifying this alias. In addition, ISPF will indicate that the TTR of this alias is different from the TTR of its primary member.

### User response

Correct this directory error before signing. If this alias is no longer useful, delete it. If you must sign this alias, convert this alias to a primary member before you sign it.

You may use the following commands in z/OS UNIX to convert an alias to a primary member. Suppose your data set name is `USER1.PGM.PDS`, and the alias name is A1:

```
cp -X "//'USER1.PGM.PDS(A1)'" A1
cp -X A1 "//'USER1.PGM.PDS(A1)'"
```

The second command may issue a warning message IEW2627I, which can be ignored. After these two commands, alias A1 becomes a primary member.

### Source

Signing utility

**IEW6011W** **The primary member of alias *alias* in DD INFILE has been renamed from *oldName* to *newName*.**

### Explanation

The primary member has been renamed.

For example, a primary member M1 had an alias A1, They had the same TTR attribute in their directory entries, which were set by the binder at link time. Later, M1 was renamed to M2.

**Note:** After the renaming, the following are true:

- The primary member name in the directory entry for A1 is still M1.
- TTR of M2 is unchanged.

When the IEWSIGN utility is analyzing directory entries of INFILE, it finds that A1 is an alias of M2 as they have the same TTR. However, the primary member name in the directory entry for A1 is M1. Therefore, the IEWSIGN utility issues this message.

For ACTION=SIGN and ACTION=UNSIGN, in OUTFILE, the alias will link to the new name rather than the old name.

### User response

Not applicable.

### Source

Signing utility

**IEW6012S** **Multiple names in a line of DD *ddname* are not allowed. Line='*xxxx*'.**

### Explanation

In INCLUDE or EXCLUDE, a line cannot have two names.

For example, the following line is not allowed:

```
IEW*   IEB*
```

The two names must be placed into two separate lines.

### User response

Place the names on separate lines to correct the problem.

### Source

Signing utility

**IEW6013S** **No load modules to be processed.**

### Explanation

No load module has been selected for the specified action. The IEWSIGN utility selects load modules in the following three steps:

1. Select load modules from INFILE by parameter STATE.

   **Note:** For ACTION=UNSIGN, only unsigned load modules are selected.
2. Selected load module in previous step is then filtered by INCLUDE.
3. Selected load module in previous step is then filtered by EXCLUDE.

Refer to IEWSIGN: Sign, unsign, and report load modules in *z/OS MVS Diagnosis: Tools and Service Aids* for more instructions on the IEWSIGN utility.

### User response

Define INFILE to a PDS with load modules, and set INCLUDE, EXCLUDE, or both, to select one or multiple load modules.

### Source

Signing utility

**IEW6014E** **RC4LIM (*number*) reached.**

### Explanation

This number of warnings reached the RC4LIM.

### User response

Resolve the warnings or set RC4LIM to a larger value.

### Source

Signing utility

**IEW6015E** **RC8LIM (*number*) reached.**

### Explanation

This number of errors reached the RC8LIM.

### User response

Resolve the errors or set RC8LIM to a larger value.

### Source

Signing utility

**IEW6016S** **RACF R_PgmSignVer failed. Function Code=*funcName*, SAF**

**return code=*xx*, RACF return code=*yy*, RACF reason code=*zz*.**

## Explanation

RACF R_PgmSignVer failed to sign a load module.

## User response

Locate the return code and reason code in R_PgmSignVer (IRRSPS00): Program Sign and Verify in *z/OS Security Server RACF Callable Services*.

## Source

Signing utility

**IEW6017S** **Concatenation of data sets for DD *ddname* is not allowed.**

## Explanation

Cannot define multiple data sets for INFILE or OUTFILE.

## User response

Resolve the errors.

## Source

Signing utility

**IEW6018S** **Member cannot be specified in DSN of DD *ddname*.**

## Explanation

Cannot specify member in DSN of INFILE or OUTFILE.

## User response

Resolve the errors.

## Source

Signing utility

**IEW6019S** **Error occurred during writing records of member *member*. Usually the reason is that DD OUTFILE is out of space.**

## Explanation

Usually the reason is that OUTFILE is out of space. Check for a preceding CEE3250C message issued by Language Environment.

## User response

Resolve the errors.

## Source

Signing utility

**IEW6020S** **Cannot create directory entry *member* in DD OUTFILE due to insufficient directory entry space.**

## Explanation

The directory entry space in OUTFILE is not enough.

## User response

Resolve the errors.

## Source

Signing utility

**IEW6021S** **Cannot allocate *mmmm* bytes of memory *above/below* the 16M line.**

## Explanation

Memory above or below the line is not sufficient.

## User response

Check the JCL attributes related to the memory limit, such as parameter REGION.

## Source

Signing utility

**IEW6022E** **Primary member name changed. Old=*name1*, new=*name2***

## Explanation

When ACTION=REPORT, IEWSIGN finds that a primary member has been renamed after its signing.

## User response

Sign the load module again.

## Source

Signing utility

**IEW6023E** **Member *member* has a directory entry error. Alias *alias* is in the DE record(s) but not in the directory.**

## Explanation

When ACTION=REPORT, IEWSIGN finds that an alias of a signed load module has been deleted after its signing. In this message and IEW6024E, DE records refers to records in the load module that record the directory entries associated with this module at the time of signing.

## User response

Sign the load module again.

## Source

Signing utility

**IEW6024E** **Member *member* has a directory entry error. Alias *alias* is in the directory but not in the DE record(s).**

## Explanation

When ACTION=REPORT, IEWSIGN finds that an alias of a signed load module was added after its signing. Refer to IEW6023E for the explanation of DE record.

## User response

Sign the load module again.

## Source

Signing utility

**IEW6025E** **Error in directory entry of member *member*. Length differs, INFILE=*len1*, Record=*len2*.**

## Explanation

The directory entries in INFILE and directory entry records do not match.

## User response

Sign the load module again.

## Source

Signing utility

**IEW6026E** **Error in directory entry of member *member*. Byte *offset* differs, INFILE=*byte1*, Record=*byte2*.**

## Explanation

The directory entries in INFILE and directory entry records do not match. Offset is 0-origin.

**Note:** The bytes printed in this message have masked out for bits to be ignored.

## User response

Sign the load module again.

## Source

Signing utility

**IEW6027E** ***number* reported load modules have errors.**

## Explanation

When ACTION=REPORT, IEWSIGN finds this number of load modules have errors.

## User response

Sign the load module again.

## Source

Signing utility

**IEW6028S** **The first character of alternate DD *ddname* cannot be a blank character.**

## Explanation

An alternate ddname cannot begin with a blank character. If a calling program wants to use the default value of this ddname, set all 8 bytes of the ddname to zero.

## User response

Correct the alternate ddname.

## Source

Signing utility

**IEW6029S** **Blocksize of DD OUTFILE(*xx*) must be equal to or larger than blocksize of DD INFILE (*yy*). The maximum value can be 32760.**

## Explanation

The blocksize of OUTFILE must be equal to or larger than blocksize of INFILE.

## User response

Set blocksize of OUTFILE with a valid value.

## Source

Signing utility

**IEW6030S** **Blocksize of DD OUTFILE(*xx*) must be equal to or larger than 1024. The maximum value can be 32760.**

## Explanation

The blocksize of OUTFILE must be equal to or larger than 1024.

## User response

Set blocksize of OUTFILE with a valid value.

## Source

Signing utility

**IEW6031E** **Member *member* cannot be signed since it has no CESD or TEXT record.**

## Explanation

The load module is corrupted and cannot be signed.

## User response

Rebuild the load module.

## Source

Signing utility

**IEW6032S** **Language Environment callable services *func* failed with message number *xxxx*.**

## Explanation

The signing utility called this Language Environment service and it failed. Refer in Language Environment documentation for more details.

## User response

Resolve the errors.

## Source

Signing utility

**IEW6033S** **The validated boot program signing support is not available in the security product.**

## Explanation

The RACF support for signing load modules has not been installed.

## User response

Resolve the errors.

## Source

Signing utility

**IEW6034S** **RDJFCB failed with return code *cccc* for DD *ddname*.**

## Explanation

The IEWSIGN utility called the DFSMS service RDJFCB to get DSNAME and volume serial of INFILE/OUTFILE, and this service failed. Locate the return code in section "Reading and Modifying a Job File Control Block (RDJFCB Macro) "“ located in Using System Macro Instructions in *z/OS DFSMSdfp Advanced Services*.

## User response

Resolve the errors.

## Source

Signing utility

**IEW6035S** **OBTAIN SEARCH failed with return code *cccc* for dataset *dsn* on volume *vol*.**

## Explanation

The IEWSIGN utility called the DFSMS service OBTAIN SEARCH to get data set attributes of INFILE/OUTFILE, and this service failed. Locate the return code in Return Codes from OBTAIN (Reading by Data Set Name) in *z/OS DFSMSdfp Advanced Services* .

## User response

Resolve the errors.

## Source

Signing utility

## IEAVBPRT: Validated Boot for z/OS print utility

The IEAVBPRT utility reports the following information after a validated boot IPL:

- Audit records that were created
- Certificate extracts that are being used
- Certificate extracts that were found not to be valid

For an enforce-mode IPL, no more than 1 audit record would be produced because any relevant issue would cause the system to enter a wait state right after building the audit record.

The IEAVBPRT utility provides options to generate a detailed report or a summary.

The same information is also provided by the IEAVBIPC utility within IPCS (VERBEXIT IEAVBIPC).

### Invoking the IEAVBPRT utility

Invoke the IEAVBPRT utility as a job step program (such as, `EXEC PGM=IEAVBPRT`). The report output is written according to the SYSPRINT DD statement. IEAVBPRT opens the SYSPRINT DD with the attributes `RECFM=FBA,LRECL=133`.

The following example shows sample JCL for such a job step:

```
//VBPRT1   EXEC PGM=IEAVBPRT,TIME=1440,PARM=parm
//SYSPRINT DD   SYSOUT=A
```

The value of the *parm* parameter can be:

**SUMMARY**
Generates a summary report. This is the default value.

**DETAIL**
Generates a detailed report.

### IEAVBPRT messages

The IEAVBPRT utility (and the IEAVBIPC utility in IPCS) issues the following messages:

**IEAVB001I Validated Boot Information**
This is the report header message.

**IEAVB003I Audit Information**
This message is followed by all the audit entries.

Within the audit entry messages, the term *DSNE* refers to a data set name entry. (Audit information is tracked by data set name.) Within those messages, the term *DSNE ModE* refers to a module name entry for a particular data set name entry. (Audit records are typically for a specific module within a specific data set.)

**IEAVB004I There are no valid certificates**
No valid certificates were found.

**IEAVB005I Valid Certificates**
This message is followed by information about each of the valid certificates.

**IEAVB006I No certificates were discarded**
There were no discarded certificates.

**IEAVB007I Discarded Certificates**
This message is followed by information about each of the discarded certificates.

**IEAVB008I Validated Boot is not in effect**
Validated boot is not in effect.

**IEAVB009I Unable to access *yyy* at *xxxxxxxx***
This message is issued only by the IEAVBIPC utility.

**IEAVB010I Unissued validated boot messages**

This message is issued only by the IEAVBIPC utility and is followed by information about each unissued message.

***(Audit mode)* IEAVB011I PLPA page data set was specified. It would not be used if enforce mode.**
***(Enforce mode)* IEAVB011I PLPA page data set was specified. It was not used.**

A PLPA page data set was specified. PLPA page data sets are not used for an enforce-mode IPL.

The enforce-mode form of this message is issued only by the IEAVBIPC utility.

***(Audit mode)* IEAVB012I Not enough storage-class memory to hold LPA. A wait state would result if enforce mode.**
***(Enforce mode)* IEAVB012I Not enough storage-class memory to hold LPA.**

There was insufficient storage-class memory to hold the LPA. This would cause an enforce-mode IPL to enter a wait state.

The enforce-mode form of this message is issued only by the IEAVBIPC utility.

## Contents of an IEAVBPRT report

The overall audit information displays one or more of the following lines:

```
There are no valid certificates
Could not retrieve certificate information
Total verification failures: n
Number of DSNEs: n
Number of DSNE ModEs: n
```

The last 2 lines are displayed only if the DETAIL option is in effect.

There might be no data set related audit entries, in which case the following line appears:

```
No dataset information is available
```

An audit entry begins with the following lines:

```
DSN(VOL): dataset_name(volume)
  Total DSN verification failures: n
  Number of DSNE ModEs: n
  [No module information is available]
```

- The "Number of DSNE ModEs" line appears only when DETAIL is in effect.
- The last line is displayed when there are no module name entries.

When DETAIL is not in effect and there is at least one module name entry, a table of module names and reasons appears:

```
Modname  Reason
m        r
```

When DETAIL is in effect and there is at least one module name entry:

```
Modname: m
  Reason: r
  {Key ID: xxxxxxxxx_xxxxxxxxx_xxxxxxxxx_xxxxxxxxx_xxxxxxxxx | Key ID: not known}
  Fetch Type: ft
  Number of failures: n
  When first failed: yyyy/mm/dd hh:mm:ss
  Cert Name: cn
  When signed: yyyy/mm/dd hh:mm:ss
  Machine loader error info: xxxxxxxx xxxx
```

- The "Key ID" and "When signed" lines appear only when the module signature is found.
- The "Cert Name" line appears only when a certificate with a matching key ID is found.
- The "Machine loader error info" line appears when there are machine loader errors, for module name IEAIPL00 only, for one of the following reasons:

- Module was not signed
- Signature verification failed
- Machine loader detected error(s)

Within the message text:

***m***
: The name of the module. When the module name ends with a X'C0' character, that character is displayed as '*'.

***r***
: One of the following reasons:

  **Module was not signed**
  The module is not signed.

  **Directory entry not found**
  The directory entry for the module could not be found.

  **Directory entry did not match**
  The directory for the module was found but does not match.

  **Signature not found**
  No signature record was found for this module.

  **Hash algorithm not valid**
  The signature record does not indicate a valid hash algorithm.

  **Signature algorithm not valid**
  The signature record does not indicate a valid signature algorithm.

  **Hash value not correct**
  The hash value in the signature record does not match the calculated hash value.

  **No certificate with matching key ID**
  The key ID in the signature record does not match any verification key available to this LPAR.

  **Signature verification failed**
  The signature verification operation did not complete successfully.

  **Overlay module**
  This is an overlay module. Signature support is not provided.

  **Signature record version not valid**
  The version of the signature record is not valid.

  **Machine loader detected error(s)**
  The machine loader detected one or more errors.

***ft***
: One of the following fetch types:

  **IPL**
  Indicates that the fetch is during the early IPL phase.

  **Nucleus**
  Indicates that the fetch is for a module that is being used to build the nucleus.

  **NIP**
  Indicates that the fetch is for a module during the later IPL phase.

  **LPA**
  Indicates that the fetch is for a module that is being placed into PLPA, MLPA, or FLPA.

An entry for a valid certificate contains the following lines:

```
Name: cert_name
    Key ID: xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx
    Successful uses: n
    Valid as of: yyyy/mm/dd hh:mm:ss
    Expiration: yyyy/mm/dd hh:mm:ss
    [Reason: Key is not valid]
```

- The "Key ID", "Valid as of", and "Expiration" lines appear only when DETAIL is requested.
- The "Reason: Key is not valid" line is determined after the system has started using the certificate. If this occurs, correct the certificate.

An entry for a discarded certificate contains the following lines:

```
Name: cert_name
    Reason: r
    KeyID: xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx
    Valid as of: yyyy/mm/dd hh:mm:ss
    Expiration: yyyy/mm/dd hh:mm:ss
```

- The "Key ID", "Valid as of", and "Expiration" lines appear only when DETAIL is requested.

Within the message text:

***r***

One of the following reasons:

**Not valid yet**
The certificate is not yet valid.

**Expired**
The certificate has expired.

**Key is not valid**
The key is not valid.

**Key type is not valid**
The key type is not valid.

**Key ID length is not valid**
The length of the key ID is not valid.

**Hash type is not valid**
The hash type is not valid.

**Hash length is not valid**
The length of the hash is not valid.

If any of these reasons occur, correct the certificate.

## IEAVBPRT return codes

*Table 7. Return codes for the IEAVBPRT utility*

| **Return code (decimal)** | **Meaning** |
|---|---|
| 0 | Successful completion. No audit information was found. |
| 2 | Successful completion. This was not a validated boot IPL. |
| 4 | Successful completion. Some audit information was found. |
| 8 | An invalid parameter was specified. |
| 12 | An invalid SYSPRINT data set was specified. |

## Examples

1. The following example shows a DETAIL entry for a module (within an entry for a data set):

```
Modname: IEAIPL00
  Reason: Module was not signed
  Fetch Type: IPL
  Number of failures: 1
  When first failed: 2022/10/19 13:15:07
  Machine loader error info: 12000000 3400
```

2. The following example shows a partial DETAIL entry for a data set and module:

```
IEAVB003I Audit Information
  Total verification failures: 1909
  Number of DSNEs: 7
  Number of DSNE ModEs: 1754

  DSN(VOL): SUPER.CSV.LOAD.PDS.HUGE.SIGNED(D16PK8)
    Total DSN verification failures: 1
    Number of DSNE ModEs: 1

    Modname: GM64
      Reason: No certificate with matching key ID
      Fetch Type: LPA
      Number of failures: 1
      When first failed: 2022/10/26 17:51:50
      Key ID: 21CC95D0_8A12F9FE_5AA01598_430EF6A0_8D58DFDE
      When signed: 2022/10/26 17:46:19
```

# Chapter 6. RACF

RACF support for Validated Boot for z/OS is provided in the PTFs for APARs OA61878 and OA61901.

In RACF, the callable service **R_PgmSignVer (IRRSPS00 or IRRSPS64)** is updated to support signing IPL data. For Validated Boot for z/OS, this service can produce signing output with the formats that are required for signing IPL data, such as Public Key Cryptographic Standards #7 (PKCS#7) Distinguished Encoding Rules (DER) format.

The topics in this section provide instructions for enabling the RACF support for Validated Boot for z/OS.

## IPL data signing for Validated Boot for z/OS

With Validated Boot for z/OS, your installation can ensure that its system IPL data is intact, untampered-with, and originates from a trusted build-time source. To use Validated Boot for z/OS, you must configure RACF to enable IPL data signing, as described in this topic.

This topic contains the following subtopics:

- “Overview of enabling your system for signed IPL data” on page 33
- “Certificate requirements for signing IPL data” on page 33
- “Defining the IRR.PROGRAM.V2.SIGNING profile” on page 34
- “Enabling IPL data signing for Validated Boot for z/OS” on page 36

## Overview of enabling your system for signed IPL data

In RACF, the callable service **R_PgmSignVer (IRRSPS00)** is used for enabling users to digitally sign programs. This service can also be used for signing IPL data, such as the system residence volume (SYSRES) contents, for Validated Boot for z/OS operations. When used in this manner, **R_PgmSignVer** produces signing output for IPL data in the required formats, such as Public Key Cryptographic Standards #7 (PKCS#7) Distinguished Encoding Rules (DER) format, or a simplified structure suitable for signature verification with the KDSA instruction.

In Validated Boot for z/OS, the following z/OS services use **R_PgmSignVer** to sign IPL data:

- Device Support Facility (ICKDSF), which requests the signature output in the PKCS#7 format.
- z/OS Binder, which requests the signature output in the simplified KDSA structure format.

For more information about the **R_PgmSignVer** service, see *z/OS Security Server RACF Callable Services*.

## Certificate requirements for signing IPL data

To enable your system for signing IPL data, you must create the required certificates for RACF.

This topic contains the following subtopics:

- “Required certificates” on page 34
- “Supported signature algorithms” on page 34
- “Required certificate extensions” on page 34

For examples of using RACDCERT GENCERT command to create certificates that meet these requirements, see “Steps for using a RACF-generated signing certificate stored in a key ring” on page 37. Otherwise, contact your external certificate authority (CA) and see “Steps for using an external signing certificate stored in a key ring” on page 39.

For details about using the RACDCERT GENCERT command, see *z/OS Security Server RACF Command Language Reference*.

### Required certificates

Your installation must supply a RACF key ring or an ICSF token that contains the following certificates:

- The default certificate in the RACF key ring or ICSF Token must have a private key and a key type of NISTECC with a 521-bit key size. This object is referred to as the *code-signing certificate*. The code-signing certificate can be self-signed or signed by a CA certificate.
- If the code-signing certificate is not a self-signed certificate, add the chain of CA certificates that signed the code-signing certificate up to the root CA. The R_PgmSignVer service supports a maximum of ten CA certificates in the key ring or ICSF token.

### Supported signature algorithms

The code-signing certificate and each CA certificate in the code-signing certificate chain must be signed by using one of the following signature algorithms:

- `sha256RSA`
- `sha224RSA`
- `sha384RSA`
- `sha512RSA`
- `sha256RSAPSS`
- `sha224RSAPSS`
- `sha384RSAPSS`
- `sha512RSAPSS`
- `sha256ECDSA`
- `sha224ECDSA`
- `sha384ECDSA`
- `sha512ECDSA`

### Required certificate extensions

The certificates require extensions, as follows:

- The code-signing certificate must have the KeyUsage extension with at least the digitalSignature indicator enabled.
- Each CA certificate in the code-signing certificate chain must have the BasicConstraints extension with the cA indicator enabled, or must not have a BasicConstraints extension.

## Defining the IRR.PROGRAM.V2.SIGNING profile

This topic describes the IRR.PROGRAM.V2.SIGNING profile, which is used for specifying the RACF key ring or ICSF token and the hash algorithm for Validated Boot for z/OS.

You must define APPLDATA information in one or more discrete profiles in the FACILITY class to specify the following information:

- Name of the z/OS signing key ring or token.
- Hash algorithm (or message digestion algorithm) that is to be used for signing IPL data. If you omit this value, SHA512 is used by default.

### Format of the profile name

The format of the IRR.PROGRAM.V2.SIGNING profile name is based on how you choose to assign signing key rings or tokens to users who are authorized to sign. The first four qualifiers of the profile name must be IRR.PROGRAM.V2.SIGNING.

The rest of the profile name reflects the available options for assigning key rings or tokens to signers. You can optionally append one or two more qualifiers to the profile name, as shown in the following list. RACF checks the profiles in the order that is listed and uses the first profile found that matches as follows.

**IRR.PROGRAM.V2.SIGNING.*groupid.userid***
: This profile assigns a key ring for the specified user when the user's current connect group is the specified group ID.

**IRR.PROGRAM.V2.SIGNING.*userid***
: This profile assigns a key ring for the specified user ID.

**IRR.PROGRAM.V2.SIGNING.*groupid***
: This profile assigns a key ring for the user IDs whose current connect group matches the group ID specified.

**IRR.PROGRAM.V2.SIGNING**
: This profile assigns a key ring to be used by all users in the RACF database for Validated Boot for z/OS signing.

**Rule:** No generic characters (*) are allowed in the name of a IRR.PROGRAM.V2.SIGNING[.*groupid*][.*userid*] profile.

**Note:** No access check is performed on the IRR.PROGRAM.V2.SIGNING[.*groupid*][.*userid*] profile.

## Format of the APPLDATA value

The format of the APPLDATA value in the IRR.PROGRAM.V2.SIGNING profile is as follows:

```
[hash-algorithm] [owning-userid]/key-ring-name
```

or

```
[hash-algorithm] *TOKEN*/token-name
```

The variables of the APPLDATA value are defined as follows:

***hash-algorithm***
: Specifies the message digestion algorithm to be used for Validated Boot for z/OS signing. The supported value is SHA512; no other values are supported. If you omit this value, RACF uses SHA512 by default. The hash-algorithm value is overridden if it is also specified in the R_PgmSignVer function call parameter list.

***owning-userid***
: Specifies the user ID that owns the Validated Boot for z/OS signing key ring. If you omit this value, RACF uses the user ID of the user who invokes the R_PgmSignVer signing service.

***/key-ring-name***
: Specifies the fully qualified name of the Validated Boot for z/OS signing key ring. This value must be preceded by the forward slash (/).

***/token-name***
: Specifies the Validated Boot for z/OS token name to be used for signing. This value must be preceded by `*TOKEN*` and the forward slash (/).

**Rules:**

- The only space character that is allowed in the APPLDATA value is the single space after the *hash-algorithm* value. If *hash-algorithm* is omitted, no space is allowed in the APPLDATA value.
- No extraneous characters are allowed in the APPLDATA value.

**Notes:**

- RACF does not check the format of the APPLDATA value when you define a IRR.PROGRAM.V2.SIGNING profile.

- The RDEFINE command that is used to create this profile converts the key ring or token name that is defined in the APPLDATA to all uppercase characters. Therefore, do not use mixed case characters when you define the key ring name or token name.

### Examples of profile names

The format of the IRR.PROGRAM.V2.SIGNING profile name is based on how you choose to assign the signing key rings or tokens to users who are authorized to sign.

- The following profile defines that user ID ZSIGNER, when this user's current connect group ID is BUILD, uses the VB_SIGNING_KEYRING key ring defined for user ID BUILDID for Validated Boot for z/OS signing operations.

```
RDEFINE FACILITY IRR.PROGRAM.V2.SIGNING.BUILD.ZSIGNER
APPLDATA('SHA512 BUILDID/VB_SIGNING_KEYRING')
```

- The following profile defines that user ID ZSIGNER uses the VB_SIGNING_KEYRING key ring that is defined to user ID ZSIGNER for Validated Boot for z/OS signing operations, unless a more specific profile applies.

```
RDEFINE FACILITY IRR.PROGRAM.V2.SIGNING.ZSIGNER
APPLDATA('SHA512 ZSIGNER/VB_SIGNING_KEYRING')
```

- The following profile defines that users whose current connect group is PROD will use the VB_SIGNING_KEYRING key ring that is defined to user ID PRODID for Validated Boot for z/OS signing operations, unless a more specific profile applies.

```
RDEFINE FACILITY IRR.PROGRAM.V2.SIGNING.PROD
  APPLDATA('SHA512 PRODID/VB_SIGNING_KEYRING')
```

- The following profile defines that users will use the VB_SIGNING_KEYRING key ring that is defined to user ID RACFADM for Validated Boot for z/OS signing operations, unless a more specific profile applies.

```
RDEFINE FACILITY IRR.PROGRAM.V2.SIGNING
  APPLDATA('SHA512 RACFADM/VB_SIGNING_KEYRING')
```

- The following profile defines that users will use the RACFADM.VBTOKEN ICSF token for Validated Boot for z/OS signing operations, unless a more specific profile applies.

```
RDEFINE FACILITY IRR.PROGRAM.V2.SIGNING
  APPLDATA('SHA512 *TOKEN*/RACFADM.VBTOKEN')
```

## Enabling IPL data signing for Validated Boot for z/OS

It is possible to perform IPL data signing by using RACF-generated or externally supplied certificates, and storing the signing key in either a RACF key ring or ICSF token, as needed. This topic provides instructions for each of these scenarios.

This topic describes the following scenarios for signing IPL data. Choose the one that best matches your particular use case.

*Table 8. Scenarios for signing IPL data*

| Scenario | See the following topic... |
|---|---|
| You plan to use:<br>• RACF key ring<br>• Certificate generated by using RACF<br>• Signing key stored in the PKDS data set. | "Steps for using a RACF-generated signing certificate stored in a key ring" on page 37 |

*Table 8. Scenarios for signing IPL data (continued)*

| Scenario | See the following topic... |
|---|---|
| You plan to use:<br>• RACF key ring<br>• Certificate imported from an external source<br>• Signing key stored in RACF. | "Steps for using an external signing certificate stored in a key ring" on page 39 |
| You plan to use:<br>• ICSF token<br>• Certificate generated by using RACF<br>• Signing key stored in an ICSF token in the TKDS data set. | "Steps for using a RACF-generated signing certificate stored in an ICSF token" on page 41 |

## Steps for using a RACF-generated signing certificate stored in a key ring

**Note:** This procedure creates a CA certificate and then a code-signing certificate signed by that CA. However, it is acceptable to create just a self-signed code-signing certificate with the required attributes.

To enable a group to digitally sign IPL data by using a signing certificate that you create with RACF, perform the following steps.

1. If it does not already exist, create a certificate-authority (CA) certificate that you can use to issue a signing certificate for users who will perform Validated Boot for z/OS signing.

   **Note:** It is recommended, but not required, to create the public and private keys for the CA certificate in the ICSF PKDS. Doing so provides the strongest level of security of the private key. The RSA(PKDS) keyword causes public and private keys to be stored in the ICSF PKDS.

   **Example:**

   ```
   RACDCERT CERTAUTH GENCERT
   SUBJECTSDN(OU('MyCompany Validated Boot Signing CA') O('MyCompany') C('US'))
   SIZE(4096) RSA(PKDS) WITHLABEL('Validated Boot Signing CA')
   ```

   ______________________________________________________________________

2. Create a code-signing certificate for users who will perform Validated Boot for z/OS signing. In the following example, the code signing certificate is owned by user ZSIGNER and signed by the CA certificate.

   **Note:** It is recommended, but not required, to create the public and private keys for the code-signing certificate in the ICSF PKDS. Doing so provides the strongest level of security of the private key. The NISTECC(PKDS(VBSIGNINGKEY)) keyword causes public and private keys to be stored in the ICSF PKDS with the VBSIGNINGKEY label. This makes it easier to share the signing key with the members of the group.

   **Example:**

   ```
   RACDCERT ID(ZSIGNER) GENCERT SUBJECTSDN(CN('Validated Boot Signing Cert')
   O('MyCompany') C('US')) NISTECC(PKDS(VBSIGNINGKEY)) SIZE(521)
   WITHLABEL('Validated Boot Signing Cert') SIGNWITH(CERTAUTH LABEL('Validated Boot Signing
   CA'))
   KEYUSAGE(HANDSHAKE DOCSIGN)
   ```

   ______________________________________________________________________

3. Create a RACF key ring to hold the certificates that you created in Steps "1" on page 37 and "2" on page 37. In the following example, the key ring is owned by user ZSIGNER.

**Rule:** Specify the key ring name in all uppercase characters so that it matches the key ring that is specified in the IRR.PROGRAM.V2.SIGNING profile APPLDATA operand, which is converted to uppercase when created with the RDEFINE command.

**Example:**

```
RACDCERT ID(ZSIGNER) ADDRING(VB_SIGNING_KEYRING)
```

___

4. Connect both certificates that you created in Steps "1" on page 37 and "2" on page 37 to the key ring you created in Step "3" on page 37.

   **Rule:** The signing certificate must be the default certificate in the ring.

   **Example:**

```
RACDCERT ID(ZSIGNER) CONNECT(CERTAUTH LABEL('Validated Boot Signing CA')
RING(VB_SIGNING_KEYRING))
RACDCERT ID(ZSIGNER) CONNECT(ID(ZSIGNER) LABEL('Validated Boot Signing Cert') DEFAULT
RING(VB_SIGNING_KEYRING))
```

___

5. Define the RDATALIB class profile that covers the key ring that was created in the previous steps. Permit the members of the BUILDGRP group to sign the code by using the key ring and private key of the code-signing certificate.

   **Example:**

```
RDEFINE RDATALIB ZSIGNER.VB_SIGNING_KEYRING.LST UACC(NONE)
PERMIT ZSIGNER.VB_SIGNING_KEYRING.LST CLASS(RDATALIB) ID(BUILDGRP) ACCESS(UPDATE)
```

   In this example, the owner of the key ring is user ZSIGNER, and the key ring name is VB_SIGNING_KEYRING. Thus, the RDATALIB profile name that covers that resource is: ZSIGNER.VB_SIGNING_KEYRING.LST (*<ring owner>*.*<ringname>*.LST).

   - If the RDATALIB class is not already active, activate and RACLIST it.

```
SETROPTS CLASSACT(RDATALIB) RACLIST(RDATALIB)
```

   - If the RDATALIB class is already active and RACLISTed, refresh it.

```
SETROPTS RACLIST(RDATALIB) REFRESH
```

___

6. Define the signing profile. In the following example, a signing profile is defined for the group of authorized signers called BUILDGRP.

   **Example:**

```
RDEF FACILITY IRR.PROGRAM.V2.SIGNING.BUILDGRP APPLDATA('SHA512 ZSIGNER/VB_SIGNING_KEYRING')
```

   For more information about the profile structure, see "Defining the IRR.PROGRAM.V2.SIGNING profile" on page 34.

___

7. Permit the users who are connected to the BUILDGRP group to perform the signing, using the signing key stored in the PKDS. In this example, assume that a CSFDSG profile is already defined in the CSFSERV resource class.

   **CAUTION:** IBM recommends that the CSFSERV and CSFKEYS classes be RACLISTed. However, be aware that a number of ICSF services are available when the CSFSERV or CSFKEYS classes are not active. Therefore, if you activate and RACLIST these classes, any product that currently uses the services that are protected by these classes can fail due to lack of authorization.

**Example:**

```
PERMIT CSFDSG CLASS(CSFSERV) ID(BUILDGRP) ACCESS(READ)
PERMIT VBSIGNINGKEY CLASS(CSFKEYS) ID(BUILDGRP) ACCESS(READ)
```

If the CSFSERV and CSFKEYS classes are not already active, activate and RACLIST them.

```
SETROPTS CLASSACT(CSFSERV CSFKEYS) RACLIST(CSFSERV CSFKEYS)
```

If the CSFSERV and CSFKEYS classes are already active and RACLISTed, refresh them.

```
SETROPTS RACLIST(CSFSERV CSFKEYS) REFRESH
```

You have now enabled a RACF group to digitally sign IPL data by using a signing certificate that you created with RACF.

## Steps for using an external signing certificate stored in a key ring

**Note:** This procedure creates a CA certificate and a code-signing certificate that is signed by that CA. However, it is acceptable to create just a self-signed code-signing certificate with the required attributes.

**Before you begin:** Obtain or locate the root certificate-authority (CA) certificate of an external CA — and the intermediate CAs, if needed — and store them in a cataloged, variable-blocked (VB) MVS data set.

To enable a group to digitally sign IPL data by using a signing certificate that you obtain from an external certificate-authority (CA), perform the following steps.

1. Add the root CA certificate of the external CA to RACF, specifying the data set in which it is stored.

   **Example:**

   ```
   RACDCERT CERTAUTH ADD(<data set containing the root CA cert>) WITHLABEL('External VB root
   CA')
   ```

   If you want to use an intermediate CA, enter this command to add it. To add multiple intermediate CAs, repeat this command for each of them with the appropriate labels.

   **Example:**

   ```
   RACDCERT CERTAUTH ADD(<data set containing the intermediate CA cert>) WITHLABEL('External VB
   intermediate CA')
   ```

2. Add the signing certificate to RACF. In the following example, the signing certificate is owned by user ZSIGNER and the signing certificate is contained in a PKCS #12 package that is stored in a data set.

   **Example:**

   ```
   RACDCERT ID(ZSIGNER) ADD(<data set containing the signing cert in pkcs12 package>)
   WITHLABEL('Validated Boot Signing Cert') PASSWORD('<pw>')
   ```

3. Create a RACF key ring to hold the certificates you added in Steps "1" on page 39 and "2" on page 39. In the following example, the key ring is owned by user ZSIGNER.

   **Rule:** Specify the key ring name in all uppercase characters so that it matches the key ring that is specified in the IRR.PROGRAM.V2.SIGNING profile APPLDATA operand. This value is converted to uppercase when the profile is created with the RDEFINE command.

   **Example:**

```
RACDCERT ID(ZSIGNER) ADDRING(VB_SIGNING_KEYRING)
```

___

4. Connect all of the certificates that you obtained in Steps "1" on page 39 and "2" on page 39 to the key ring that you created in Step "3" on page 39.

   **Rule:** The signing certificate must be the default certificate in the ring.

   **Example:**

   ```
   RACDCERT ID(ZSIGNER) CONNECT(CERTAUTH LABEL('External VB root CA')
   RING(VB_SIGNING_KEYRING))

   RACDCERT ID(ZSIGNER) CONNECT(CERTAUTH LABEL('External VB intermediate CA')
   RING(VB_SIGNING_KEYRING))

   RACDCERT ID(ZSIGNER) CONNECT(ID(ZSIGNER) LABEL('Validated Boot Signing Cert') DEFAULT
   RING(VB_SIGNING_KEYRING))
   ```

___

5. Define the RDATALIB class profile that covers the key ring that was created in the previous steps. Permit the members of the BUILDGRP group to sign the code by using the key ring and private key of the code-signing certificate.

   **Example:**

   ```
   RDEFINE RDATALIB ZSIGNER.VB_SIGNING_KEYRING.LST UACC(NONE)
   PERMIT ZSIGNER.VB_SIGNING_KEYRING.LST CLASS(RDATALIB) ID(BUILDGRP) ACCESS(UPDATE)
   ```

   In this example, the owner of the key ring is user ZSIGNER, and the key ring name is VB_SIGNING_KEYRING. Thus, the RDATALIB profile name that covers that resource is: ZSIGNER.VB_SIGNING_KEYRING.LST (*<ring owner>*.*<ringname>*.LST).

   - If the RDATALIB class is not already active, activate and RACLIST it.

     ```
     SETROPTS CLASSACT(RDATALIB) RACLIST(RDATALIB)
     ```

   - If the RDATALIB class is already active and RACLISTed, refresh it.

     ```
     SETROPTS RACLIST(RDATALIB) REFRESH
     ```

___

6. Define the signing profile. In the following example, a signing profile is defined for a group of authorized signers called BUILDGRP.

   **Example:**

   ```
   RDEF FACILITY IRR.PROGRAM.V2.SIGNING.BUILDGRP APPLDATA('SHA512 ZSIGNER/VB_SIGNING_KEYRING')
   ```

   Refresh the FACILITY class.

   ```
   SETR RACLIST(FACILITY) REFRESH
   ```

   For more information about the profile structure, see "Defining the IRR.PROGRAM.V2.SIGNING profile" on page 34.

___

7. Permit the users who are connected to the BUILDGRP group to perform the signing, using the signing key stored in the RACF database. In this example, assume that the CSF1TRC, CSF1PKS and CSF1TRD profiles are already defined in the CSFSERV resource class.

   **CAUTION:** IBM recommends that the CSFSERV class be RACLISTed. However, be aware that a number of ICSF services are available when the CSFSERV class is not active. Therefore, if you activate and RACLIST this class, any product that currently uses the services that are protected by this class can fail due to lack of authorization.

**Example:**

```
PERMIT CSF1TRC CLASS(CSFSERV) ID(BUILDGRP) ACCESS(READ)
PERMIT CSF1PKS CLASS(CSFSERV) ID(BUILDID) ACCESS(READ)
PERMIT CSF1TRD CLASS(CSFSERV) ID(BUILDID) ACCESS(READ)
```

If the CSFSERV class is not already active, activate and RACLIST it.

```
SETROPTS CLASSACT(CSFSERV) RACLIST(CSFSERV)
```

If the CSFSERV class is already active and RACLISTed, refresh it.

```
SETROPTS RACLIST(CSFSERV) REFRESH
```

___________________________________________________________________

You have now enabled a RACF group to digitally sign IPL data by using a signing certificate that you obtained from an external certificate-authority (CA).

## Steps for using a RACF-generated signing certificate stored in an ICSF token

**Note:** This example creates a CA certificate and then a code-signing certificate signed by that CA. However, it is acceptable to create just a self-signed code-signing certificate with the required attributes.

To enable a group to digitally sign IPL data by using a certificate that you create with RACF and store in an ICSF token, perform the following steps.

1. Create an ICSF token with the RACDCERT command. The token will contain the certificates and keys that you create in Steps "2" on page 41 and "3" on page 41.

   **Rule:** Specify the token name to match the token name that is specified in the IRR.PROGRAM.V2.SIGNING profile APPLDATA operand.

   **Example:**

   ```
   RACDCERT ADDTOKEN(ZSIGNER.VBTOKEN)
   ```

   ___________________________________________________________________

2. If it does not already exist, create a certificate-authority (CA) certificate that you can use to issue a signing certificate for users who will perform Validated Boot for z/OS signing.

   **Example:**

   ```
   RACDCERT CERTAUTH GENCERT
   SUBJECTSDN(OU('MyCompany Validated Boot Signing CA') O('MyCompany') C('US'))
   SIZE(4096) RSA WITHLABEL('Validated Boot Signing CA')
   ```

   ___________________________________________________________________

3. Create a code-signing certificate for users who will perform Validated Boot for z/OS signing. In the following example, the code signing certificate is owned by user ZSIGNER and signed by the CA certificate.

   **Example:**

   ```
   RACDCERT ID(ZSIGNER) GENCERT SUBJECTSDN(CN('Validated Boot Signing Cert')
   O('MyCompany') C('US')) NISTECC(TOKEN(ZSIGNER.VBTOKEN)) SIZE(521) WITHLABEL('Validated Boot
   Signing Cert')
   SIGNWITH(CERTAUTH LABEL('Validated Boot Signing CA')) KEYUSAGE(HANDSHAKE DOCSIGN)
   ```

   ___________________________________________________________________

4. Bind the code-signing certificate and its CA certificate to the ICSF token.

   **Example:**

   ```
   RACDCERT BIND(CERTAUTH LABEL('Validated Boot Signing CA') TOKEN(ZSIGNER.VBTOKEN))
   RACDCERT BIND(ID(ZSIGNER) LABEL('Validated Boot Signing Cert') DEFAULT USAGE(PERSONAL)
   TOKEN(ZSIGNER.VBTOKEN))
   ```

   ______________________________________________________________________

5. Define the signing profile. In the following example, a signing profile is defined for a group of authorized signers called BUILDGRP.

   **Example:**

   ```
   RDEF FACILITY IRR.PROGRAM.V2.SIGNING.BUILDGRP APPLDATA('SHA512 *TOKEN*/ZSIGNER.VBTOKEN')
   ```

   For more information about the profile structure, see "Defining the IRR.PROGRAM.V2.SIGNING profile" on page 34.

   ______________________________________________________________________

6. Permit the users who are connected to the BUILDGRP group to access the signing key. The key is stored in the ICSF token and is protected by the CSFSERV and CRYPTOZ classes. In this example, assume that the profiles for CSFDSG, CSF1TRL, and CSF1GAV are already defined in the CSFSERV resource class.

   **CAUTION:** IBM recommends that the CSFSERV and CRYPTOZ classes be RACLISTed. However, be aware that a number of ICSF services are available when the CSFSERV or CRYPTOZ class is not active. Therefore, if you activate and RACLIST these classes, any product that currently uses the services that are protected by these classes can fail due to lack of authorization.

   **Example:**

   ```
   PERMIT CSFDSG CLASS(CSFSERV) ID(BUILDGRP) ACCESS(READ)
   PERMIT CSF1TRL CLASS(CSFSERV) ID(BUILDGRP) ACCESS(READ)
   PERMIT CSF1GAV CLASS(CSFSERV) ID(BUILDGRP) ACCESS(READ)
   PERMIT USER.ZSIGNER.VBTOKEN CLASS(CRYPTOZ) ID(BUILDGRP) ACCESS(READ)
   ```

   - If the CSFSERV and CRYPTOZ classes are not already active, activate and RACLIST them.

     ```
     SETROPTS CLASSACT(CSFSERV CRYPTOZ) RACLIST(CSFSERV CRYPTOZ)
     ```

   - If the CSFSERV and CRYPTOZ classes are already active and RACLISTed, refresh them.

     ```
     SETROPTS RACLIST(CSFSERV CRYPTOZ) REFRESH
     ```

   ______________________________________________________________________

You have now enabled a group to digitally sign IPL data by using a signing certificate that you created with RACF and stored in an ICSF token.

## R_PgmSignVer (IRRSPS00): Program Sign and Verify

### Function

The R_PgmSignVer service provides the functions required to apply a digital signature to a z/OS program object, and the functions required to verify such a signature. The signing services are intended for use by the z/OS program binder. The verification services are intended for use by the z/OS loader.

The signing services consist of the following functions:

1. Initialize signing - Allocates and initializes a work area to perform message digestion (hash) against the program's data, and reads the digital certificates from the program signing key ring.

2. Digest intermediate program data - Hashes a portion of the program’s data for signing.
3. Generate signature - Hashes the final portion of the program’s data, if provided, and generates the digital signature by encrypting the calculated hash using the private key from the default certificate in the key ring. Any resources obtained by the initialize signing function are freed before returning.
4. Cleanup - Frees resources obtained by the initialize signing function. To be called if the signature generation is not completed by the generate signature function (for example, for recovery cleanup).

The verification services consist of the following functions:

1. Initialize signature-verification - Allocates and initializes a work area to perform message digestion (hash) against the program’s data, and hashes any initial program data that is supplied.
2. Digest intermediate program data - Hashes a portion of the program’s data for verification.
3. Final verification - Hashes the final portion of the program’s data, if provided. The signature provided with the program is then decrypted with the public key from the end-entity certificate that accompanies the signature, and the two hash values are compared to verify the signature. Any resources obtained by the initialize signature-verification function are freed before returning.
4. Cleanup - Frees resources obtained by the initialize signature-verification function. To be called if that signature-verification is not completed by calling the final verification function (for example, for recovery cleanup)
5. Interrogate directive - Generate appropriate return code and perform auditing according to security settings when a program signature cannot be verified.

## Requirements

**Authorization:**
Any PSW key in supervisor or problem state.

**Note:** In the following documentation, a caller that is in either supervisor state or a system key is referred to as “authorized”. Otherwise, the caller is referred to as “unauthorized”.

**Dispatchable unit mode:**
Task of user

**Cross memory mode:**
PASN = HASN

**AMODE:**
31 or 64

**RMODE:**
Any

**ASC mode:**
Primary or AR mode

**Recovery mode:**
ESTAE. Caller cannot have a FRR active.

**Serialization:**
Enabled for interrupts

**Locks:**
No locks held

**Control parameters:**
The parameter list and the work area must be in the primary address space. The words containing the ALETs must be in the primary address space. The Num_parms parameter must be in the primary address space.

## Linkage conventions

Callers in 31-bit addressing mode should link-edit the IRRSPS00 stub module with their code, and use the IRRPCOMP mapping macro. Callers in 64-bit addressing mode should link-edit the IRRSPS64 stub module with their code, and use the IRRPCOMY mapping macro.

## RACF authorization

For an unauthorized caller, the caller's identity must have sufficient authority to use the key ring or the token that is specified in the parameter list. Or, if not specified in the parameter list, then as defined in the appropriate profile in the FACILITY class, as follows:

- IRR.PROGRAM.SIGNING[.*groupid*][.*userid*] profile, when Num_parms is 10.
- IRR.PROGRAM.V2.SIGNING[.*groupid*][.*userid*] profile, when Num_parms is 11 and the Function_attributes is nonzero.

Also, the caller requires authority to the private key contained in the key ring as determined by the R_datalib callable service and ICSF. .

If a token is used instead of a key ring, READ access to the ICSF resources CSF1TRL and CSF1GAV in the CSFSERV class is required.

When Num_parms is 11 and the Function_attributes is nonzero, the ICSF subsystem must be operational and configured for PKA operations with the appropriate crypto hardware, depending on where the NISTECC signing key is stored. Further, the caller must be authorized to the appropriate ICSF resources.

The hardware and authorization that are required for the signing process are as follows:

- When the signing key is stored in the ICSF token data set (TKDS):
  - Enterprise PKCS#11 cryptographic coprocessor is required.
  - READ authority to the CSFDSG resource in the CSFSERV class
  - READ authority to the USER.*<pkcs11 token name>* in the CRYPTOZ class
- When the signing key is stored in the ICSF PKA key data set (PKDS):
  - A Crypto Express3 coprocessor (CEX3C), or later, is required.
  - READ authority to the CSFDSG resource in the CSFSERV class
  - READ authority to the resource that represents the key label in the CSFKEYS class.
- When the signing key is stored in the RACF database:
  - READ authority to the CSF1TRC, CSF1PKS, and CSF1TRD resources in the CSFSERV class

For the signature-verification services, there are no authorization requirements, regardless of the caller's state.

## Format

```
     CALL IRRSPS00 (Work_area,
                    ALET, SAF_return_code,
                    ALET, RACF_return_code,
                    ALET, RACF_reason_code,
                    Num_parms,
                    Function_code,
                    Function_parmlist,
                    Function_attributes
                   )
```

## Parameters

**Work_area**
The name of a 1024-byte work area for SAF. The work area must be in the primary address space.

**ALET**

The name of a word that contains the ALET for the following parameter. Each parameter must have an ALET specified. Each ALET must be 0 for this service. The words that contain the ALETs must be in the primary address space.

**SAF_Return_Code**

The name of a fullword in which the SAF router returns the SAF return code.

**RACF_Return_Code**

The name of a fullword in which the service routine stores the return code.

**RACF_Reason_Code**

The name of a fullword in which the service routine stores the reason code.

**Num_parms**

Specifies the name of a fullword that contains the total number of parameters in the parameter list. The contents of this field must be set to decimal ten or eleven. If Function_attributes is specified, this field must be set to 11.

**Function_code**

The name of a 2-byte area that contains the Function code. The function code has one of the following values:

**X'0001'**

Initialize signing. (Function name SIGINIT.) This function must be called before calling any of the other signing functions.

**X'0002'**

Digest intermediate program data for signature generation. (Function name SIGUPDAT.) This function is optional. It should be called only if all the program's data cannot be processed on one call to generate signature. It can be called multiple times before calling generate signature.

**X'0003'**

Generate signature. (Function name SIGFINAL.) This function finalizes the signature generation and returns the result. It also frees any work area storage that might have been allocated.

**X'0004'**

Terminates the signing operation and frees resources that are allocated by SIGINIT. (Function name SIGCLEAN.) This function should be called only if signature generation is not to be finalized with a call to SIGFINAL. All R_PgmSignVer functions perform this cleanup if they return an error to the caller. The caller needs to call the cleanup function only if it is terminating for its own reason.

**X'0005'**

Initialize signature-verification and optionally digest initial program data. (Function name VERINIT.) This function must be called before calling any of the other verification functions except VERINTER (interrogate directive).

**X'0006'**

Digest intermediate program data for signature-verification. (Function name VERUPDAT.) This function is optional. It should be called only if all the program's data cannot be processed on the VERINIT and VERFINAL calls. It can be called multiple times before performing final verification.

**X'0007'**

Perform final verification. (Function name VERFINAL.) This function finalizes the signature-verification and returns the result. It also audits the event and frees any work area storage that might have been allocated. If all the program data can be specified in a single call, then VERFINAL can be called without first calling VERINIT. See Table 15 on page 49 for more information.

**X'0008'**

Terminates the signing operation and frees resources that are allocated by VERINIT. (Function name VERCLEAN.) This function should be called only if signature generation is not to be finalized with a call to VERFINAL. All R_PgmSignVer functions perform this cleanup if they return an error to the caller. The caller must call the cleanup function only if it is terminating for its own reason.

**X'0009'**

Interrogate directive. (Function name VERINTER.) This function examines the directive (supplied within the ICHSFENT in the function-specific parameter list) to determine the appropriate action.

This would be used for cases in which VERFINAL is not called. For example, when digital signature processing is required, but the module does not have a digital signature. This function is not available to unauthorized callers.

**Function_parmlist**

Specifies the name of the function code-specific parameter list area for the Function_code specified.

All address fields are 8-byte addresses. When referring to 31-bit storage addresses, the caller must make sure that the high-order word of the address field is set to binary zeros.

*Table 9. Function_parmlist for SIGINIT*

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_SI_PLIST | Structure | In | Function-specific parameter list for signing initialization. |
| PGSN_SI_EYE | 8 characters | In | Eye catcher, 8 characters. Actual value must be set by invoker: 'SIGINIT '. |
| PGSN_SI_VERS | 4 byte numeric | In | The version number for this function-specific parameter list. The contents of this field must be set to binary zero. |
| PGSN_SI_PGM_NAME_LEN | 4 byte numeric | In | Length of the name of the program being signed. The length must not exceed 8 characters. |
| PGSN_SI_PGM_NAME@ | Address of | In | Address of the name of the program being signed.<br>**Note:** This parameter is used to derive the name/ token that is used for subsequent calls. As such, it does not necessarily need to be the program name, but must be a unique value, which does not result in a name collision with other signing operations. |
| PGSN_SI_KEYRING_NAME@ | Address of | In | Address of the name of the SAF key ring or the ICSF token that contains the certificates to be used for signing. This address is meaningful only if PGSN_SI_KEYRING_LEN is a nonzero value.<br>For a SAF key ring, the name has the following syntax:<br>*owning-userid*/*ring-name*<br>The owning-userid (but not the slash) can be omitted if the key ring is owned by the user ID associated with the calling application.<br>For an ICSF token, the name has the following syntax:<br>*TOKEN*/*token-name* |
| PGSN_SI_KEYRING_LEN | 4 byte numeric | In | Length of the name of the SAF key ring or the ICSF token that contains the certificates to be used for signing. If this value is set to zero, the PGSN_SI_KEYRING_NAME@ value is ignored. |
| PGSN_SI_SIGINFO_LEN | 4 byte numeric | Out | Length of the ZOSSignatureInfo structure, which is returned as part of the signature area structure in the SIGFINAL call. |
| PGSN_SI_DIGEST_ALG | 1 byte numeric | In | Numeric value indicating what message digest algorithm to use for the signing. A value of 1 indicates that SHA256 is to be used if Num_parms = 10. A value of 2 indicates that SHA512 is to be used if Num_parms = 11.<br>To have the security manager determine the algorithm to use, set this field to zero. |

*Table 10. Function_parmlist for SIGUPDAT*

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_SU_PLIST | Structure | In | Function-specific parameter list for intermediate signing. |
| PGSN_SU_EYE | 8 characters | In | Eye catcher, 8 characters. Actual value must be set by invoker: 'SIGUPDAT'. |
| PGSN_SU_VERS | 4 byte numeric | In | The version number for this function-specific parameter list. The contents of this field must be set to binary zero. |
| PGSN_SU_PGM_NAME_LEN | 4 byte numeric | In | Length of the name of the program being signed. The length must not exceed 8 characters. |
| PGSN_SU_PGM_NAME@ | Address of | In | Address of the name of the program being signed. Must be the same as the value supplied on the SIGINIT call. |
| PGSN_SU_PGM_DATA@ | Address of | In | Address of a structure specifying the intermediate ranges of data to sign. The structure is mapped by PGSN_DATA_RANGE. See usage note "7" on page 62 in "Usage notes for program verification" on page 61 for the format of this structure. |

*Table 11. Function_parmlist for SIGFINAL*

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_SF_PLIST | Structure | In | Function-specific parameter list for final signing. |
| PGSN_SF_EYE | 8 characters | In | Eye catcher, 8 characters. Actual value must be set by invoker: 'SIGFINAL'. |
| PGSN_SF_VERS | 4 byte numeric | In | The version number for this function-specific parameter list. The contents of this field must be set to binary zero. |
| PGSN_SF_PGM_NAME_LEN | 4 byte numeric | In | Length of the name of the intermediate ranges program being signed. The length must not exceed 8 characters. |
| PGSN_SF_PGM_NAME@ | Address of | In | Address of the name of the program being signed. Must be the same as the value supplied on the SIGINIT call. |
| PGSN_SF_PGM_DATA@ | Address of | In | Address of a structure specifying the final ranges of data to sign. The structure is mapped by PGSN_DATA_RANGE. See usage note "7" on page 62 in "Usage notes for program verification" on page 61 for the format of this structure. |
| PGSN_SF_SIG_AREA@ | Address of | Out | Address of the allocated signature area structure. See usage note "6" on page 59 in "Usage notes for program signing" on page 58 for the final ranges format of the area. |
| PGSN_SF_SUBPOOL | 1 byte numeric | In | Subpool to be used for allocation of the signature data structure. For unauthorized callers, this must be a value in the range 1 – 127. |

*Table 12. Function_parmlist for SIGCLEAN*

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_SC_PLIST | Structure | In | Function-specific parameter list for signing cleanup. |
| PGSN_SC_EYE | 8 characters | In | Eye catcher, 8 characters. Actual value must be set by invoker: 'SIGCLEAN'. |
| PGSN_SC_VERS | 4 byte numeric | In | The version number for this function-specific parameter list. The contents of this field must be set to binary zero. |

*Table 12. Function_parmlist for SIGCLEAN (continued)*

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_SC_PGM_NAME_LEN | 4 byte numeric | In | Length of the name of the program being signed. The length must not exceed 8 characters. |
| PGSN_SC_PGM_NAME@ | Address of | In | Address of the name of the program being signed. Must be the same as the value supplied on the SIGINIT call. |

*Table 13. Function_parmlist for VERINIT*

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_VI_PLIST | Structure | In | Function-specific parameter list for verification initialization. |
| PGSN_VI_EYE | 8 characters | In | Eye catcher, 8 characters. Actual value must be set by invoker: 'VERINIT '. |
| PGSN_VI_VERS | 4 byte numeric | In | The version number for this function-specific parameter list. The contents of this field must be set to binary zero. |
| PGSN_VI_PGM_NAME_LEN | 4 byte numeric | In | For unauthorized callers, length of the name of the program being verified. The length must not exceed 8 characters. Ignored for authorized callers. |
| PGSN_VI_PGM_NAME@ | Address of | In | For unauthorized callers, address of the name of the program being verified. Ignored for authorized callers. |
| PGSN_VI_CONTEXT@ | Address of | Out | For authorized callers, address of the allocated verify context that the caller should pass in to subsequent verification calls. Ignored for unauthorized callers. |
| PGSN_VI_PGM_DATA@ | Address of | In | Address of a structure specifying the initial ranges of data to verify. The structure is mapped by PGSN_DATA_RANGE. See usage note "7" on page 62 in "Usage notes for program verification" on page 61 for the format of this structure. |
| PGSN_VI_SIGINFO@ | Address of | In | Address of the ZOSSignatureInfo structure that is extracted from the program object being verified. |
| PGSN_VI_SIGINFO_LEN | 4 byte numeric | In | Length of the ZOSSignatureInfo structure that is extracted from the program object being verified. |
| PGSN_VI_DIGEST_ALG | 1 byte numeric | In | Numeric value indicating what message digest algorithm to use for the verification. A value of 0 means the value that is contained in the ZOSSignatureInfo structure should be used. This is the only supported value. |

*Table 14. Function_parmlist for VERUPDAT*

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_VU_PLIST | Structure | In | Function-specific parameter list for intermediate verification. |
| PGSN_VU_EYE | 8 characters | In | Eye catcher, 8 characters. Actual value must be set by invoker: 'VERUPDAT'. |
| PGSN_VU_VERS | 4 byte numeric | In | The version number for this function-specific parameter list. The contents of this field must be set to binary zero. |
| PGSN_VU_PGM_NAME_LEN | 4 byte numeric | In | For unauthorized callers, length of the name of the program being verified. The length must not exceed 8 characters. Ignored for authorized callers. |

*Table 14. Function_parmlist for VERUPDAT (continued)*

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_VU_PGM_NAME@ | Address of | In | For unauthorized callers, address of the name of the program being verified. Must be the same as the value supplied on the VERINIT call. Ignored for authorized callers. |
| PGSN_VU_CONTEXT@ | Address of | In | For authorized callers, address of the verify context area that is allocated on the VERINIT call. Ignored for unauthorized callers. |
| PGSN_VU_PGM_DATA@ | Address of | In | Address of a structure specifying the intermediate ranges of data to verify. The structure is mapped by PGSN_DATA_RANGE. See usage note "7" on page 62 in "Usage notes for program verification" on page 61 for the format of this structure. |

*Table 15. Function_parmlist for VERFINAL*

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_VF_PLIST | Structure | In | Function-specific parameter list for final verification. |
| PGSN_VF_EYE | 8 characters | In | Eye catcher, 8 characters. Actual value must be set by invoker: 'VERFINAL'. |
| PGSN_VF_VERS | 4 byte numeric | In | The version number for this function-specific parameter list. The contents of this field must be set to binary zero. |
| PGSN_VF_PGM_NAME_LEN | 4 byte numeric | In | For unauthorized callers, length of the name of the program being verified. The length must not exceed 8 characters. Ignored for authorized callers.<br><br>If the length is zero, it is assumed that no VERINIT call was made, and the signature is generated based on the data that is supplied in this call, by using the default digest algorithm. |
| PGSN_VF_PGM_NAME@ | Address of | In | For unauthorized callers, address of the name of the program being verified. Must be the same as the value supplied on the VERINIT call. Ignored for authorized callers. |
| PGSN_VF_CONTEXT@ | Address of | In | For authorized callers, address of the verify context area that is allocated on the VERINIT call. Ignored for unauthorized callers. If the address is zero, it is assumed that no VERINIT call was made, and the signature is generated based on the data that is supplied in this call, by using the default digest algorithm. |
| PGSN_VF_PGM_DATA@ | Address of | In | Address of a structure specifying the final ranges of data to verify. The structure is mapped by PGSN_DATA_RANGE. See usage note "7" on page 62 in "Usage notes for program verification" on page 61 for the format of this structure. |
| PGSN_VF_LOGSTRING@ | Address of | In | Address of an area that consists of a 1-byte length field followed by character data (up to 255 bytes) to be included in any audit records that are created. If the address or the length byte is 0, this parameter is ignored. |
| PGSN_VF_ICHSFENT@ | Address of | In | For authorized callers, address of the FASTAUTH entity parameter mapping containing the directive (previously retrieved from RACF by Contents Supervision). This parameter is optional. See usage notes "6" on page 61 and "16" on page 62 in "Usage notes for program verification" on page 61. Ignored for unauthorized callers. |

Table 15. Function_parmlist for VERFINAL (continued)

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_VF_SIGINFO@ | Address of | In | Address of the ZOSSignatureInfo structure that is extracted from the program object being verified. This field is required if VERFINAL is the only call being made. It is ignored if it was already passed to VERINIT. |
| PGSN_VF_SIGINFO_LEN | 4 byte numeric | In | Length of the ZOSSignatureInfo structure that is extracted from the program object being verified. This field is required if VERFINAL is the only call that is being made. It is ignored if it was already passed to VERINIT. |

Table 16. Function_parmlist for VERCLEAN

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_VC_PLIST | Structure | In | Function-specific parameter list for verification cleanup. |
| PGSN_VC_EYE | 8 characters | In | Eye catcher, 8 characters. Actual value must be set by invoker: 'VERCLEAN'. |
| PGSN_VC_VERS | 4 byte numeric | In | The version number for this function-specific parameter list. The contents of this field must be set to binary zero. |
| PGSN_VC_PGM_NAME_LEN | 4 byte numeric | In | For unauthorized callers, length of the name of the program being verified. The length must not exceed 8 characters. Ignored for authorized callers. |
| PGSN_VC_PGM_NAME@ | Address of | In | For unauthorized callers, address of the name of the program being verified. Must be the same as the value supplied on the VERINIT call. Ignored for authorized callers. |
| PGSN_VC_CONTEXT@ | Address of | In | For authorized callers, address of the verify context area that is allocated on the VERINIT call. Ignored for unauthorized callers. |

Table 17. Function_parmlist for VERINTER

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_ID_PLIST | Structure | In | Function-specific parameter list for interrogating the directive. |
| PGSN_ID_EYE | 8 characters | In | Eye catcher, 8 characters. Actual value must be set by invoker: 'VERINTER'. |
| PGSN_ID_VERS | 4 byte numeric | In | The version number for this function-specific parameter list. The contents of this field must be set to binary zero. |
| * | 4 characters | In | Reserved |
| PGSN_ID_ ICHSFENT@ | Address of | In | For authorized callers, address of the FASTAUTH entity parameter mapping (previously retrieved from RACF by Contents Supervision). Ignored for unauthorized callers. |
| PGSN_ID_LOGSTRING@ | Address of | In | Address of an area that consists of a 1-byte length field followed by character data (up to 255 bytes) to be included in any audit records that are created. If the address or the length byte is 0, this parameter is ignored. |

Table 17. Function_parmlist for VERINTER (continued)

| Field | Attributes | Usage | Description |
|---|---|---|---|
| PGSN_ID_EVENT | 1 byte numeric | In | Constant indicating what sigver event was detected:<br>• x'01' – Digital signature processing is required but the module does not have a digital signature.<br>• x'02' – Digital signature processing is required. The PDSE directory entry for the module indicates it's signed but the digital signature is missing. |

**Function_attributes**

The name of a 4-byte area that contains bit settings that indicate the output format for signing. These bit settings are applicable only when Num_parms = 11. It must set to X'00000000', except for the following sign functions.

The bit settings for functions SIGINIT, SIGUPDAT, and SIGFINAL are mapped as follows:

**X'80000000'**
Indicates that the signature is to be returned in PKCS#7 format. See usage note "7" on page 59.

**X'40000000'**
Indicates that the signature is to be returned in KDSA format. See usage note "7" on page 59 .

Otherwise, if a different value is specified in the bit settings, SAF return code 8, RACF return code 100, RACF reason code *xx* is returned. Here, *xx* is the offset of the parameter in error, relative to the start of COMP or COMY.

**Notes:**

1. The same attribute value must be specified for functions SIGINIT, SIGUPDAT, and SIGFINAL. Otherwise, error 8 100 *xx* is returned.
2. The processing for the verify functions when Num_parms is 11 and Function_attributes is X'00000000' is the same as when Num_parms is 10.

## Return and reason codes

R_PgmSignVer can return the following values in the return and reason code parameters:

Table 18. Return and reason codes

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 0 | 0 | 0 | Successful completion |
| 4 | 0 | 0 | RACF not installed |
| 8 | 8 | 4 | An internal error occurred during RACF processing of the requested function. |
| 8 | 8 | 8 | Unable to establish a recovery environment. |
| 8 | 8 | 12 | Function not available for unauthorized callers. |
| 8 | 100 | xx | A parameter list error is detected. The RACF reason code identifies the parameter in error. The reason code is the offset of the parameter in error, relative to the start of COMP or COMY. |

*Table 18. Return and reason codes (continued)*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 104 | yy | A function-specific parameter list (pointed to by the *Function_parmlist* parameter) error has been detected. The RACF reason code identifies the field in error. The reason code is the offset of the field in error, relative to the start of the function-specific parameter list. When the field is an address, the error might pertain to the address itself, or to something to which it points. |

In addition to the preceding, R_PgmSignVer can return function-specific return and reason codes:

*Table 19. SIGINIT-specific return and reason codes*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 8 | 100 | Signature operation is already in progress for the specified program name. |
| 8 | 8 | 104 | Security Manager is unable to determine the key ring or token to use. |
| 8 | 8 | 108 | Syntax error in the key ring name or token name specified as an input parameter or within the APPLDATA of the RACF FACILITY class profile. |
| 8 | 8 | 112 | Key ring or token does not exist or does not contain a default certificate. |
| 8 | 8 | 116 | Caller not authorized to use R_datalib to access the key ring or token. |
| 8 | 8 | 120 | Certificate chain in the key ring or token is incomplete. |
| 8 | 8 | 124 | Certificate chain contains more than 10 certificates, or key ring or token contains more than 50 certificates. Some of these might not constitute part of the trust chain. However, you should not connect any certificates that do not. |
| 8 | 8 | 128 | CA certificate in the key ring or token does not have certificate signing capability. (KeyUsage extension present but keyCertSign flag is off or BasicConstraints extension is present but cA flag is off.) |
| 8 | 8 | 132 | Default certificate or token in key ring does not have a private key. |

*Table 19. SIGINIT-specific return and reason codes (continued)*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 8 | 136 | Default certificate in key ring or token does not have code signing capability.<br>• When Num_parms = 10, the keyUsage extension is optional. If the extension is present, both the digitalSignature bit and the nonRepudiation bit must be set.<br>• When Num_parms = 11, the keyUsage extension must be present and the digitalSignature bit must be set. |
| 8 | 8 | 140 | The certificate signature algorithm of one or more certificates in the key ring or token is not supported. |
| 8 | 8 | 144 | The key type of one or more certificates in the key ring or token is not supported. This reason code is also issued for the following conditions:<br>• When Num_parms = 10, but the private key of the signing certificate is stored in ICSF.<br>• When Num_parms = 11, the signing key can be stored in ICSF, but it must be a 521-bits NIST Elliptic Curve Cryptography(ECC) key. Also, the key size of any other certificates in the key ring or token must be at least 2048 bits for RSA keys, or 224 bits for NIST ECC and Brainpool ECC keys. |
| 8 | 8 | 148 | The specified message digest algorithm is not supported.<br>• When Num_parms = 10, the supported digest algorithm is SHA256.<br>• When Num_parms = 11, the supported digest algorithm is SHA512. |
| 8 | 8 | 152 | CA or signing certificate is expired or not yet active. |
| 8 | 8 | 156 | The signing certificate does not have the Subject Key Identifier extension. This return and reason code is applicable only when Num_parms = 11. |
| 8 | 12 | xx | Unexpected error returned from R_datalib. RACF reason code is the DataGetFirst/DataGetNext reason code that is returned by R_datalib. |
| 8 | 16 | xx | Unexpected error returned from IEANTCR. RACF reason code is the return code that is returned by IEANTCR. |

*Table 19. SIGINIT-specific return and reason codes (continued)*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 20 | 0x00xxyyyy | An unexpected error is returned from ICSF. The hexadecimal reason code value is formatted as follows: **xx** ICSF return code. **yyyy** ICSF reason code. |

*Table 20. SIGUPDAT-specific return and reason codes*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 8 | 100 | Signature operation has not been initialized for the specified program name. |
| 8 | 12 | xx | Unexpected error returned from IEANTRT. RACF reason code is the return code that is returned by IEANTRT. |
| 8 | 2nn | xx | Unexpected error from the cryptographic module. The return code is 200+*nn* where *nn* identifies the function being performed. The reason code is the return code from the cryptographic module. This information should be reported to IBM service. |

*Table 21. SIGFINAL-specific return and reason codes*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 8 | 16 | No input data to be signed. This return and reason code is applicable only when Num_parms = 11. |
| 8 | 8 | 24 | The load of an ICSF service failed and the RACF reason code is the return code of the LOAD macro. This return code and reason code are applicable only when Num_parms = 11. |
| 8 | 8 | 100 | Signature operation has not been initialized for the specified program name. |
| 8 | 12 | xx | Unexpected error returned from IEANTRT. RACF reason code is the return code that is returned by IEANTRT. |
| 8 | 16 | xx | Unexpected error returned from IEANTDL. RACF reason code is the return code that is returned by IEANTDL. |

*Table 21. SIGFINAL-specific return and reason codes (continued)*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 20 | 0x*ffxxyyyy* | An unexpected error is returned from ICSF. This return and reason code is applicable only when Num_parms = 11.<br>The hexadecimal reason code value is formatted as follows:<br>**ff**<br>ICSF function:<br>• 01: CSFPTRC<br>• 02: CSFPPKS<br>• 03: CSFPTRD<br>• 04: CSNDDSG<br>**xx**<br>ICSF return code<br>**yyyy**<br>ICSF reason code |
| 8 | 2nn | xx | Unexpected error from the cryptographic module. The return code is 200+*nn* where *nn* identifies the function being performed. The reason code is the return code from the cryptographic module. This information should be reported to IBM service. |

*Table 22. SIGCLEAN-specific return and reason codes*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 8 | 100 | Signature operation has not been initialized for the specified program name. |
| 8 | 12 | xx | Unexpected error returned from IEANTRT. RACF reason code is the return code that is returned by IEANTRT. |
| 8 | 16 | xx | Unexpected error returned from IEANTDL. RACF reason code is the return code that is returned by IEANTDL. |

*Table 23. VERINIT-specific return and reason codes*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 8 | 100 | Verification operation is already in progress for the specified program name. |
| 8 | 12 | xx | Unexpected error returned from IEANTCR. RACF reason code is the return code that is returned by IEANTCR. |

Table 23. VERINIT-specific return and reason codes (continued)

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 16 | 116 | The program verification module (IRRPVERS) is not loaded. See *z/OS Security Server RACF Security Administrator's Guide* and *z/OS Security Server RACF System Programmer's Guide* for information about configuring and loading the verification module with the IRRVERLD program. |
| 8 | 2nn | xx | Unexpected error from the cryptographic module. The return code is 200+*nn* where *nn* identifies the function being performed. The reason code is the return code from the cryptographic module. This information should be reported to IBM service. |

Table 24. VERUPDAT-specific return and reason codes

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 8 | 100 | Verification operation has not been initialized for the specified program name. |
| 8 | 12 | xx | Unexpected error returned from IEANTRT. RACF reason code is the return code that is returned by IEANTRT. |
| 8 | 16 | 116 | The program verification module (IRRPVERS) is not loaded. See *z/OS Security Server RACF Security Administrator's Guide* and *z/OS Security Server RACF System Programmer's Guide* for information about configuring and loading the verification module with the IRRVERLD program. |
| 8 | 2nn | xx | Unexpected error from the cryptographic module. The return code is 200+*nn* where *nn* identifies the function being performed. The reason code is the return code from the cryptographic module. This information should be reported to IBM service. |

Table 25. VERFINAL-specific return and reason codes

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 0 | 0 | See reason codes for SAF return code 8, RACF return code 16. | Signature failed verification. Continue the load. |
| 8 | 8 | 100 | Verification operation has not been initialized for the specified program name. |
| 8 | 12 | xx | Unexpected error returned from IEANTRT. RACF reason code is the return code that is returned by IEANTRT. |

*Table 25. VERFINAL-specific return and reason codes (continued)*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 16 | See the following | Signature failed verification. Fail the load. |
| The following group of reason codes are considered problems with the program signature (the zOSSignatureInfo structure). These would cause the load to fail when FAILLOAD(BADSIGONLY) or FAILLOAD(ANYBAD) is in effect. | | | |
| | | 4 | The ZOSSignatureInfo structure is missing or not correct. |
| | | 8 | Signature algorithm in ZOSSignatureInfo is not supported. |
| | | 12 | Signer certificate is revoked. The certificate status is NOTRUST. |
| | | 16 | Certificate chain is incomplete. |
| | | 20 | One or more CA certificates do not have certificate signing capability. (KeyUsage extension present but keyCertSign flag is off or BasicConstraints extension is present but cA flag is off.) |
| | | 24 | End-entity certificate does not have code signing capability. (KeyUsage extension present but digitalSignature or nonRepudiation flag is off.) |
| | | 28 | The certificate signature algorithm of one or more certificates is not supported. |
| | | 32 | The type or size of key found in one or more certificates is not supported. |
| | | 36 | CA or signing certificate was expired or not yet active at the time that the module was signed. |
| | | 40 | Digital signature not valid. |
| | | 44 | Unsupported certificate format. |
| The following group of reason codes are the additional conditions that would cause the load to fail due to signature processing, but do not represent a bad signature. These would cause the load to fail when FAILLOAD(ANYBAD) is in effect, but not FAILLOAD(BADSIGONLY). | | | |
| | | 100 | The program appears to be correctly signed but one of the following conditions exists: <br>• The root CA certificate in the zOSSignatureInfo structure of the program object is not connected to the signature-verification key ring. <br>• The root CA certificate is marked NOTRUST. |
| | | 104 | The FACILITY class profile, IRR.PROGRAM.SIGNATURE.VERIFICATION, is missing. |
| | | 108 | The APPLDATA information in the FACILITY class profile, IRR.PROGRAM.SIGNATURE.VERIFICATION, is missing or not correct. |

*Table 25. VERFINAL-specific return and reason codes (continued)*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| | | 112 | The signature-verification key ring is missing. |
| | | 116 | The program verification module (IRRPVERS) is not loaded. See *z/OS Security Server RACF Security Administrator's Guide* and *z/OS Security Server RACF System Programmer's Guide* for information about configuring and loading the verification module with the IRRVERLD program. |
| | | 120 | An error occurred while performing a cryptographic self-test on the IRRPVERS module during initialization. Contact IBM support. |
| 8 | 2nn | xx | Unexpected error from the cryptographic module. The return code is 200+*nn* where *nn* identifies the function being performed. The reason code is the return code from the cryptographic module. This information should be reported to IBM service. |

*Table 26. VERCLEAN-specific return and reason codes*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 8 | 8 | 100 | Verification operation has not been initialized for the specified program name. |
| 8 | 12 | xx | Unexpected error returned from IEANTRT. RACF reason code is the return code that is returned by IEANTRT. |
| 8 | 16 | xx | Unexpected error returned from IEANTDL. RACF reason code is the return code that is returned by IEANTDL. |

*Table 27. VERINTER-specific return and reason codes*

| SAF return code | RACF return code | RACF reason code | Explanation |
|---|---|---|---|
| 0 | 0 | 0 | Continue the load. |
| 8 | 8 | 0 | Fail the load. |

## Usage notes

## Usage notes for program signing

1. This service tracks the resources that are used for signing by using a task-related name/token pair. The 16–byte token name has the following format:

```
IRRPSIGNprogram-name
```

Where *program-name* is one of the parameters that are provided by the caller. Therefore, for any given series of SIGINIT, SIGUPDAT, SIGFINAL, and SIGCLEAN calls used to sign a single program object, the program name value must be the same.

2. Calls to this service that use different program name values are considered independent operations.
3. For a given program name, SIGINIT must be called before calling any of SIGUPDAT, SIGFINAL, or SIGCLEAN.
4. For a given program name, SIGINIT cannot be called a second time without terminating the first SIGINIT with a call to SIGFINAL or SIGCLEAN.
5. For a given program name, it is the caller's responsibility to call the SIGCLEAN function if signature generation is not completed by calling SIGFINAL. All R_PgmSignVer functions perform this cleanup if they return an error to the caller. The caller must call the cleanup function if it is terminating for its own reason.
6. When Num_parms = 11 and Function_Attributes = X'40000000', PGSN_SF_SIG_AREA@ must be set to zeros on input. Otherwise, a function-specific parameter list error (8/104/*yy*) is returned. The signature area that is allocated and returned to the caller in the PGSN_SF_SIG_AREA@ parameter by SIGFINAL has the following format:

*Table 28. PGSN_SF_SIG_AREA@ signature area format*

| **Offset** | **Length** | **Description** |
|---|---|---|
| 0 | 4 | Eye catcher, "PSSD". |
| 4 | 4 | Length of entire area, including the eyecatcher. |
| 8 | 1 | Subpool used to obtain the area storage. |
| 9 | 3 | Reserved. |
| 12 | 4 | Length of z/OS signature information area. |
| 16 | * | ZOSSignatureInfo structure to be included in the signed program object. See the next usage note for the format. |

7. The ZOSSignatureInfo structure, which is returned in the signature area, is the signature data that is to be placed in the signed program object.

   - When Num_parms = 10, it is DER encoded according to the following ASN.1 definition:

```
ZOSSignatureInfo ::= SEQUENCE {
  signDetails SignatureDetails
  certs SET OF Certificate -- In reverse hierarchy order, EE to root
  signature BIT STRING -- PKCS #1 format - Encrypted DigestInfo}

SignatureDetails ::= SEQUENCE { -- DER encoding included in data signed
  version INTEGER(0)
  signatureAlg AlgorithmIdentifier -- From PKCS #1
  signatureTime OCTET STRING(12) -- TIME DEC,ZONE=UTC,DATETYPE=YYYYMMDD -- format (EBCDIC)

}
```

   - When Num_parms = 11 and Function_Attributes = X'80000000', it is DER encoded according to the following ASN.1 definition:

```
ZOSSignatureInfo ::= SEQUENCE { (contentInfo)

  OBJECT IDENTIFIER signedData (1 2 840 113549 1 7 2)
  [0] { (content)
    SEQUENCE { (SignedData)
      INTEGER 3 (CMSversion)
      SET { (DigestAlgorithmIdentifiers)
        SEQUENCE { (DigestAlgorithmIdentifier)
          OBJECT IDENTIFIER sha2-512 (2 16 840 1 101 3 4 2 3)
        }
      }
      SEQUENCE { (EncapsulatedContentInfo)
        OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
      }
```

```
       SET { (SignerInfos)
          SEQUENCE { (SignerInfo)
           INTEGER 3  (CMSversion)
           [0] { (SignerIdentifier)
           OCTET STRING (SubjectKeyIdentier)
           }

           SEQUENCE { (DigestAlgorithmIdentifier)
            OBJECT IDENTIFIER sha2-512 (2 16 840 1 101 3 4 2 3)
           }
           SEQUENCE {  (SignatureAlgorithmIdentifier)
             OBJECT IDENTIFIER (1 2 840 10045 4 3 4)
           }
           OCTET STRING (SignatureValue)
             SEQUENCE {
               INTEGER R
               INTEGER S
             }
         } (end of SignerInfo SEQUENCE)
       }  (end of SignerInfos SET)
     }  (end of SignedData SEQUENCE)
   }  (end of content)
 }  (end of contentInfo SEQUENCE)
```

- When Num_parms = 11 and Function_Attributes = X’40000000’, the structure is as follows:

```
ZOSSignatureInfo ::= SignatureValue,

                RValue CHAR(80 ),
                -- 80 bytes in hex: right justified R value of the signature padded with
zeros to the left
                SValue CHAR(80),
                -- 80 bytes in hex: right justified S value of the signature padded with
zeros to the left
                SigningHash CHAR(64),
                -- 64 bytes in hex, SHA512 hash of the to-be-signed object
          SigningKeyID CHAR(20),
                -- 20 bytes in hex, SHA1 hash of the signing key ID
                SigningCertID CHAR(32),
                -- 32 bytes in hex, SHA256 hash of the signing certificate fingerprint
          DigestAlg FIXED(8),
                -- integer represents the Digest Algorithm, 2 maps to SHA512
          SigningAlg FIXED(8)
                -- integer represents the Signing Algorithm, 2 maps to SHA512ECDSA
```

8. - When Num_parms = 10, the only supported algorithm for the signatureAlg field is sha256WithRSAEncryption with NULL parameters.
   - When Num_parms = 11, the only supported signing algorithm is SHA512ECDSA (OID 1.2.840.10045.4.3.4).
9. It is the caller's responsibility to free the signature area when it is no longer needed.
10. The supported message digest algorithm is :
    - SHA256 when Num_parms = 10.
    - SHA512 when Num_parms = 11.
11. The supported certificate key type is :
    - RSA when Num_parms = 10. The maximum RSA key size is 4096 bits.
    - NISTECC with a 521-bit key size when Num_parms = 11.
12. The supported certificate signature algorithms on the signing certificate chain are described as follows:

    **When Num_parms = 10**

    - sha256RSA
    - sha1RSA

    **When Num_parms = 11**

    - sha256RSA
    - sha224RSA

- sha384RSA
- sha512RSA
- sha256RSAPSS
- sha224RSAPSS
- sha384RSAPSS
- sha512RSAPSS
- sha256ECDSA
- sha224ECDSA
- sha384ECDSA
- sha512ECDSA

The supported certificate public key types and minimum key sizes on the signing certificate chain are:

- RSA – 2048 bits
- NISTECC/BPECC – 224 bits

13. All numeric parameters are treated as unsigned.
14. All length parameters must be nonzero unless otherwise indicated.
15. On SIGINIT, if the key ring or token to use is not specified, the security manager determines the key ring or token that is based on security settings. See the topics on program signing and verification and Validated Boot for z/OS support in *z/OS Security Server RACF Security Administrator's Guide* for information on these security settings and on how to populate the key ring or token. No more than 10 certificates can exist within the trust chain, starting with the code signer and ending with the self-signed certificate authority certificate.
16. If no program data is ever passed in by the caller, a digital signature is generated solely for the SignatureDetails structure documented. When Num_parms = 11, if no program data is passed in by the caller, the SIGFINAL function fails with SAF return code 8, RACF return code 8, RACF reason code 16.

## Usage notes for program verification

1. For unauthorized callers, this service tracks the resources used for verification in a ‘context’ using a task related name/token pair. The 16–byte token name has the following format:

   ```
   IRRPVERFprogram-name
   ```

   Where program-name is one of the parameters provided by the caller. Consequently, for any given series of VERINIT, VERUPDAT, VERFINAL, and VERCLEAN calls used to verify the signature of a single program object, the program name must be the same.
2. Calls to this service using different program names are considered independent operations.
3. For a given program name, VERINIT must be called before calling any of VERUPDAT, VERFINAL (with the exception documented in the descriptions of the PGSN_VF_CONTEXT@ and PGSN_VF_PGM_NAME_LEN fields in the VERFINAL parameter list), or VERCLEAN.
4. For a given program name, VERINIT cannot be called a second time without terminating the first VERINIT with a call to VERFINAL or VERCLEAN.
5. For a given program name, it is the caller’s responsibility to call the VERCLEAN function in the event that signature generation will not be completed by calling VERFINAL. Note that all R_PgmSignVer functions will perform this cleanup if they return an error to the caller. The caller only needs to call the cleanup function if it is terminating for its own reason.
6. If auditing is required, it is performed in the VERFINAL (or VERINTER) call. Auditing is only performed when the ICHSFENT is provided by an authorized caller, subject to the audit settings from the directive within and the outcome of the VERFINAL service.

7. Some signature generation and all verification functions allow, from a pointer in the function-specific parameter list, the specification of an array of ranges of data to be hashed. This is optional. If the address is 0, no data will be hashed. The ranges are defined using the structure mapped by PGSN_DATA_RANGE in the IRRPCOMP mapping macro. This structure must exist in storage within the primary address space. The structure consists of an ALET followed by a fullword specifying the number of ranges which follow (if the number of ranges is 0, no data will be hashed). This is followed by an array of pointer pairs. Each pointer is an 8-byte pointer. AMODE(31) callers must set the high order fullword of the pointer fields to 0. The first pointer is the address of the first byte of the range, and the second pointer is the address of the last byte of the range (they can be the same, for a length of 1). The maximum number of ranges which can be specified per call is defined in the PGSN_DATA_NUM_RANGES_MAX constant.

| **Field** | **Attributes** | **Description** |
|---|---|---|
| PGSN_DATA_RANGE | Structure | Ranges of data to verify. |
| PGSN_DATA_ALET | 4 byte numeric | The ALET for the address space containing the data. |
| PGSN_DATA_NUM_RANGES | 4 byte numeric | The number of data ranges in the following array, not to exceed PGSN_DATA_NUM_RANGES_MAX. |
| PGSN64_DATA_RANGE_LIST | Array | Repeating array of the following data items. |
| PGSN_DATA_START@ | Address of | Address of the first byte in the range. |
| PGSN_DATA_END@ | Address of | Address of the last byte in the range. |

8. The default message digest algorithm is SHA256. This is the only supported message digest algorithm.
9. The ZOSSignatureInfo structure is DER encoded. It has the following ASN.1 definition:

```
ZOSSignatureInfo ::= SEQUENCE {
   signDetails     SignatureDetails
   certs           SET OF Certificate -- In reverse hierarchy order, EE to root
   signature       BIT STRING         -- PKCS #1 format - Encrypted DigestInfo
}

SignatureDetails ::= SEQUENCE {       -- DER encoding included in data signed
   version         INTEGER(0)
   signatureAlg    AlgorithmIdentifier -- From PKCS #1
   signatureTime     OCTET STRING(12)      -- TIME DEC,ZONE=UTC,DATETYPE=YYYYMMDD
                                       -- format (EBCDIC)
}
```

10. The only supported algorithm for the signatureAlg field is sha256WithRSAEncryption with NULL parameters.
11. The only supported certificate key type is RSA. The maximum RSA key size is 4096 bits.
12. The supported certificate signature algorithms are:
    - sha256WithRSAEncryption
    - sha1WithRSAEncryption
13. All numeric parameters are treated as unsigned.
14. All length parameters must be non-zero unless otherwise indicated.
15. The program signature-verification key ring is specified using the APPLDATA field of FACILITY class profile IRR.PROGRAM.SIGNATURE.VERIFICATION. See *z/OS Security Server RACF Security Administrator's Guide* for more information about creating profiles.
16. If there is no ICHSFENT, and thus no directive, which is supplied by the caller, the verification occurs on the signature, but there is no check for the root CA certificate being trusted, and no auditing performed.

## Related services

None.

# Chapter 7. Commands

## Displaying system configuration information (M)

Use the DISPLAY M command to display the status of sides, processors, ICRFs, channel paths, devices, storage-class memory (SCM) and central storage, or to compare the current hardware configuration to the configuration in a CONFIGxx parmlib member.

The DISPLAY M command can accept the subchannel set number to qualify the input device number. The output of message IEE097I includes the applicable subchannel set number.

When you specify a device number that might be mistaken for the device name, precede the device number with a slash. The slash is optional with a 3-digit device number.

### Syntax

**D M**

```
D M[=CHP[{(xx)|(xx-xx)|(list)}[,PATHINFO]]
    |=CONFIG[(xx)]
    |=CORE[(x)|(list)]
    |={CPUAD|CPU}[(x)|(list)]
    |=CU(xxxx)
    |={DEVICE|DEV}[([/]devnum)|([/]lowdevnum-[/]highdevnum)|(list)]
                                [,{ZHYPERLINK|ZHL|READSEC}]
    |={DEVICE|DEV}([/]devnum,(chp))[,ROUTE={TODEV|FROMDEV|BOTH}[,HEALTH]]
                                [,LINKINFO={FIRST|LAST|REFRESH|COMPARE}]
    |={DEVICE|DEV}(([/]devnum),chp)[,ROUTE={TODEV|FROMDEV|BOTH}[,HEALTH]]
                                [,LINKINFO={FIRST|LAST|REFRESH|COMPARE}]
    |=HIGH
    |=HSA
    |=SCM(DETAIL)
    |=SIDE[(id)]
    |={STORAGE|STOR}[(ddddM-ddddM)|(list)|(E[=id])]
    |={STORAGE|STOR}[(ddddM-ddddM)|(list)]
    |=SWITCH(sssss [,pp[-pp] [,pp[-pp]]...])
    |=(parm[,parm]...)

   [,L={a|name|name-a}]
```

### Parameters

**M**

The system is to display information about system configuration. When you enter DISPLAY M with no operands, the system displays the starting address and length of each portion of the hardware system area (HSA). The system also displays the status of all processors, ICRFs, central storage, channel paths, storage-class memory (SCM) and devices, depending on the type of processor or processor complex.

If the processor complex is partitioned, the system does not provide information about resources that are not part of the configuration on which you issue the command. Message IEE174I gives you the status of resources on the side from which you issue the command and tells you that information about the other side is unavailable. If you are running your processor complex in single-image mode with all resources in one side offline, message IEE174I identifies the other side as being offline but gives you the information about those resources. For example, to partition a processor complex, you configure offline the resources on one side. To verify that those resources are offline, issue the DISPLAY M=SIDE command. The display lists the side as offline and gives the status of the resources.

**CHP**

The system is to display the online and offline status of channel paths. If you do not specify any channel path, the system displays the status of all channel paths, as well as a status of either “managed and online” or “managed and offline” as part of the support of dynamic channel path management. For a description of the display format, see message IEE174I.

**(*xx*)**

A single channel path identified by *xx*. The channel path identifier can have a value from 0 to FF.

**(*xx*-*xx*)**

A range of channel path identifiers. The starting and ending channel path identifiers can have a value from 0 to FF.

**(*list*)**

One or more single channel path identifiers, or a combination of single channel path identifiers and ranges of channel path identifiers, each separated by a comma.

**[,PATHINFO]**

Display information for the paths sharing the specified channel path identifiers. This information includes, for example, the destination link address, link speed, and Fibre Channel Endpoint Security status.

**CONFIG[(*xx*)]**

The system is to display the differences between the current configuration and the configuration described in the CONFIGxx parmlib member. If you omit *xx*, the system assumes that you mean CONFIG00.

For a description of the display format, see message IEE097I.

You can also start this function from the HCD dialog. For details refer to the "Process Display M=CONFIG(xx) Command" in *z/OS HCD User's Guide*.

**CORE**

The system is to display the online or offline status of one or more cores as well as the HiperDispatch setting and multithreading (MT) mode. If you do not specify any core identifiers, the system displays the online or offline status of all cores.

For a description of the display format, see message IEE174I.

**(*x*)**

A single core identified by a core identifier in hexadecimal format.

**(*list*)**

One or more core identifiers, each separated by a comma.

**Note:** When you issue the DISPLAY M=CORE command while a zIIP boost is active, any zIIPs added due to the boost processing are displayed as "B" instead of "I". The key at the end of the IEE174I display contains B BOOST (TRANSIENT) zIIP.

**Note:** When you issue the DISPLAY M=CORE command from a PR/SM partition, the system displays the status for the logical cores and ICRFs defined to the partition.

**CPUAD *or* CPU**

The system is to display the online or offline status of one or more processors and any ICRFs attached to those processors. See message IEE174I.

If you do not specify any processor identifiers, the system displays the online or offline status of all processors and any ICRFs attached to them. Whether you specify a processor identifier or not, the system displays “N” when a processor is neither online or offline, but is recognized by the machine.

**Note:** When you issue the DISPLAY M=CPU command while a zIIP boost is active, any zIIPs added due to the boost processing are displayed as "B" instead of "I". The key at the end of the IEE174I display contains B BOOST (TRANSIENT) zIIP.

**Note:** When you issue the DISPLAY M=CPU command from a PR/SM partition, the system displays the status for the logical processors, and ICRFs defined to the partition.

**(*x*)**
A single processor identified by processor identifier in hexadecimal format.

**(*list*)**
One or more processor identifiers, each separated by a comma.

**Note:** When you issue the DISPLAY M=CPU command from a PR/SM partition, the system displays the status for the logical CPUs, and ICRFs defined to the partition.

**Note:** When you issue the DISPLAY M=CPU command from a system where PROCVIEW CORE is in effect, the command is rejected. With LOADxx PROCVIEW CORE,CPU_OK, CPU is accepted and treated as an alias for CORE.

**CU**
The system is to display the information for a specific control unit. For a description of the display format, see message IEE174I.

**(*xxxx*)**
The control unit number.

**Note:** The D M=CU command does not support displaying information for CTC control units.

**DEVICE *or* DEV**
The system is to display the number of online channel paths to devices (including special devices) or a single channel path to a single device.

For a description of the display format, see message IEE583I.

**([/]*devnum*)**
A single device number.

**([/]*lowdevnum*-[/]*highdevnum*)**
The lower device number *lowdevnum* and the upper device number *highdevnum* of a range of devices.

**([/]*devnum*,(*chp*))**
A single device number and single channel path identifier.

**(([/]*devnum*),*chp*)**
A single device number and single channel path identifier.

**ROUTE**
The ROUTE parameter displays the route through the fabric between the channel and the device.

Specify one of the following keywords on the ROUTE parameter:

**TODEV**
Displays the route through the fabric, starting with the channel and going to the device.

**FROMDEV**
Displays displays the route through the fabric, starting with the device and going to the channel.

**BOTH**
Displays the route through the fabric in both directions.

Routing and health information will only be determined and displayed when the channel is connected to a switch and the control unit definition for the channel path is defined in the I/O configuration with a two-byte link address.

For a description of the display format, see message IEE583I in *z/OS MVS System Messages, Vol 7 (IEB-IEE)*.

**HEALTH**
Displays the health information, which includes the utilization, average delay, and error counts, for the fabric, switch, and port.

**LINKINFO**
The LINKINFO parameter displays link diagnostic information for a device and CHPID. Link diagnostic information consists of the optical transceiver values, error counters, and buffer credits for each port from the channel to the control unit, except inter-switch link (ISL) ports.

- For switched point-to-point configurations, information for the channel port, entry switch port, exit switch port, and control unit port is displayed.
- For point-to-point configurations, information for the channel port and control unit port is displayed.

The LINKINFO parameter may only be specified when a path is specified on the D M=DEV command.

Specify one of the following keywords on the LINKINFO parameter:

**FIRST**
Displays the link diagnostic information that was obtained during IPL or when the path was varied online for the first time after IPL.

**LAST**
Displays the link diagnostic information that was last retrieved by the system. The system retrieves new information for a path every 24 hours or when you specify LINKINFO=REFRESH.

**REFRESH**
Requests that the system obtain new link diagnostic information for the physical path and then displays that information. This replaces the prior information; a subsequent LINKINFO=LAST request will display this new information.

**Notes:**

1. A REFRESH request does not cause the entry switch port, exit switch port, and control unit port to retrieve new optical transceiver information; it simply causes the last retrieved values to be returned to the channel subsystem. The frequency at which a port retrieves its own optical transceiver information is manufacturer- and model-specific.
2. The system rejects a REFRESH request if the channel specified in the command is already processing the maximum number of concurrent requests. These requests could be from this system or from other systems running on the same CPC. The allowed maximum number of concurrent requests for a channel is model-dependent.

The system issues the IEE584I message in response to the FIRST, LAST, and REFRESH keywords.

**COMPARE**
Displays a comparison of the first and last set of link diagnostic information that was retrieved by the system. The system issues the IEE586I message in response to the COMPARE keyword.

**ZHYPERLINK *or* ZHL**
The ZHYPERLINK parameter is used to display the zHyperLink capabilities of the device. If zHyperLink capability or zHyperLink reads or writes are disabled for the device, the list of reasons why the function is disabled is displayed. The ZHYPERLINK keyword is ignored if it is specified for a parallel access volume (PAV) alias device.

**READSEC**
The READSEC parameter is used to display the consistent read from secondary capability of the device. If consistent read from secondary is disabled for the device, the list of reasons why the function is disabled is displayed. The READSEC keyword is ignored if it is specified for a parallel access volume (PAV) alias device.

Device numbers and ranges can be specified in any combination.

A device number consists of 3, 4, or 5 hexadecimal digits, optionally preceded by a slash (/). A channel path identifier can have a value from 0 to FF. In the 5-digit format, *sdddd*, *s* is the subchannel set identifier and *dddd* is the device number.

If a range of device numbers is found and one of the two numbers is a 5-digit number, the other number in the range must also be a 5-digit number.

**HIGH**
The system is to display the highest possible central storage addresses. Each address indicates the amount of storage available at system initialization. For a description of the display format, see message IEE174I.

**HSA**
The system is to display the starting address and length of each portion of the hardware system area (HSA). For a description of the display format, see message IEE174I.

**SCM[(DETAIL)]**
Displays the online or offline status for all installed storage-class memory (SCM) increments, and usage information. Information about reconfigurability of online SCM is also displayed. If `DETAIL` is specified, details for each online increment are displayed; otherwise, summary information is displayed for ranges of SCM.

For a complete description of the display format of DISPLAY M=SCM, refer to message IEE174I.

**SIDE[(*id*)]**
The system is to display the resources installed in side (physical partition) *id*, whether the resources are online or offline, and whether the side is online, offline, or unavailable. If the processor complex is partitioned and the specified side is part of another configuration, no information is provided. If the processor complex is running in single-image mode and you do not specify an *id*, the system displays both sides. If the command is issued from MVS running in a partition, no information is provided.

For a complete description of the display format of DISPLAY M=SIDE, see message IEE174I.

**STORAGE *or* STOR**
The system is to display the status of central storage. The display includes storage offline, storage waiting to go offline, and reconfigurable storage sections. For storage waiting to go offline, the system displays:

- The address space identifier (ASID)
- The job name of the current user of the storage
- The amount of unassigned storage in offline storage elements
- The amount of storage that belongs to another configuration

STORAGE also indicates if a given range of central storage contains data that is shared through the use of the IARVSERV macro.

In this display, storage offline does not include the hardware save area (HSA). To find the location and length of the HSA, enter DISPLAY M=HSA.

If you do not specify (*dddddX*-*dddddX*), (*list*), or (E[=*id*]), the system displays the status of all central storage. For a description of the display format, see message IEE174I.

**(*dddddX*-*dddddX*)**
The starting and ending addresses of a range in central storage for which you want the status display. Specify up to five decimal digits followed by a multiplier (M-megabytes, G-gigabytes, T-terabytes, P-petabytes) for each address. The starting and ending addresses (*dddddX*) must each be on a valid storage boundary and cannot exceed 16383P. The starting and ending addresses must not be the same.

Instead of specifying the range using decimal numbers, you can specify it in hexadecimal, with or without a multiplier, in the format X'xxxxxx'-X'xxxxxx'. For example:

- X'123456789A00000'-X'123456789B00000'
- X'123'M-X'124'M

You can use underscores in any hexadecimal specification for better clarity. Underscores in the specification are ignored during processing.

**(*list*)**
One or more address ranges (in decimal), each separated by a comma.

**(E[=*id*])**
The system is to display the status of the requested storage element. The display includes the amount of storage (in megabytes) the system owns in each online storage element, the amount of storage available to be configured online, whether the storage element is online or offline. If you omit the *id*, the system displays this information for all installed storage elements.

**Note:** If the processor complex is partitioned and the specified storage element is part of another configuration, no information is provided.

**SWITCH(*ssss* [,*pp*[-*pp*] [,*pp*[-*pp*]]...])**
The system is to display the status of a specific switch, switch port, or list of switch ports.

For a description of the display format, see message IEE174I.

***ssss***
The device number of the switch device.

**[,*pp*[-*pp*] [,*pp*[-*pp*]]...]**
The port address or port address list.

**(parm[,*parm*]...)**
The system is to display the status of each resource you specify as *parm*. The list of *parm*s you specify within the parentheses may contain any combination of CHP, CPU, DEV, HIGH, HSA, STOR(E[=id]), and STOR. You must separate the resources in the list with commas and you must enclose the list in parentheses. Do not use blanks within the parentheses and do not specify CONFIG in the list.

**L=*a, name, or name-a***
Specifies the display area (*a*), console name (*name*), or both (*name-a*) where the display is to appear.

If you omit this operand, the display is presented in the first available display area or the message area of the console through which you enter the command.

## Examples

### Example 1:

To display the online or offline status of all devices on channel path 01, enter:

```
D M=CHP(01)
```

### Example 2:

To display the following information:

- The online or offline status of all processors
- The number of online channel paths to each device
- The highest central storage address available
- The status of central storage

enter the following command:

```
D M=(CPU,DEV,HIGH,STOR)
```

**Example 3:**

To display the number of megabytes of storage the system owns in storage element 0 and the status of the storage element, enter:

```
D M=STOR(E=0)
```

**Example 4:**

To display the number of megabytes of storage the system owns in each storage element and the status of each element, enter:

```
D M=STOR(E)
```

**Example 5:**

To display the status of all processors, the status for channel paths 1, 3, 4, 5, and the high storage addresses for central storage, enter:

```
D M=CPU
D M=CHP(01,03-05)
D M=HIGH
       or
D M=(CPU,CHP(01,03-05),HIGH)
```

**Example 6:**

The following example displays the status of cores. In this example, the configuration supports MT Mode=2 (MT=2) where standard CP cores 0 and 1 are exploiting MT Mode=1 (CP=1) and zIIP cores 2 and 3 are exploiting MT Mode=2 (zIIP=2).

```
D M=CORE
CORE STATUS: HD=Y  MT=2  MODE: CP=1  zIIP=2
ID    ST   ID RANGE   VP  ISCM  CPU THREAD STATUS
0000   +   0000-0001  H   FC00  +N
0001   +   0002-0003  H   FC00  +N
0002   +I  0004-0005  H   0200  ++
0003   +I  0006-0007  H   0200  ++
```

It is possible for a core status to be mixed (/). A core status of mixed means that a core's CPU thread status is unexpected given the MT Mode for cores of that type. In the following example, the status of core 3 is mixed, because CPU 6 is online, CPU 7 is offline, and zIIPs are exploiting MT Mode=2. With zIIPs exploiting MT Mode=2, the system expects core 3 to have both threads (CPUs 6 and 7) online. If a core appears with a mixed mode, it is generally due to an internal system error and should be configured to the desired online or offline state.

```
D M=CORE
CORE STATUS: HD=Y  MT=2  MODE: CP=1  zIIP=2
ID    ST   ID RANGE   VP  ISCM  CPU THREAD STATUS
0000   +   0000-0001  H   FC00  +N
0001   +   0002-0003  H   FC00  +N
0002   +I  0004-0005  H   0200  ++
0003   /I  0006-0007  H   0200  +-
```

**Example 7:**

The following example displays the channel path information for the control units connected to channel path identifier (CHPID) 5A.

```
D M=CHP(5A),PATHINFO
IEE588I 07.58.47 DISPLAY M 251
CHPID 5A:  TYPE=1B, DESC=FICON SWITCHED, ONLINE
Path Information for Channel Path 5A
Connection Security Capability: Encryption
Dest                                  Link-Speed  Conn
Link Intf Node Descriptor             Curr  Cap   Sec
1211 0131 002107.996.IBM.75.0000000KMP31 32G   32G   Encr
1111 0030 002107.996.IBM.75.0000000KMP31 16G   16G   Auth
```

## VERBEXIT IEAVBIPC subcommand — Format validated boot information

Specify the IEAVBIPC verb name on the VERBEXIT subcommand to format the Validated Boot for z/OS report. The report provides the following information after a validated boot IPL:

- Audit records that were created
- Certificate extracts that are being used
- Certificate extracts that were found not to be valid

For an enforce-mode IPL, no more than 1 audit record would be produced because any relevant issue would cause the system to enter a wait state right after building the audit record.

The messages within the IEAVBIPC report are the same as those used in the IEAVBPRT report. For details, see "IEAVBPRT: Validated boot print utility" in *z/OS MVS Diagnosis: Tools and Service Aids*.

### Syntax

```
VERBEXIT IEAVBIPC ['parameter']
```

### Parameters

The parameter can be one of the following values:

**SUMMARY**
Displays a summary report. This is the default.

**DETAIL**
Displays a detailed report.

### Examples

See the examples of the IEAVBPRT output in "IEAVBPRT: Validated boot print utility" in *z/OS MVS Diagnosis: Tools and Service Aids*.

## RACDCERT ADDTOKEN (Add token)

### Purpose

Use the RACDCERT ADDTOKEN command to create a new z/OS PKCS #11 token.

### Issuing options

The following table identifies the eligible options for issuing the RACDCERT ADDTOKEN command:

| As a RACF TSO command? | As a RACF operator command? | With command direction? | With automatic command direction? | From the RACF parameter library? |
|---|---|---|---|---|
| Yes | No | No. (See rules.) | No. (See rules.) | No |

**Rules:**

- The RACDCERT command cannot be directed to a remote system using the AT or ONLYAT keyword.
- The updates made to the RACF database by RACDCERT are eligible for propagation with automatic direction of application updates based on the RRSFDATA profiles AUTODIRECT.*target-node.*DIGTCERT.APPL and AUTODIRECT.*target-node.*DIGTRING.APPL, where *target-node* is the remote node to which the update is to be propagated.

## Authorization required

To issue the RACDCERT ADDTOKEN command, you must have sufficient authority to the appropriate resource in the CRYPTOZ class. (No authority to resources in the FACILITY class is required.) If you do not have authority to create the specified token as determined by ICSF, the command stops and an error message is displayed.

For example, if you want userid JOHN to create a token with the name MYTOKEN and manage the objects in this token, you can enter commands such as the following:

1. RDEFINE CRYPTOZ SO.MYTOKEN UACC(NONE)
2. PERMIT SO.MYTOKEN CLASS(CRYPTOZ) ID(JOHN) ACCESS(CONTROL)
3. RDEFINE CRYPTOZ USER.MYTOKEN UACC(NONE)
4. PERMIT USER.MYTOKEN CLASS(CRYPTOZ) ID(John) ACCESS(CONTROL)

When your installation controls access to ICSF services and the CSFSERV class is active, you must also have READ access to the CSF1TRC resource in the CSFSERV class.

For authorization details about the CRYPTOZ and CSFSERV classes, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

## Related commands

- To delete a token, see RACDCERT DELTOKEN.
- To list a token, see RACDCERT LISTTOKEN.

)X SYNTAX
RACDCERT ADDTOKEN

## Syntax

The complete syntax of the RACDCERT ADDTOKEN command is:

| RACDCERT ADDTOKEN(*token-name*) |
|---|

**Note:** The `ID(`*certificate-owner*`) | SITE | CERTAUTH` parameter is ignored for this RACDCERT function.

If you specify more than one RACDCERT function, only the last specified function is processed. Extraneous keywords that are not related to the function being performed are ignored.

If you do not specify a RACDCERT function, LIST is the default function.

For information on issuing this command as a RACF TSO command, see *z/OS Security Server RACF Command Language Reference*.

)O OPERANDS

## Parameters

**))ADDTOKEN(*token-name*)**
The *token-name* value is the name of the token being created. This token must not already exist. For token name rules, see the Tokens subsection in the Overview of z/OS support for PKCS #11 in *z/OS Cryptographic Services ICSF Writing PKCS #11 Applications*.

**Examples**

| Example | Activity label | Activity description |
|---|---|---|
| **1** | *Operation* | User RACFADM wants to create tokens for two servers that have existing RACF certificates. |
| | *Known* | User RACFADM has SPECIAL authority. The RACF certificate for each server already exists. |
| | *Commands* | `RACDCERT ADDTOKEN(ftpsrv.ftp.server.pkcs11.token)`<br>`RACDCERT ADDTOKEN(websrv.web.server.pkcs11.token)` |
| | *Output* | None. |

## RACDCERT GENCERT (Generate certificate)

### Purpose

Use the RACDCERT GENCERT command to create a digital certificate and potentially a public/private key pair.

### Processing details

When you specify an optional request data set containing the PKCS #10 request data, and extensions are present in the request data (not overridden by other keywords that are specified with the RACDCERT command), they are copied to the certificate being created. These extensions and the logic involved with using them are described in the following tables:

- For subjectKeyIdentifier, see Table 29 on page 74.
- For authorityKeyIdentifier, see Table 30 on page 74.
- For keyUsage, see Table 31 on page 75.
- For basicConstraints, see Table 32 on page 75.
- For subjectAltName, see Table 33 on page 75.
- For issuerAltName, see Table 34 on page 76.

*Table 29. Logic for the subjectKeyIdentifier extension for GENCERT*

| When the request data set is specified | When the request data set is not specified |
|---|---|
| The extension is encoded using the subjectKeyIdentifier value from the request data set if present, if not present the extension is encoded by generating the `keyIdentifier` according to the Public Key Infrastructure Standards. | The extension is encoded by generating the `keyIdentifier` according to Public Key Infrastructure Standards. |

*Table 30. Logic for the authorityKeyIdentifier extension for GENCERT*

| When SIGNWITH is specified | When SIGNWITH is not specified |
|---|---|
| The extension is encoded using the subjectKeyIdentifier value of the signing certificate if present, if not present the extension is not created. | The authorityKeyIdentifier extension is not created. |

*Table 31. Logic for the keyUsage extension for GENCERT*

| **Situation** | **keyUsage is present in the request data set** | **keyUsage is not present in the request data set** |
|---|---|---|
| When KEYUSAGE is specified and the target ID is CERTAUTH | If the certSign bit is turned off in the request data set, the request fails. Otherwise the extension is encoded as requested by the RACDCERT invoker. Additionally, the certSign and cRLSign bits are turned on if not already specified by the CERTSIGN keyword. | The extension is encoded as requested by the RACDCERT invoker. Additionally, the certSign and cRLSign bits are turned on. |
| When KEYUSAGE is specified and the target ID is SITE or ID(*cert-owner*) | The extension is encoded as requested by the RACDCERT invoker. | The extension is encoded as requested by the RACDCERT invoker. |
| When KEYUSAGE is not specified and the target ID is CERTAUTH | If the certSign bit is turned off this command fails, otherwise the extension is encoded as specified in the request data set. | The extension is encoded by turning the certSign and cRLSign bits on. |
| When KEYUSAGE is not specified and the target ID is SITE or ID(*cert-owner*) | The extension is encoded using the request data set values. | The keyUsage extension is not created. |

*Table 32. Logic for the basicConstraints extension for GENCERT*

| **Situation** | **basicConstraints is present in the request data set** | **basicConstraints is not present in the request data set** |
|---|---|---|
| **When the target ID is CERTAUTH** | If the cA boolean value is false, the command fails. Otherwise the extension is encoded turning the cA bit on. The pathLength value is not included. | The extension is encoded turning the cA bit on. The pathLength value is not included. |
| **When the target ID is SITE or ID(*cert-owner*)** | The extension is encoded using the request data set values, including the pathLength value. | The basicConstraints extension is not created. |

*Table 33. Logic for the subjectAltName extension for GENCERT*

| **Situation** | **subjectAltName is present in the request data set** | **subjectAltName is not present in the request data set** |
|---|---|---|
| **When ALTNAME is specified** | The extension is encoded as requested by the RACDCERT invoker. | The extension is encoded as requested by the RACDCERT invoker. |
| **When ALTNAME is not specified** | The extension is encoded using the request data set values. | The subjectAltName extension is not created. |

*Table 34. Logic for the issuerAltName extension for GENCERT*

| **When SIGNWITH is specified** | **When SIGNWITH is not specified** |
|---|---|
| The extension is encoded using the subjectAltName value of the signing certificate if the extension is present. Otherwise, the issuerAltName extension is not created. | The IssuerAltName extension is not created. |

## Issuing options

The following table identifies the eligible options for issuing the RACDCERT GENCERT command:

| **As a RACF TSO command?** | **As a RACF operator command?** | **With command direction?** | **With automatic command direction?** | **From the RACF parameter library?** |
|---|---|---|---|---|
| Yes | No | No. (See rules.) | No. (See rules.) | No |

**Rules:** The following rules apply when issuing this command.

- The RACDCERT command cannot be directed to a remote system using the AT or ONLYAT keyword.
- The updates made to the RACF database by RACDCERT are eligible for propagation with automatic direction of application updates based on the RRSFDATA profiles AUTODIRECT.*target-node.*DIGTCERT.APPL and AUTODIRECT.*target-node.*DIGTRING.APPL, where *target-node* is the remote node to which the update is to be propagated.

## Authorization required

To issue the RACDCERT GENCERT command, you must have the following authorizations:

- The SPECIAL attribute, or
- Sufficient authority to the IRR.DIGTCERT.ADD and IRR.DIGTCERT.GENCERT resource in the FACILITY class, based on the certificate owner and the SIGNWITH value, as shown in Table 35 on page 77, or
- Sufficient authority to the appropriate resources in the RDATALIB class, as shown in Table 36 on page 78, if Granular Authority Checking has been enabled by defining the IRR.RACDCERT.GRANULAR resource in the RDATALIB class.

When you specify the name of the request data set that contains the PKCS #10 request data, you must also have READ access to the specified data set.

When your installation controls access to ICSF services and the CSFSERV, CSFKEYS and CRYPTOZ classes are active, additional access to CSFSERV, CSFKEYS, and CRYPTOZ resources might be required as follows:

- When you specify RSA(PKDS), you must have READ authority to the CSFDSG, CSFDSV, CSFIQF, CSFOWH, CSFPKG, CSFPKRC, and CSFPKX resources.
- When you specify RSA(TOKEN(token-name)), you must have READ authority to the CSF1GAV, CSF1GKP, CSF1PKV, CSF1TRC, CSF1TRD, CSFDSG, CSFOWH, and CSFIQF resources.
- When you specify RSA (or omit key type) and omit PKDS and TOKEN, you must have READ authority to the CSFIQF resource.
- When you specify NISTECC or BPECC, you must have the following access authorities:
  - When you specify PKDS, you must have:
    - READ access to the CSFDSG, CSFDSV, CSFOWH, CSFPKG, CSFPKRC, and CSFPKX resources in the CSFSERV class.
    - READ access to key label resource in the CSFKEYS class. Note that if a system-generated key label is used, it starts with IRR.DIGTCERT.<*cert owner*>).

**Example:** If you want user ID JOHN to be able to create a certificate with a key stored in PKDS with a system-generated key label, you can enter commands such as the following:

1. RDEFINE CSFSERV CSF* UACC(NONE)
2. PERMIT CSF* CLASS(CSFSERV) ID(John) ACCESS(READ)
3. RDEFINE CSFKEYS IRR.DIGTCERT.JOHN.* UACC(NONE)
4. PERMIT IRR.DIGTCERT.JOHN.* CLASS(CSFKEYS) ID(JOHN) ACCESS(READ)

– When you specify TOKEN, you must have:

  - READ access to the CSF1GAV, CSF1GKP, CSF1PKV, CSF1TRC, CSF1TRD, CSFDSG, and CSFOWH resources in the CSFSERV class.
  - UPDATE or CONTROL access to the token resource with format USER.*<token name>* in the CRYPTOZ class

**Example:** If you want user ID JOHN to be able to create a certificate with a key that is stored in an existing token called MYTOKEN in TKDS, you can create profiles in CSFSERV and CRYPTOZ, as follows:

1. RDEFINE CSFSERV CSF* UACC(NONE)
2. PERMIT CSF* CLASS(CSFSERV) ID(John) ACCESS(READ)
3. RDEFINE CRYPTOZ USER.MYTOKEN UACC(NONE)
4. PERMIT USER.MYTOKEN CLASS(CRYPTOZ) ID(John) ACCESS(CONTROL)

– When you omit PKDS and TOKEN, you must have READ access to the CSF1GAV, CSF1GKP, CSF1PKS, CSF1PKV, CSF1TRC, CSF1TRD, and CSFOWH resources.

- When you specify ICSF, you must have READ authority to the CSFIQF, CSFPKI, and CSFPKRC resources.
- When you specify FROMICSF, you must have READ authority to the CSFIQF and CSFPKX resources.
- When you specify SIGNWITH, you must have the following access authorities:

  – If the private key of the signing certificate is an ECC key that is stored in the RACF database, you must have READ authority to the CSF1PKS, CSF1PKV, CSF1TRC, CSF1TRD, and CSFOWH resources.

  – If the private key of the signing certificate is stored in the ICSF PKA key data set (PKDS) or in the ICSF Token Data Set (TKDS), you require additional access based on the key type, as follows:

    - When the key is an RSA type, you must have READ authority to the CSFDSG resource.
    - When the key is an ECC type, you must have READ authority to the CSF1PKV, CSF1TRC, CSF1TRD, CSFDSG, and CSFOWH resources.

For details about the CSFSERV resources, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

**Important:** The GENCERT function allows a user to generate and sign a certificate. Carefully consider which users are authorized to use GENCERT, which user ID is associated with the generated certificate, and which certificate is used to sign the generated certificate.

*Table 35. Authority required for the RACDCERT GENCERT function under the FACILITY class*

| SIGNWITH | Your own certificate | Another user's certificate | SITE or CERTAUTH certificate |
|---|---|---|---|
| SIGNWITH your own certificate | READ authority to IRR.DIGTCERT. ADD and READ authority to IRR.DIGTCERT. GENCERT | UPDATE authority to IRR.DIGTCERT. ADD and READ authority to IRR.DIGTCERT. GENCERT | CONTROL authority to IRR.DIGTCERT. ADD and READ authority to IRR.DIGTCERT. GENCERT |
| SIGNWITH a SITE or CERTAUTH certificate | READ authority to IRR.DIGTCERT. ADD and CONTROL authority to IRR.DIGTCERT. GENCERT | UPDATE authority to IRR.DIGTCERT. ADD and CONTROL authority to IRR.DIGTCERT. GENCERT | CONTROL authority to IRR.DIGTCERT. ADD and CONTROL authority to IRR.DIGTCERT. GENCERT |

*Table 35. Authority required for the RACDCERT GENCERT function under the FACILITY class (continued)*

| SIGNWITH | Your own certificate | Another user's certificate | SITE or CERTAUTH certificate |
|---|---|---|---|
| SIGNWITH not specified | READ authority to IRR.DIGTCERT. ADD and READ authority to IRR.DIGTCERT. GENCERT | UPDATE authority to IRR.DIGTCERT. ADD and UPDATE authority to IRR.DIGTCERT. GENCERT | CONTROL authority to IRR.DIGTCERT. ADD and CONTROL authority to IRR.DIGTCERT. GENCERT |

*Table 36. Authority required for the RACDCERT GENCERT function under the RDATALIB class when IRR.RACDCERT.GRANULAR is defined*

| READ access to the resource based on cert owner and cert label * | Purpose |
|---|---|
| IRR.DIGTCERT.<cert owner>.<cert label>.UPD.GENCERT | Create a certificate under <cert owner> with specified <cert label><br><br>Create a certificate under <cert owner> with specified <cert label> |
| IRR.DIGTCERT.<cert owner>.LABEL*.UPD.GENCERT | Create a certificate under <cert owner> with no label specified.<br><br>Create a certificate under <cert owner> with no label specified. |
| IRR.DIGTCERT.<cert owner>.<cert label>.UPD.GENCERT and IRR.DIGTCERT.<signer ID>.<signer cert label> .UPD.GENCERT | Create a certificate under <cert owner> with specified <cert label> signed by <signer cert label> owned by <signer ID><br><br>Create a certificate under <cert owner> with specified <cert label> signed by <signer cert label> owned by <signer ID> |

***** 'cert owner' is the RACF user ID, or CERTIFAUTH (for CERTAUTH), or SITECERTIF (for SITE); 'signer ID' is CERTIFAUTH or SITECERTIF.

**Authority processing details under the Facility class:** RACF performs two checks that determine the authority that is required for the GENCERT command:

1. How the certificate is being signed, specified with the SIGNWITH keyword.

Users with SPECIAL authority can use the SIGNWITH keyword with any value. Users without SPECIAL authority must have authority to the IRR.DIGTCERT.GENCERT resource in the FACILITY class. If SIGNWITH is specified without the CERTAUTH or SITE keyword, the certificate is signed with the certificate that is identified with the LABEL keyword for the user who is issuing the RACDCERT command. This requires READ access to the resource IRR.DIGTCERT.GENCERT in the FACILITY class. If either SIGNWITH(CERTAUTH...) or SIGNWITH(SITE) is specified, CONTROL authority is required to the resource IRR.DIGTCERT.GENCERT in the FACILITY class.

Not specifying SIGNWITH indicates that the certificate is to be self-signed. The signing key is owned by the certificate itself. Thus the authority that is needed for signing is determined by the owner of the generated certificate.

2. What type of certificate is being generated, which is specified with the `ID()`, SITE or CERTAUTH keywords.

   Users with SPECIAL authority can generate a digital certificate for any RACF-defined user or for any certificate-authority or site certificate. Users without SPECIAL authority can generate certificate authority or site certificates if they have CONTROL authority to the resource IRR.DIGTCERT.ADD in the FACILITY class. Users without SPECIAL authority can generate certificates for other users if they have UPDATE authority to the resource IRR.DIGTCERT.ADD in the FACILITY class. Users without SPECIAL authority can generate certificates for themselves if they have READ authority to the resource IRR.DIGTCERT.ADD in the FACILITY class.

## Activating your changes

If the DIGTCERT class is RACLISTed, refresh the class to activate your changes.

**Example**:

```
SETROPTS RACLIST(DIGTCERT) REFRESH
```

## Syntax

The complete syntax of the RACDCERT GENCERT command is:

```
    RACDCERT GENCERT [ (request-data-set-name) ]

        [ ID(certificate-owner) | SITE | CERTAUTH ]
        [ SUBJECTSDN(
              [ CN('common-name') ]
              [ T('title') ]
              [ OU('organizational-unit-name1' [  , 'organizational-unit-name2', ...] ) ]

              [ O('organization-name') ]
              [ L('locality') ]
              [ SP('state-or-province') ]
              [ C('country') ]
               ) ]
        [ SIZE(key-size) ]
        [ NOTBEFORE( [ DATE(yyyy-mm-dd) ] [ TIME(hh:mm:ss) ] ) ]
        [ NOTAFTER( [ DATE(yyyy-mm-dd) ] [ TIME(hh:mm:ss) ] ) ]
        [ WITHLABEL('label-name') ]
        [ SIGNWITH( [ CERTAUTH | SITE ] LABEL('label-name') ) ]
    [ { RSA [ (PKDS [ (pkds-label / * ) ] | TOKEN(token-name) ) ]
            | NISTECC [ (PKDS [ (pkds-label / * ) ] | TOKEN(token-name)  ) ]
            | BPECC [ (PKDS [ (pkds-label / * ) ]  |  TOKEN(token-name)  ) ]
            | DSA
            | FROMICSF(pkds-label) }
]
 [ SIGATTR [ (RSAPSS) ] ]
            [ KEYUSAGE(
                [ CERTSIGN ]
                [ DATAENCRYPT ]
                [ DOCSIGN ]
                 [ HANDSHAKE ]
                [ KEYAGREE ]
              ) ]
        [ ALTNAME(
            [ IP(numeric-IP-address) ]
            [ DOMAIN('internet-domain-name') ]
            [ EMAIL('email-address') ]
            [ URI('universal-resource-identifier') ]
            )
 ]
```

If you specify more than one RACDCERT function, only the last specified function is processed. Extraneous keywords that are not related to the function being performed are ignored.

If you do not specify a RACDCERT function, LIST is the default function.

)O OPERANDS

## Parameters

**))GENCERT**
**GENCERT(*request-data-set-name*)**
*Request-data-set-name* is the name of an optional data set that contains the PKCS #10 certificate request data. The request data contains the user's generated public key and X.509 distinguished name. The request data must be signed, DER-encoded, and then Base64 encoded according to the PKCS #10 standard.

The subkeywords of the GENCERT function specify the information that is to be contained within the certificate that is being created.

*Request-data-set-name* has characteristics (for example, RECFM) identical to the data set that can be specified with the ADD and CHECKCERT keywords. If *request-data-set-name* is specified, SIGNWITH must also be specified because the *request-data-set-name* data set does not contain a private key. If SIGNWITH is not specified, an informational message is issued. Note that the issuer of the RACDCERT command must have READ access to the *request-data-set-name* data set to prevent an authorization abend from occurring when the data set is read.

**When GENCERT is issued with a request data set:** The following conditions apply:

- No key-pair is generated. This is because the request data set contains the user's public key.
- The public key from the request data set is used in the generated certificate.
- If FROMICSF is specified, the GENCERT command fails.
- If the RSA, NISTECC, BPECC, or DSA keyword is specified, it is ignored.
- If the RSA(PKDS), NISTECC(PKDS), or BPECC(PKDS) keyword is specified, it is ignored *unless* one of the following conditions is true:
  - The certificate profile (containing the private key of the corresponding public key) in the request data set exists *and* the private key is not yet stored in the PKDS. When this occurs, RACF stores the private key in the ICSF PKDS.
  - There is no corresponding private key profile *and* you specified a PKDS label value. When this occurs, RACF stores the public key in the ICSF PKDS.

**))ID(*certificate-owner*) | SITE | CERTAUTH**
Specifies that the new certificate that is associated is either a user certificate with the specified user ID, a site certificate, or a certificate-authority certificate. If you do not specify ID, SITE, or CERTAUTH, the default is ID, and *certificate-owner* defaults to the user ID of the command issuer. If more than one keyword is specified, the last specified keyword is processed, and the others are ignored by TSO command parse processing.

**))SUBJECTSDN**
Specifies the subject's X.509 distinguished name, which consists of the following components:

- **CommonName** - specified with the CN subkeyword.
- **Title**—specified with the T subkeyword.
- **Organizational Unit**—specified with the OU subkeyword. Multiple values can be specified for the organizational unit.
- **Organization**—specified with the O subkeyword.
- **Locality**—specified with the L subkeyword.
- **State/Province**—specified with the SP subkeyword.
- **Country**—specified with the C subkeyword.

SUBJECTSDN completely overrides the values that are contained in the certificate request in the data set specified with the GENCERT function.

The length of the value you specify for each component of the SUBJECTSDN is limited to 64 characters. Each SUBJECTSDN subkeyword can be specified only once. The total length of the subject's distinguished name is limited to 1024 characters, including the X.509 identifiers (such as C= and CN=) and the dot qualifiers.

If the SUBJECTSDN name is too long, an informational message is issued, and the certificate is not added.

Any printable character that can be mapped to an ASCII character can be specified. Characters that cannot be mapped, such as X'4A' (¢) and X'00' are shown by RACDCERT LIST as blanks.

If SUBJECTSDN and *request-data-set-name* are not specified, the programmer name data from the ID() user (either specified or defaulted), or the programmer name from the SITE or CERTAUTH anchor user IDs (`irrsitec` or `irrcerta`) is used as the common name (CN). If the programmer name is all blanks (X'40'), nulls (X'00'), # characters (X'7B'), or X'FF' characters, the common name is set to the user ID that is to be associated with this certificate.

**))SIZE(*key-size*)**

Specifies the size of the private key expressed in decimal bits. This keyword is ignored if GENCERT is specified with *request-data-set-name*.

If SIZE is not specified, it defaults to 2048 for RSA and DSA keys, or 192 for NISTECC and BPECC keys.

For NISTECC keys, valid key sizes are 192, 224, 256, 384, and 521 bits. For BPECC keys, valid key sizes are 160, 192, 224, 256, 320, 384, and 512 bits.

For DSA keys, the minimum key size is 512.

For RSA keys, the minimum key size for clear keys and secure keys in the PKDS (PKA key data set) is 512; the minimum key size for secure keys in the TKDS (token key data set) is 1024 and the size must be a multiple of 256.

- The maximum key size for RSA and DSA keys is determined by United States export regulations and is controlled by RACF and non-RACF code in z/OS. Depending on the installation, non-RACF code may enforce a lower maximum size.
- Rounding up to the next appropriate key size might occur. Therefore, the key size of the generated key might be longer than the value you specify with SIZE but the generated key is never shorter than requested.

**Maximum key sizes:** The maximum key size for a private key depends on key type, as follows:

| Private key type | Maximum key size |
|---|---|
| RSA key stored in the RACF database | 4096 bits |
| RSA key stored in the ICSF TKDS as secure key | 4096 bits |
| RSA key stored in the ICSF PKDS as a CRT key token | 4096 bits |
| DSA key | 2048 bits |
| RSA key stored in the ICSF PKDS as an ME key token | 1024 bits |
| NISTECC key | 521 bits |
| BPECC key | 512 bits |

**Note:** To generate an RSA key that is longer than 1024 bits and is to be stored in the RACF database, the CP Assist for Cryptographic Function (CPACF) must be enabled.

**Standard key sizes:** Currently, standard sizes for RSA keys are as follows:

| Key size | Key strength |
|---|---|
| 512 bits | Low-strength key |
| 1024 bits | Medium-strength key |
| 2048 bits | High-strength key |
| 4096 bits | Very high-strength key |

**Key strength considerations:** Shorter keys of the ECC type, which are generated when you specify NISTECC or BPECC, achieve comparable key strengths when compared with longer RSA keys.

RSA, NISTECC, and BPECC keys of the following sizes are comparable in strength:

| RSA key size | NISTECC key size | BPECC key size |
|---|---|---|
| 1024 bits | 192 bits | 160 or 192 bits |
| 2048 bits | 224 bits | 224 bits |
| 3072 bits | 256 bits | 256 or 320 bits |
| 7680 bits | 384 bits | 384 bits |
| 15360 bits | 521 bits | 512 bits |

**Hashing algorithm used for signing:** RACF signs certificates using a set of secure hash algorithms based on the SHA-1 or SHA-2 hash functions. The size of the signing key determines the hashing algorithm used for signing, as follows:

| Hashing algorithm used for signing | Signing key size | | | |
|---|---|---|---|---|
| | RSA / DSA | RSAPSS for RSA signing key | NISTECC | BPECC |
| **SHA-1** | Less than 2048 bits | — | — | — |
| **SHA-256** | 2048 bits or longer | 2048 bits or longer and less than 3072 bits | 192, 224, or 256 bits | 160, 192, 224, 256, or 320 bits |
| **SHA-384** | — | 3072 bits or longer and less than 4096 bits | 384 bits | 384 bits |
| **SHA-512** | — | 4096 bits | 521 bits | 512 bits |

**))NOTBEFORE(DATE(*yyyy-mm-dd*) TIME(*hh:mm:ss*))**

Specifies the local date and time from which the certificate is valid. If DATE is not specified, it defaults to the current local date. If TIME is not specified, it defaults to TIME(00:00:00).

If DATE is specified, the value of *yyyy* must be 1950 - 9997.

Note that the use of the date format *yyyy-mm-dd* is valid. However, to aid installations familiar with the RACF date format, the value can be specified in the format *yyyy/mm/dd*.

The time and date values are stored in the certificate as a universal time coordinated (UTC) value. The calculated UTC value might be incorrect if the date and time values for NOTBEFORE and NOTAFTER represent a time that has a different local offset from UTC.

**))NOTAFTER(DATE(*yyyy-mm-dd*) TIME(*hh:mm:ss*))**

Specifies the local date and time after which the certificate is no longer valid. If DATE is not specified, it defaults to one year from the NOTBEFORE date value. If TIME is not specified, it defaults to TIME(23:59:59).

If DATE is specified, the value of *yyyy* must be 1950 - 9997. If DATE is defaulted, the value must be 1951 - 9998.

The NOTBEFORE value must be earlier than the NOTAFTER value or an informational message is issued.

Note the use of the date format *yyyy-mm-dd* is valid. However, to aid installations familiar with the RACF date format, the value can be specified as *yyyy/mm/dd*.

The time and date values are stored in the certificate as a universal time coordinated (UTC) value. The calculated UTC value might be incorrect if the date and time values for NOTBEFORE and NOTAFTER represent a time that has a different local offset from UTC.

**))WITHLABEL('*label-name*')**

Specifies the label assigned to this certificate. If specified, this must be unique to the user ID with which the certificate is associated. If not specified, it defaults in the same manner as the WITHLABEL keyword on the RACDCERT ADD command.

The *label-name* is stripped of leading and trailing blanks. If a single quotation mark is intended to be part of the *label-name*, use two single quotation marks together for each single quotation mark within the string, and enclose the entire string within single quotation marks.

See the WITHLABEL keyword for RACDCERT ADD for information on label rules.

**))SIGNWITH**
**SIGNWITH(CERTAUTH LABEL('*label-name*'))**
**SIGNWITH(SITE LABEL('*label-name*'))**
**SIGNWITH(LABEL('*label-name*'))**

Specifies the certificate with a private key that is signing the certificate. If not specified, the default is to sign the certificate with the private key of the certificate that is being generated. This creates a self-signed certificate. The signing certificate must belong to the user ID running the command, or SITE or CERTAUTH. If SITE and CERTAUTH keywords are omitted, the signing certificate owner defaults to the user ID of the command issuer.

If SIGNWITH is specified, it must refer to a certificate that has a private key associated with it. If no private key is associated with the certificate, an informational message is issued and processing stops.

If you specify either *request-data-set-name* or FROMICSF, you must specify SIGNWITH.

Note that self-signed certificates are always trusted, while all other certificates are created with the trust status of the certificate specified in the SIGNWITH keyword. If the certificate specified in the SIGNWITH keyword is not trusted, an informational message is issued but the certificate is still generated.

**Note:** Prior to z/OS V2R4, the RSA private key was stored in the PKDS under the RSA master key. This key cannot be used to sign with the RSAPSS algorithm without converting it to be an ECC master key protected key. You can use this tool to convert it: Translate and replace an RSA key for RSA PSS (community.ibm.com/community/user/ibmz-and-linuxone/blogs/bob-petti1/2021/03/10/translate-and-replace-an-rsa-key-for-rsa-pss?CommunityKey=6593e27b-caf6-4f6c-a8a8-10b62a02509c&tab=recentcommunityblogsdashboard).

**))RSA | PCICC | ICSF | DSA | NISTECC | BPECC | FROMICSF**

Specifies if RACF should generate a new key pair, and if so, how to generate the key pair and where to store the private key for future use. The default action for a new key is to store it as a software key. If no keyword is specified, the key pair is generated using software with the RSA algorithm and the private key is stored in the RACF database as an RSA key.

**Guidelines:**

The PCICC and ICSF keywords are deprecated. IBM discourages the use of these parameters.

The following guidelines apply when choosing key options:

- Choose RSA(PKDS), ICSF, PCICC or RSA(TOKEN) when the private key to be generated is an RSA type and you need hardware protection for the key.
  - The RSA(PKDS) keyword is equivalent to the PCICC keyword and stores the key as an RSA Chinese Remainder Theorem (CRT) key token. RACDCERT LIST will display this key with key type RSA along with a PKDS label.
  - The ICSF keyword stores the key as an RSA Modulus-Exponent (ME) key token. RACDCERT LIST will display this key with key type RSA Mod-Exp along with a PKDS label.
- Specify RSA when the key to be generated is an RSA type but no hardware protection is needed. Such software keys can be up to 4096 bits in size.
- Choose NISTECC or BPECC when the key to be generated is an ECC type.

- Specify NISTECC(PKDS), BPECC(PKDS),NISTECC(TOKEN) or BPECC(TOKEN) when hardware protection is needed.
- Choose DSA when the key to be generated is a DSA type. Note that no hardware protection is available for DSA keys.

**When you issue GENCERT with a request data set:** If the certificate you are generating is associated with a public or private key that is already stored in the PKDS, the following restriction applies:

- **Restriction:** Respecifying the PKDS label with the RSA(PKDS), ICSF, PCICC, NISTECC(PKDS), or BPECC(PKDS) keyword does *not* change the existing PKDS label or key type. For example:
  - If the private key already exists in the PKDS as an RSA Modulus-Exponent (ME) key token, specifying RSA(PKDS) or PCICC does not convert the key to an RSA Chinese Remainder Theorem (CRT) key token.
  - If the private key already exists in the PKDS as an RSA Chinese Remainder Theorem (CRT) key token, specifying ICSF does not convert the key to an RSA Modulus-Exponent (ME) key token.

"PKDS label considerations" in *z/OS Security Server RACF Command Language Reference*.

"PKDS hardware requirements" in *z/OS Security Server RACF Command Language Reference*.

**RSA**

Specifies that the key pair is to be generated using software with the RSA algorithm and the private key is to be stored in the RACF database as an RSA key. RSA is the default key type.

When you specify RSA without the PKDS option or accept RSA as the default key type, the CP Assist for Cryptographic Function (CPACF) must be enabled to generate a key that is longer than 1024 bits.

**PKDS[(*pkds-label* / *)]**

Specifies that the key pair is to be generated using a a CCA cryptographic coprocessor. The resulting private key is stored in the ICSF PKA key data set (PKDS) as an RSA Chinese Remainder Theorem (CRT) key token with either a system-generated label, a label specfied by pkds-label, or a label copied from the certificate label.

**TOKEN (*token-name*)**

Specifies that the key pair is to be generated using an Enterprise PKCS#11 cryptographic coprocessor. The resulting private key is stored in the specified existing token-name token in the ICSF token key data set (TKDS) as an RSA Chinese Remainder Theorem (CRT) key token.

**PCICC[(*pkds-label* | *)]**

This parameter is deprecated. IBM recommends that you use `RSA(PKDS[pkds-label | *])` instead of `PCICC[(pkds-label | *)]`.

It specifies the same function as the PKDS suboperand of the RSA operand. See the RSA operand of GENCERT for details.

**ICSF[(*pkds-label* | *)]**

This parameter is deprecated. IBM discourages the use of this parameter, as it is only applicable to RSA keys that are limited to 1024 bits.

It specifies that the key pair is to be generated using software. The resulting private key is generated with the RSA algorithm and stored in the ICSF PKA key data set (PKDS) as an RSA Modulus-Exponent (ME) key token.

**DSA**

Specifies that the key pair is to be generated using software with the DSA algorithm. The resulting private key is stored in the RACF database as a DSA key. **Note:** DSA key generation can be very slow, especially for keys longer than 1024 bits.

**NISTECC**

Specifies that the key pair is to be generated using software if clear key is not restricted in the system, with the elliptic curve cryptography (ECC) algorithm in accordance with the standard proposed by the National Institute of Standards and Technology (NIST). The resulting private key is stored in the RACF database as an ECC key.

When specifying NISTECC, the ICSF subsystem must be operational and configured for PKCS #11 operations.

**PKDS[(*pkds-label* | *)]**
Specifies that the key pair is to be generated using a CCA cryptographic coprocessor. The resulting private key is stored in the ICSF PKA data set (PKDS) in the PKA token with either a system-generated label, a label specified by *pkds-label*, or a label copied from the certificate label.

**TOKEN (*token-name*)**
Specifies that the key pair is to be generated using an Enterprise PKCS#11 cryptographic coprocessor. The resulting private key is stored in the specified existing token-name token in the ICSF token key data set (TKDS) as an RSA Chinese Remainder Theorem (CRT) key token.

**BPECC**
Specifies that the key pair is to be generated using software, if clear key is not restricted in the system, with the elliptic curve cryptography (ECC) algorithm in accordance with the standard proposed by the ECC Brainpool working group of the Internet Engineering Task Force (IETF). The resulting private key is stored in the RACF database as an ECC key.

When specifying BPECC, the ICSF subsystem must be operational and configured for PKCS #11 operations.

**Restriction:** When ICSF is operating in FIPS mode, you cannot generate a Brainpool ECC private key.

**PKDS[(*pkds-label* | *)]**
Specifies that the key pair is to be generated using a CCA cryptographic coprocessor. The resulting private key is stored in the ICSF PKA data set (PKDS) as an ECC key in the PKA token with either a system-generated label, a label specified by *pkds-label*, or a label copied from the certificate label.

**TOKEN (*token-name*)**
Specifies that the key pair is to be generated using an Enterprise PKCS#11 cryptographic coprocessor. The resulting private key is stored in the specified existing token-name token in the ICSF token key data set (TKDS) as an RSA Chinese Remainder Theorem (CRT) key token.

**FROMICSF(*pkds-label*)**
Specifies that no new key pair is to be generated for this new certificate. Instead, RACF uses an existing public key specified by its PKDS label. The public key must reside in the ICSF PKA key data set (PKDS).

When you specify FROMICSF, you must also specify SIGNWITH to sign the new certificate with an existing certificate. The new certificate will contain no private key and therefore cannot be self-signed.

You cannot specify both *request-data-set-name* and FROMICSF.

**SIGATTR(*attribute*)**
Specifies the signing attribute if the signing key is an RSA key. The only valid value for *attribute* is RSAPSS. If *attribute* is not specified, the default value is RSAPSS.

***))*KEYUSAGE**
Specifies the appropriate values for the keyUsage certificate extension, of which one or more of the values might be coded. For certificate authority certificates, the default is CERTSIGN and is *always* set. There is no default for certificates that are not certificate-authority certificates.

**HANDSHAKE**
Facilitates identification and key exchange during security handshakes, such as SSL, which set the digitalSignature and keyEncipherment indicators if the key algorithm is RSA. If key type is DSA, NISTECC, or BPECC, this usage sets only the digitalSignature indicator.

**DATAENCRYPT**
Encrypts data, which sets the dataEncipherment indicator. This usage is not valid for DSA, NISTECC, or BPECC keys.

**DOCSIGN**
Specifies a legally binding signature, which sets the nonRepudiation indicator.

**CERTSIGN**
Specifies a signature for other digital certificates and CRLs, which sets the keyCertSign and cRLSign indicators.

**KEYAGREE**
Facilitates key exchange, which sets the keyAgreement indicator. This usage is valid only for NISTECC and BPECC keys.

A certificate with no keyUsage value other than keyAgreement cannot be used for signing.

***))*ALTNAME**
Specifies the appropriate values for the subjectAltName extension, of which one or more of the values might be coded. If required for the extension, RACF converts the entered values to ASCII.

**Note:** RACF assumes the terminal code page is IBM-1047 and translates to ASCII accordingly.

**IP(*numeric-IP-address*)**
Specifies a fully qualified numeric IP address in IPv4 or IPv6 form.

IPv4 dotted decimal form consists of four decimal numbers (each number must be a value from 0 - 255) separated by periods:

**Example:** `9.117.2.45`

IPv6 form consists of eight 16-bit blocks of hexadecimal characters separated by colons:

**Example:** `ABCD:EF01:2345:6789:ABCD:EF01:2345:6789`

In IPv6 form, leading zeros in each block are optional. Successive blocks of zeros can be represented by a single occurrence of `::`.

**Example:** `2001:DB8::8:800:200C:417A`

An IPv6 address can contain an IPv4 address:

**Example:** `0:0:0:0:0:ABCD:1.2.3.4`

**DOMAIN(*'internet-domain-name'*)**
Specifies a quoted string containing a fully qualified *'internet-domain-name'* (such as `'www.widgits.com'`). RACF does not check this value's validity.

**EMAIL(*'email-address'*)**
Specifies a quoted string containing a fully qualified *'email-address'*, such as `'jasper at moes.bar.com'`. RACF replaces the word `at` with the @ symbol (X'7C') to conform with RFC822. If RACF cannot locate the word `at` it assumes the address is already in RFC822 form and makes no attempt to alter it other than converting it to ASCII.

**URI(*'universal-resource-identifier'*)**
Specifies the *'universal-resource-identifier'* (such as `'http://www.widgits.com'`). RACF does not check the validity of this value.

## Examples

| Example | Activity label | Description |
|---|---|---|
| **1** | *Operation* | User RACFADM requests the creation of a certificate-authority certificate, with values for the subjectAltName extension and the keyUsage extension, for the local certificate authority. The certificate will be self-signed and a SIGNWITH value need not be specified. |
| | *Known* | User RACFADM has CONTROL access authority to the IRR.DIGTCERT.* resource in the FACILITY class. |
| | *Command* | `RACDCERT GENCERT`<br>`  CERTAUTH`<br>`  SUBJECTSDN(CN('Local CA'))`<br>`  ALTNAME(IP(9.117.170.150) DOMAIN('www.widgits.com')`<br>`  EMAIL('localca@www.widgits.com')`<br>`  URI('http://www.widgits.com/welcome.html'))`<br>`  KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN CERTSIGN)`<br>`  WITHLABEL('Local PKIX CA')` |
| | *Output* | None. |
| **2** | *Operation* | User WENTING wants to create a new certificate with a 2048-bit public/ private key pair so she can share encrypted data with a business partner. She wants to call her certificate `Wen Ting's certificate`. |
| | *Known* | IBM Encryption Facility requires a PKDS label. RACF generates a default PKDS label when no value is specified with the PKDS keyword. |
| | *Command* | `RACDCERT GENCERT`<br>`  SUBJECTSDN(CN('Wen Ting''s certificate'))`<br>`  WITHLABEL('Wen Ting''s certificate')`<br>`  SIZE(2048)`<br>`  RSA(PKDS)`<br>`  NOTAFTER(DATE(2030/10/10))` |
| | *Output* | None. |
| **3** | *Operation* | User RACFADM wants to create a CA certificate that can be used to issue code-signing certificates for users who need to digitally sign programs. |
| | *Known* | User RACFADM has CONTROL access authority to the IRR.DIGTCERT.* resource in the FACILITY class, and appropriate authority in the CSFSERV and CSFKEYS classes to be able to use the PKDS option. |
| | *Command* | `RACDCERT CERTAUTH GENCERT`<br>`  SUBJECTSDN(OU('MyCompany Code Signing CA') O('MyCompany')`<br>`C('US'))`<br>`  SIZE(2048) RSA(PKDS) WITHLABEL('MyCompany Code Signing CA')` |
| | *Output* | None. |
| **4** | *Operation* | User RACFADM wants to issue a code-signing certificate to user RAMOS who needs to digitally sign programs. The new code-signing certificate will be signed by the CA certificate created in Example 3. |
| | *Known* | User RACFADM has CONTROL access authority to the IRR.DIGTCERT.* resource in the FACILITY class. |
| | *Command* | `RACDCERT ID(RAMOS) GENCERT`<br>`  SUBJECTSDN(CN('Ramos Code Signing Cert') O('MyCompany') C('US'))`<br>`  SIZE(1024) WITHLABEL('Ramos Code Signing Cert')`<br>`  SIGNWITH(CERTAUTH LABEL('MyCompany Code Signing CA'))`<br>`  KEYUSAGE(HANDSHAKE DOCSIGN)` |
| | *Output* | None. |

| Example | Activity label | Description |
|---|---|---|
| **5** | *Operation* | User ANNA wants to create a new certificate with an ECC private key. The new certificate will be called `Anna's certificate`. The key requires hardware protection so she will store it in the ICSF PKDS. |
| | *Known* | User ANNA has sufficient authority to the appropriate resources in the FACILITY and CSFSERV classes. The system contains an operational ICSF subsystem and Crypto Express3 coprocessor (CEX3C). |
| | *Command* | `RACDCERT GENCERT`<br>`  SUBJECTSDN(CN('COMPANY A'))`<br>`  WITHLABEL('Anna''s certificate')`<br>`  BPECC(PKDS(ECCKEY4ANNASCERTIFICATE))` |
| | *Output* | None. |
| **6** | *Operation* | User CLAUSEN wants to create a new certificate with an RSA private key. The new certificate will be called `Christine Clausen's certificate`. The key requires secure hardware protection so she will create the key in the ICSF TKDS. |
| | *Known* | User CLAUSEN has sufficient authority to the appropriate resources in the FACILITY, CRYPTOZ and CSFSERV classes. The token labelled COMPANYA.TOKEN has been defined. The system contains an operational ICSF subsystem and Crypto Express4 coprocessor (CEX4X). |
| | *Command* | `RACDCERT GENCERT`<br>`    SUBJECTSDN(CN('COMPANY A'))`<br>`    WITHLABEL('Christine Clausen''s certificate')`<br>`    RSA(TOKEN(COMPANYA.TOKEN))` |
| | *Output* | None |

# Chapter 8. Auditing and monitoring

The system records information about Validated Boot for z/OS activity in SMF type 90 subtype 42 records.

## Subtype 42 — Validated Boot for z/OS configuration event

An SMF type 90 subtype 42 record is generated at IPL time during Validated Boot for z/OS processing. The record contains:

- Certificate extract information
- Audited validation failures
- An indication of whether this validated boot was in audit mode or enforce mode

### Header/self-defining section

| Offsets | | Name | Length | Format | Description |
|---|---|---|---|---|---|
| 0 | 0 | SMF90T42_Data | * | * | Start of the header/self-defining section. |
| 0 | 0 | SMF90T42_Cont | 1 | binary | Continuation information. CX and Bad_CX entries will not be in any record until all the Audit entries are in record(s). Bad_CX entries will not be in any record until all the Audit and CX entries are in record(s).<br>**Bit (Name)**<br>**Meaning when set**<br>**X'80' (SMF90T42_NotFirst)**<br>Not the first; this is a continuation record.<br>**X'40' (SMF90T42_NotLast)**<br>There is another record that continues this one. |
| 1 | 1 | SMF90T42_Flags | 1 | binary | Flags.<br>**Bits (Name)**<br>**Meaning when set**<br>**X'C0' (SMF90T42_F_VBMODE)**<br>Validated boot mode. The options are mutually exclusive.<br>**X'80' (SMF90T42_F_Enforce)**<br>Enforce mode.<br>**X'40' (SMF90T42_F_Audit)**<br>Audit mode. |
| 2 | 2 | SMF90T42_Part | 2 | binary | 0 (part 0) for the first of a set of continued records; 1 (part 1) for the next, and so on. |
| 4 | 4 | * | 4 | binary | Reserved. |
| 8 | 8 | SMF90T42_NumFailures | 4 | binary | Total number of verification failures. Set only for the record that is "part 0." |
| 12 | C | SMF90T42_NumFailures_NoDSNE | 4 | binary | Number of verification failures that are not represented by the more detailed information mapped by SMF90T42_Audit (see "Audit entry section" on page 92). This could be because there was not enough storage available to do the tracking. Set only for the record that is "part 0." |
| 16 | 10 | SMF90T42_Audit_Off | 4 | binary | Offset to first audit entry. Valid only when there is at least one audit entry. |
| 20 | 14 | SMF90T42_Audit_Len | 2 | binary | Length of an audit entry. Must use this length to navigate from one entry to the next. |
| 22 | 16 | SMF90T42_Audit_Num | 2 | binary | Number of contiguous audit entries. |

| Offsets | | Name | Length | Format | Description |
|---|---|---|---|---|---|
| 24 | 18 | SMF90T42_CX_Off | 4 | binary | Offset to first certificate extract (CX) entry. Valid only when there is at least one CX entry. |
| 28 | 1C | SMF90T42_CX_Len | 2 | binary | Length of a CX entry. Must use this length to navigate from one entry to the next. |
| 30 | 1E | SMF90T42_CX_Num | 2 | binary | Number of contiguous CX entries. |
| 32 | 20 | SMF90T42_Bad_CX_Off | 4 | binary | Offset to first bad certificate extract (Bad_CX) entry. Valid only when there is at least one Bad_CX entry. |
| 36 | 24 | SMF90T42_Bad_CX_Len | 2 | binary | Length of a Bad_CX entry. Must use this length to navigate from one entry to the next. |
| 38 | 26 | SMF90T42_Bad_CX_Num | 2 | binary | Number of contiguous Bad_CX entries. |
| 40 | 28 | SMF90T42_TZO | 8 | binary | Time zone offset, in hexadecimal. |
| 48 | 30 | SMF90T42_LeapSeconds | 8 | binary | Leap seconds, in hexadecimal. |

## Audit entry section

| Offsets | | Name | Length | Format | Description |
|---|---|---|---|---|---|
| 0 | 0 | SMF90T42_Audit | * | * | Start of an audit entry section. |
| 0 | 0 | SMF90T42_A_Modname | 8 | EBCDIC | Module name. If this is "*unknown", the system was unable to obtain enough storage to track the name of the failing module and settled for an entry describing the data set from which the module was being fetched. |
| 8 | 8 | SMF90T42_A_Dsname | 44 | EBCDIC | Data set name. |
| 52 | 34 | SMF90T42_A_VolID | 6 | EBCDIC | Volume ID. |

| Offsets | | Name | Length | Format | Description |
|---|---|---|---|---|---|
| 58 | 3A | SMF90T42_A_Fail_Reason | 2 | binary | Reason for validation failure.<br><br>**Value (Name)**<br>**Description**<br><br>**1 (SMF90T42_VF_NotSigned)**<br>Not signed.<br><br>**2 (SMF90T42_VF_DENotFound)**<br>Directory entry not found.<br><br>**3 (SMF90T42_VF_DENotMatch)**<br>Directory entry does not match.<br><br>**4 (SMF90T42_VF_SigNotFound)**<br>Signature not found.<br><br>**5 (SMF90T42_VF_BadHashAlg)**<br>Signature record does not indicate a valid hash algorithm.<br><br>**6 (SMF90T42_VF_BadSigAlg)**<br>Signature record does not indicate a valid signature algorithm.<br><br>**7 (SMF90T42_VF_BadHashVal)**<br>Hash value in the signature record does not match the calculated hash value.<br><br>**8 (SMF90T42_VF_NoMatchingKeyID)**<br>The key ID in the signature record does not match any verification key available to this LPAR.<br><br>**9 (SMF90T42_VF_SigVerFailed)**<br>The signature verification operation did not complete successfully.<br><br>**10 (SMF90T42_VF_OverlayModule)**<br>This is an overlay module. Signature support is not provided.<br><br>**11 (SMF90T42_VF_BadSigRecVersion)**<br>Signature record version is not valid.<br><br>**12 (SMF90T42_VF_MachLoaderError)**<br>Machine loader error. |
| 60 | 3C | SMF90T42_A_Flags | 1 | binary | Flags.<br><br>**Bit (Name)**<br>**Meaning when set**<br><br>**X'80' (SMF90T42_A_FoundSig)**<br>Found the signature record for this module.<br><br>**X'20' (SMF90T42_A_HaveMachLoaderErrors)**<br>SMF90T42_A_MachLoaderErrors has information. |
| 61 | 3D | * | 3 | binary | Reserved. |
| 64 | 40 | SMF90T42_A_NumFailures | 4 | binary | Number of validation failures. |
| 68 | 44 | SMF90T42_A_DSN_NumFailures | 4 | binary | Total number of validation failures for the entire data set. It is presented (duplicated) in every audit record for the data set. |
| 72 | 48 | SMF90T42_A_SignTime | 8 | binary | Signing time (first 8 bytes of ETOD). Valid only when SMF90T42_A_FoundSig is on. |
| 80 | 50 | SMF90T42_A_CertFP | 32 | binary | Signing certificate fingerprint, in hexadecimal. Valid only when SMF90T42_A_FoundSig is on. |
| 112 | 70 | SMF90T42_A_Union | 20 | binary | Mapping depends on SMF90T42_A_FoundSig and SMF90T42_A_HaveMachLoaderErrors bits, as follows. |
| 112 | 70 | SMF90T42_A_KeyID | 20 | binary | Signing key ID, in hexadecimal. Applies only when SMF90T42_A_FoundSig is on. |
| 112 | 70 | SMF90T42_A_MachLoaderErrors | 6 | binary | Machine loader errors, in hexadecimal. Applies only when SMF90T42_A_HaveMachLoaderErrors is on. |

| Offsets | | Name | Length | Format | Description |
|---|---|---|---|---|---|
| 112 | 70 | SMF90T42_A_MLE_ED | 4 | binary | Error details. |
| 116 | 74 | SMF90T42_A_MLE_IIEI | 2 | binary | Error information. |
| 132 | 84 | SMF90T42_A_FailTime | 8 | binary | Failure time (first 8 bytes of ETOD). |

### Certificate extract (CX) section

| Offsets | | Name | Length | Format | Description |
|---|---|---|---|---|---|
| 0 | 0 | SMF90T42_CX | * | * | Start of a certificate extract section. |
| 0 | 0 | SMF90T42_CX_CertName | 64 | EBCDIC | Certificate name. |
| 64 | 40 | SMF90T42_CX_CertFP | 32 | binary | Certificate fingerprint, in hexadecimal. |
| 96 | 60 | SMF90T42_CX_KeyID | 20 | binary | Key ID, in hexadecimal. |
| 116 | 74 | SMF90T42_CX_NumSuccessfulUses | 4 | binary | Number of successful uses. |
| 120 | 78 | SMF90T42_CX_StartTime | 8 | binary | Certificate start time (first 8 bytes of ETOD). |
| 128 | 80 | SMF90T42_CX_ExpirationTime | 8 | binary | Certificate expiration time (first 8 bytes of ETOD). |
| 136 | 88 | SMF90T42_CX_ReasonBad | 4 | binary | Failure reason. When non-0, the reason will be SMF90T42_BCX_Reason_BadKey. |

### Bad certificate extract (Bad_CX) section

| Offsets | | Name | Length | Format | Description |
|---|---|---|---|---|---|
| 0 | 0 | SMF90T42_Bad_CX | * | * | Start of a bad certificate extract section. |
| 0 | 0 | SMF90T42_BCX_CertName | 64 | EBCDIC | Certificate name. |
| 64 | 40 | SMF90T42_BCX_CertFP | 32 | binary | Certificate fingerprint, in hexadecimal. Zeros when not known. |
| 96 | 60 | SMF90T42_BCX_KeyID | 20 | binary | Key ID, in hexadecimal. Zeros when not known. |
| 116 | 74 | SMF90T42_BCX_StartTime | 8 | binary | Certificate start time (first 8 bytes of ETOD). Zeros when not known. |
| 124 | 7C | SMF90T42_BCX_ExpirationTime | 8 | binary | Certificate expiration time (first 8 bytes of ETOD). Zeros when not known. |
| 132 | 84 | SMF90T42_BCX_ReasonBad | 4 | binary | Failure reason code.<br>**Value (Name)**<br>**Description**<br>**1 (SMF90T42_BCX_Reason_NotStarted)**<br>Certificate is not yet valid.<br>**2 (SMF90T42_BCX_Reason_Expired)**<br>Certificate has expired.<br>**3 (SMF90T42_BCX_Reason_BadKey)**<br>Key is not valid.<br>**4 (SMF90T42_BCX_Reason_BadKeyType)**<br>Key type is not valid.<br>**5 (SMF90T42_BCX_Reason_BadKeyIDLen)**<br>Key ID length is not valid.<br>**6 (SMF90T42_BCX_Reason_BadHashType)**<br>Hash type is not valid.<br>**7 (SMF90T42_BCX_Reason_BadHashLen)**<br>Hash length is not valid. |

# Chapter 9. MVS system messages

**AMD126I** **Error in List-Directed Dump-Save Area**

## Explanation

An error occurred while saving or restoring storage contents or storage keys from the Dump-Save Area during a List-Directed Dump. Some storage or storage keys in the dump may not be the original contents of the storage or storage keys at the time that the dump was initiated.

## System action

The SADMP program continues.

## Operator response

None.

## System programmer response

This is most likely a result of a problem in the machine (or the virtual machine control program, if running in virtual machine). Contact the IBM support center.

## Source

Stand-alone Dump (SADMP)

## Module

AMDSAICN, AMDSARDX

## Routing code

-

## Descriptor code

-

**AMD127I** **SADMP IPL device is write inhibited**

## Explanation

For a List-Directed Dump, the device from which SADMP was IPLed is write inhibited. This prevents SADMP from saving the contents of some storage pages before using them, so that those pages in the dump may contain SADMP code or work areas instead of the original contents of the storage at the time that the dump was initiated.

## System action

The SADMP program continues.

## Operator response

None.

## System programmer response

None.

## Source

Stand-alone Dump (SADMP)

## Module

AMDSAICN

## Routing code

-

## Descriptor code

-

**AMD128I** **List-Directed Dump. Validated Boot: *type***

## Explanation

Indicates that a list-directed IPL of SADMP was done. If this message is not present, a channel command word (CCW) IPL of SADMP was done.

In the message text:

***type***
One of the following values:

**Audit**
Indicates that the "Secure Boot" option was not identified when doing the IPL.

**Enforce**
Indicates that the "Secure Boot" option was identified when doing the IPL.

## System action

The SADMP program continues.

## Operator response

None.

## System programmer response

None.

## Source

Stand-alone Dump (SADMP)

## Module

AMDSAICN

## Routing code

-

## Descriptor code

-

**AMD129I** ***xxxxxxxx yyyy description***

## Explanation

During a List-directed IPL of SADMP when the "Secure Boot" option was not identified, problems were detected which would have cause the IPL to fail if the "Secure Boot" option had been identified.

In the message text:

***xxxxxxxx***
: Secure-IPL Code Loading Attributes Facility (SCLAF) error details.

***yyyy***
: IPL-information Error Indicators (IIEI).

***description***
: One of the following reasons:

```
Machine loader detected error(s)
Signature verification failed
Module was not signed
```

## System action

The SADMP program continues.

## Operator response

Record any messages from the machine loader which appear on the System Console (Operating System Messages).

## System programmer response

Regenerate the stand-alone dump program to correct any problems.

## Source

Stand-alone Dump (SADMP)

## Module

AMDSAICN

## Routing code

-

## Descriptor code

-

**CSV050I** **Signature verification failed for module *mmmmmmmm* in dataset *d*. {Not signed | Reason: *r*}**

## Explanation

Validated Boot for z/OS "enforce" mode is in effect. A module to be verified did not pass verification. This message is followed by a wait state EC9 with a reason corresponding to the reason in the message. In some cases, wait state EC9 can occur without being preceded by this message.

In the message text:

***mmmmmmmm***
: The name of the load module.

***d***
: The name of the data set (with trailing blanks removed).

***r***
: One of the following decimal reason codes:

**02**
: Directory entry not found.

**03**
: Directory entry does not match.

**04**
: Signature not found.

**05**
: Signature record does not indicate a valid hash algorithm.

**06**
: Signature record does not indicate a valid signature algorithm.

**07**
: Hash value in the signature record does not match the calculated hash value.

**08**
: The key ID in the signature record is not matched by any verification key available to this LPAR.

**09**
: The signature verification operation (KDSA instruction) did not complete successfully.

**10**
: This is an overlay module. Signature support is not provided.

**11**
: The version of the signature record is not valid.

## System action

The system enters wait state EC9.

## System programmer response

Correct the issue.

## Source

Contents supervision (CSV)

## Routing code

Not applicable.

## Descriptor code

12

**IAR078I** **VALIDATED BOOT - STORAGE-CLASS MEMORY IS NOT AVAILABLE. WOULD WAIT STATE IF ENFORCE MODE.**

**Explanation:**
No SCM is available during IPL for an audit-mode Validated Boot for z/OS.

## System action

The system continues.

## Operator response

Contact the system programmer.

## System programmer response

Add more SCM.

## Source

Real storage manager (RSM).

## Module

IAXBI.

## Routing code

2, 10.

## Descriptor code

12.

**IAR079W** **VALIDATED BOOT - STORAGE-CLASS MEMORY IS REQUIRED**

**Explanation:**
No SCM is available during IPL for an enforce-mode Validated Boot for z/OS.

## System action

The system enters wait state 'A2D'X.

## Operator response

Contact the system programmer.

## System programmer response

Add more SCM.

## Source

Real storage manager (RSM)

## Module

IAXBI.

## Routing code

1.

## Descriptor code

2.

**IEA980I** **CERTIFICATE *xxxx* NOT USED: *reason***

## Explanation

A certificate was found that could not be used. *xxxx* is the certificate name defined by the customer when providing this certificate. The provided *reason* describes the specific issue which is one of the following:

- Not valid yet.
- Expired.

- Key is not valid.
- Key type is not valid.
- Key ID length is not valid.
- Hash type is not valid.
- Hash length is not valid.

**System action:**
The system continues, not using this certificate.

**System programmer response:**
Correct the issue.

## Source

System initialization (IPL/NIP)

**IEA981I** **VALIDATED BOOT FOR Z/OS ERROR: *reason***

## Explanation

An error was detected within the validated boot processing. The provided *reason* describes the specific issue which is one of the following:

- Certificate info not available 1.
- Certificate info not available 2.
- CPACF functions not available.
- Certificate format not supported.
- Certificate info not available 3.
- No valid certificates.

**System action:**
The system continues.

## System programmer response

For all reasons but no valid certificates, contact IBM Service. For no valid certificates, make sure that one or more valid certificates have been associated with this LPAR.

## Source

System initialization (IPL/NIP)

**IEE174I (form 33 of 37)** **DISPLAY M STORAGE-CLASS MEMORY STATUS *status***

## Explanation

The DISPLAY M=SCM command requested the system to display the status of storage-class memory (SCM). If DISPLAY M=SCM(DETAIL) was specified, the message also includes - INCREMENT DETAIL.

In the message text, *status* includes the following information:

***dddd*{M|G|T} DEFINED**
The amount of SCM defined to this partition.

**ADDRESS IN USE STATUS *dddd*{M|G|T} *dd*% ONLINE**
The amount of SCM that is currently in use.

**ONLINE-PERMANENTLY RESIDENT *dddd*{M|G|T}-*dddd*{M|G|T}**
The range of permanently resident SCM that is currently online.

**ONLINE-RECONFIGURABLE *dddd*{M|G|T}-*dddd*{M|G|T}**
The range of reconfigurable SCM that is currently online.

***dd*{M|G|T} OFFLINE-AVAILABLE**
The amount of SCM that is currently offline.

***ddd*% IN USE**
The percentage of the total SCM that is in use.

**SCM INCREMENT SIZE IS *dd*{M|G|T}**
The SCM increment size.

**SCM STATUS NOT OBTAINED: SCM NOT SUPPORTED**
SCM status was not obtained because SCM is not supported on this processor.

## Source

Reconfiguration

## Module

IEEDMSCM

## Routing code

-

## Descriptor code

5, 8, 9

**IEE193I** **NO SCM RECONFIGURED – *reason* OR DISPLAY SCM INSUFFICIENT STO RAGE FOR COMMAND**

## Explanation

The system did not reconfigure any storage-class memory (SCM) in response to a CONFIG SCM command.

In the message text, *reason* can be one of the following:

**TIMEOUT OCCURRED**
The system attempted to physically configure the SCM online, but could not determine that the reconfiguration was performed.

**OPERATOR CANCELLED**
The operator replied CANCEL to message IEE575A to cancel a CONFIG SCM command.

**SCM NOT DEFINED**
There is no SCM defined to this partition.

**SCM NOT SUPPORTED**
SCM is not supported on this processor.

**AMOUNT NOT VALID**
The amount specified on the CONFIG SCM command is not a valid value.

**REQUEST EXCEEDS *dd*M|G|T DEFINED**
The amount specified exceeds the amount of SCM that is currently defined to the partition.

**REQUEST IS NOT A MULTIPLE OF *dd*M|G|T**
The amount specified is not a multiple of the SCM increment size.

**RANGE EXCEEDS MAX ADDRESS *dd*M|G|T**
The values in the range exceed the highest possible SCM address for this system.

**NO OFFLINE SCM**
There is no SCM eligible to be brought online to this partition.

**NO ONLINE SCM**
There is no SCM online to this partition.

**INSUFFICIENT RECONFIGURABLE ONLINE SCM**
Some SCM storage may be online but it is not reconfigurable.

**INSUFFICIENT AUXILIARY STORAGE**
Insufficient auxiliary storage would remain if the CONFIG SCM OFFLINE request were completed. This reason text is displayed when the amount of auxiliary storage currently being used is more than 50% of the auxiliary storage that would remain after the offline completed.

**REAL FRAME SHORTAGE**
The system is critically low on available real frames, so the CONFIG SCM OFFLINE command was canceled.

**INTERNAL ERROR, DIAG1=*xxxxxxxx***
An internal error occurred or an abnormal condition was detected while processing the CONFIG command. The DIAG1 value is internal diagnostic information to supply to IBM when requesting service.

**CF ONLINE BY RANGE NOT SUPPORTED**
Bringing SCM online by specifying a range is not supported.

## System action

The system continues processing.

## Operator response

Notify the system programmer.

## System programmer response

If the command was rejected because of a problem with the amount of SCM specified, examine the amount of SCM that was requested to be brought online or taken offline and verify that the amount is correct and specified as a valid increment. You can use the D M or D M=SCM command to obtain information regarding the amount of SCM that is eligible to be brought online or taken offline, and to obtain other SCM-related attributes such as the SCM increment size.

If the command was rejected because of a timeout condition, retry the command. If the command continues to be rejected with the timeout condition, contact IBM service.

If the command was rejected because of the SCM NOT DEFINED reason, verify your image profile.

If the command was rejected because of the SCM NOT SUPPORTED reason, this indicates that the system on which the command was issued does not support the use or definition of SCM.

If the message was issued with the INTERNAL ERROR reason text, contact IBM service and provide the internal diagnostic information value.

## Source

Reconfiguration

## Module

IEEVSCM

## Routing code

Note 2

## Descriptor code

5

**IEE194I** **CF SCM REQUEST NOT FULLY SATISFIED – *reason***

## Explanation

The system was not able to configure the full amount of storage-class memory (SCM) that was requested with the CONFIG SCM command.

In the message text, *reason* can be one of the following:

**INSUFFICIENT OFFLINE SCM**
: Less than the requested amount of SCM is eligible to be brought online.

**INSUFFICIENT ONLINE SCM**
: Less than the requested amount of SCM is eligible to be brought offline.

**INSUFFICIENT RECONFIGURABLE ONLINE SCM**
: SCM storage may be online but some of it is not reconfigurable.

**TIMEOUT OCCURRED**
: The system attempted to physically configure the SCM online, but could not determine that the reconfiguration was performed.

**INTERNAL ERROR, DIAG1=*xxxxxxxx***
: An internal error occurred or an abnormal condition was detected while processing the CONFIG command. The DIAG1 value is internal diagnostic information to supply to IBM when requesting service.

## System action

The system configures a portion of the requested amount of SCM and continues processing.

## Operator response

Notify the system programmer.

## System programmer response

Examine the amount of SCM that was requested to be brought online and verify that the amount is correct.

If the command was rejected because of a timeout condition, retry the command. If the command continues to be rejected with the timeout condition, contact IBM service.

If the message was issued with the INTERNAL ERROR reason text, contact IBM service and provide the internal diagnostic information value.

## Source

Reconfiguration

## Module

IEEVSCM

## Routing code

Note 2

## Descriptor code

5

**IEE195I** **SCM LOCATIONS *xxx*G TO *yyy*G NOT RECONFIGURED – *reason***

## Explanation

The SCM ranges displayed in the message have been configured online or offline to the system. For OFFLINE, the range can include increments that are already offline when the CONFIG command is issued.

In the message text, *reason* is:

**PERMANENTLY RESIDENT**
: Indicates which SCM storage failed to be taken offline because it is permanently resident.

## System action

The system continues processing.

## Source

Reconfiguration

## Module

IEEVSCM

## Routing code

Note 2

## Descriptor code

5

**IEE196I** **AMOUNT OF CENTRAL STORAGE EXCEEDS *n*T MAXIMUM: RECONFIGURATION FUNCTIONS ARE NOT AVAILABLE.**

## Explanation

Storage reconfiguration is not supported when there is more than *n* terabytes of central storage.

## System action

Central storage reconfiguration is disabled. The system continues its initialization process.

**Operator response:**

Contact the system programmer if reconfiguration is required.

**System programmer response:**
If reconfiguration is required, reduce the installed storage to a maximum of *n* TB, then re-IPL the system to enable storage reconfiguration.

## Source

Reconfiguration

## Module

IEERMAXW

## Routing code

2, 10

## Descriptor code

12

**IEE254I** ***hh.mm.ss* IPLINFO DISPLAY *text***

## Explanation

Where *text* is:

```
SYSTEM IPLED AT hh.mm.ss ON mm/dd/yyyy
RELEASE fmid LICENSE = system
USED LOADxx IN loadxxdsname ON devx
ARCHLVL = n MTLSHARE = Y|N
VALIDATED BOOT: {NO | {ENFORCE|AUDIT},INACTIVE}
IEASYM LIST = s1|NONE
IEASYS LIST = s2{(OP)}
IODF DEVICE: ORIGINAL(iodfdev1)
CURRENT(iodfdev2)
IPL DEVICE: ORIGINAL(ipldev1) CURRENT(ipldev2)
VOLUME(iplvol)
[vminfo]
```

Displays IPL information when a DISPLAY IPLINFO command is issued.

In the message text:

***hh.mm.ss***
The current time. The time format is in hours (00–23), minutes (00–59) and seconds (00–59).

***hh.mm.ss***
The master scheduler initialization completed; the IPL completed. The time format is in hours (00–23), minutes (00–59) and seconds (00–59).

***mm/dd/yyyy***
The master scheduler initialization completed; the IPL completed. That date format is in month (01–12), day (01–31) and year (0000–9999).

***rrrrrrr***
The release level of the system being IPLed.

***xx***
The LOADxx member used to IPL the system.

***loadxxdsname***
The data set where the LOADxx originated. The data set name will be either:

- SYSn.IPLPARM (where *n* can be 0–9)
- SYS1.PARMLIB

***devx***
The device address where the LOADxx member originated. When the device number is in the form *snnnn*, the first digit indicates the subchannel set.

**ARCHLVL = *n***
The value *n* is either 1 to indicate ESA/390 or 2 to indicate z/Architecture®. The value displayed is the value specified (or defaulted) by the ARCHLVL statement of the LOADxx parmlib member.

**MTLSHARE = *Y|N***
MTL tape devices are treated as regular standalone drives (**Y**), as compared to MTL resident drives (**N**).

**VALIDATED BOOT: NO**
**VALIDATED BOOT: ENFORCE,INACTIVE**
**VALIDATED BOOT: AUDIT,INACTIVE**
Indicates whether this is a validated boot, an audited validated boot, or not a validated boot.

**NO**
Indicates that is not a validated boot.

**ENFORCE**
Indicates that this is a validated boot with the "Secure Boot" option identified when doing the IPL.

**AUDIT**
Indicates that this is a validated boot without the "Secure Boot" option identified when doing the IPL.

**INACTIVE**
Indicates that validation is no longer being done. Validation ends when the LPA is built. Building of LPA precedes when the DISPLAY IPLINFO command can be issued.

**IEASYM LIST = *s1***
The IEASYMxx member or members used by the IPLed system. *s1* can be either a single member name, a list of members (specified in parentheses), or NONE. The default value is NONE.

**IEASYS LIST = *s2***
The IEASYSxx member or members used by the IPLed system. *s2* can be either a single member name or a list of members (specified in parentheses).

**OP**
The IEASYSxx values were specified in LOADxx SYSPARM statements or from the reply to the IEA101A system parameters prompt.

***iodfdev1***
The device number of the volume where the I/O configuration resided when the system was originally IPLed. When the device number is in the form *snnnn*, the first digit indicates the subchannel set.

***iodfdev2***
The device number of the volume where the I/O configuration now resides. *iodfdev1* and *iodfdev2* can be the same or can be different if they were primary and secondary of a PPRC pair being monitored for HyperSwap and a HyperSwap has occurred. When the device number is in the form *snnnn*, the first digit indicates the subchannel set.

***ipldev1***
The SYSRES device number from which the system was originally IPLed. When the device number is in the form *snnnn*, the first digit indicates the subchannel set.

***ipldev2***
The current SYSRES device number. *ipldev1* and *ipldev2* can be the same or can be different if they were primary and secondary of a PPRC pair being monitored for HyperSwap and a HyperSwap has occurred. When the device number is in the form *snnnn*, the first digit indicates the subchannel set.

***iplvol***
The IPL volume serial.

***system***
The IPLed system, z/OS or zNALC (a z/OS system that requested zNALC pricing).

***vminfo***
When z/OS is running in a virtual machine, *vminfo* is displayed, as follows. The data is provided by the hypervisor via the STSI instruction. See the documentation provided by the hypervisor, if and when available, for details about the data values.

```
VM CPID = vmcpid
VM UUID = vmuuid | VM UUID IS NOT PROVIDED
VM NAME = vmname
VM EXT NAME = vmextname | VM EXT NAME IS NOT
PROVIDED | VM EXT NAME IS NOT EBCDIC ENCODED
```

where:

***vmcpid***
The 16-character control program identifier. For example:

```
VM CPID = zHYPaaS
```

***vmuuid***
The universally unique identifier (UUID) for the virtual machine, in the 8-4-4-4-12 format. For example:

```
VM UUID = 72DF625E-AD77-4A6C-865D-
B9718A67898F
```

If the value is not provided, the display line shows:

```
VM UUID IS NOT PROVIDED
```

***vmname***
The 8-character virtual machine name. For example:

```
VM NAME = k8s_4094
```

***vmextname***
The extended virtual machine name. When available, the name can be from 1–256 characters long. If necessary, the characters will flow to the next line of the display.

- Example 1:

```
VM EXT NAME =
k8s_4094bc65b1bb4196b016f4651f0788bc_02
u7
```

- Example 2:

```
VM EXT NAME =
k8s_4094bc65b1bb4196b016f4651f0788bc_02
u7_72df625e-ad77-4a6c-865d
              -b9718a67898f
```

If the value is not provided, the display line shows:

```
VM EXT NAME IS NOT PROVIDED
```

If the extended virtual machine name is not EBCDIC-encoded, the display line shows:

```
VM EXT NAME IS NOT EBCDIC ENCODED
```

## System action

The system continues processing.

## Source

Master scheduler

## Module

IEECB985

## Routing code

*

## Descriptor code

5

**ILR041I** **VALIDATED BOOT - PLPA PAGE DATA SET SPECIFICATION IGNORED DUE TO ENFORCE MODE**

## Explanation

The specified PLPA paging data set is not used and the IPL continues as if *NONE* was specified instead in an enforce-mode Validated Boot for z/OS.

## System action

The IPL continues as if *NONE* was specified.

## Operator response

Contact the system programmer.

## System programmer response

Avoid specifying a PLPA data set.

## Source

Auxiliary Storage Management (ASM)

## Module

ILRASRIM

## Routing code

2, 10..

## Descriptor code

12.

**ILR042I** **VALIDATED BOOT - PLPA PAGE DATA SET SPECIFICATION WOULD BE IGNORED IF ENFORCE MODE**

## Explanation

A paging data set is specified which the system uses if it overfills SCM in an audit-mode Validated Boot for z/OS, but this specification would be ignored if enforce mode.

## System action

The system continues.

## Operator response

Contact the system programmer.

## System programmer response

Avoid specifying a PLPA data set.

## Source

Auxiliary Storage Management (ASM)

## Module

ILRASRIM

## Routing code

2, 10.

## Descriptor code

12.

**ILR043I** **VALIDATED BOOT - STORAGE-CLASS MEMORY IS FULL. WOULD WAIT STATE IF ENFORCE MODE.**

## Explanation

SCM is filled up during IPL with LPA data, and that LPA data is going to be stored onto the next paging data set with available space in an audit-mode Validated Boot for z/OS.

## System action

The system continues, storing LPA data onto the next paging data set with available space.

## Operator response

Contact the system programmer.

## System programmer response

Ensure the size of SCM memory is at least the size of LPA storage.

## Source

Auxiliary Storage Management (ASM)

### Module

ILRIODRV

### Routing code

2, 10.

### Descriptor code

12.

# Chapter 10. Wait state codes

**03C**

## Explanation

The auxiliary storage manager (ASM) found that not enough auxiliary storage space is available for system operation:

**During IPL**

- Either the required number of page data sets was not specified, or ASM detected a problem with a required page data set.
- ASM has run out of usable storage-class memory (SCM) while paging out the pageable link pack area (PLPA) data during an enforce-mode Validated Boot for z/OS.

**After IPL**

ASM has run out of usable auxiliary storage for the pageable link pack area (PLPA), common or local page data sets, and any storage-class memory (SCM).

A reason code identifies the error:

| Code | Explanation |
|---|---|
| 00 | The cause of the error cannot be determined because of an error in recovery processing. |
| 01 | Insufficient paging space. All local paging data sets are full, and there is no available SCM. |
| 02 | The PLPA data set is full, SCM is full, and the common data set is unavailable. |
| 03 | The common data set is full, SCM is full, and the PLPA data set is unavailable. |
| 04 | SCM is full, and the PLPA and common data sets are unavailable, or Validated Boot for z/OS is preventing paging to other data sets. |

## System action

During IPL, the system issues message IEA935W, then enters this wait state. After IPL, the system issues message ILR008W, then enters this wait state, unless all the local page data sets and storage-class memory (SCM) blocks were unusable; in that case, the system does not issue a message before entering this wait state.

## Operator response

Notify the system programmer. ReIPL the system, specifying larger page data sets or additional page data sets, or additional SCM blocks.

## System programmer response

Do one of the following:

- Redefine spaces to conform with the description provided in paging planning specifications. Ask the operator to reIPL with the CLPA option.
- Provide additional paging spaces and make them available through either the PAGE parameter or the IEASYSxx parmlib member during reIPL.
- After additional paging spaces are added and the system is re-IPLed, issue the D ASM command to monitor the available paging space to help prevent a reoccurrence of the wait state condition.
- Provide additional SCM blocks and make them available to the system using the CONFIG SCM ONLINE command.

## Source

Auxiliary storage manager (ASM)

**A2D**

## Explanation

The `PAGE=*NONE*` parameter was specified but an error occurred for one of the following reasons:

- No storage-class memory (SCM) is available for paging. When `*NONE*` is used on the `PAGE=` parameter, SCM must be online and available, and `PAGESCM=*NONE*` must not be specified.
- Validated Boot for z/OS ignored the input paging dataset because of enforce-mode and used `*NONE*` instead, causing the error.

## System action

- If no storage-class memory (SCM) is available for paging, message IAR036W is issued and the system enters wait state X'A2D'.
- If Validated Boot for z/OS ignored the input paging dataset because of enforce-mode, message IAR079W is issued and the system enters wait state X'A2D'.

## Operator response

Notify the system programmer.

## System programmer response

If PAGE=*NONE* is intended, determine why SCM is not available for paging. Configure SCM online for the system and ensure that PAGESCM=*NONE* is not specified. Alternatively, change *NONE* to a valid page data set on the PAGE= system parameter.

## Source

Real storage manager (RSM)

**EC9**

## Explanation

The IPL requested Validated Boot in Secure mode and an error was found.

The right-most four bytes of the program status word (PSW) have the following format:

```
80rrrwww
```

where:

***rrr***
: A hexadecimal reason code indicating the reason for the failure.

***www***
: The wait state code (x'EC9')

The reason code is one of the following:

**Code**
: **Explanation**

**1**
: The module is not signed. Modules being validated must be signed. Use the IEWSIGN utility to sign the module.

**2**
: The module's signing records do not include an entry for the module name itself. The module might be corrupted. Possibly you renamed the module which will cause validation to fail.

**3**
: The module's signing records has a directory entry for the module name but it does not match the actual directory entry. The module (or its directory entry) might be corrupted.

**4**
: The module is indicated as being signed but does not have a signature record. The module might be corrupted.

**5**
: The module is signed with a hash algorithm that is not supported. The module might be corrupted.

**6**
: The module is signed with a signature algorithm that is not supported. The module might be corrupted.

**7**
: The hash value in the signing records does not match the hash value of the module itself. The module might be corrupted.

**8**
: The signing record indicates a key ID that must be used for validation, but no certificate with that key ID has been made available to this LPAR. Make sure that the appropriate certificate is available to this LPAR.

**9**
: The signature verification did not succeed. The module might be corrupted, or the certificate is no longer valid.

**X'A'**
: A module to be validated has the overlay attribute. This is not supported. Remove this from the data set with members being validated.

**X'B'**
: The signature record has a version value that is not supported. The module might be corrupted.

**X'C'**
: Machine loader detected a problem.

**X'101'**
: Report this error to IBM service.

**X'102'**
: Report this error to IBM service.

**X'103'**
: Report this error to IBM service.

**X'104'**
: Report this error to IBM service.

**X'105'**
: Report this error to IBM service.

**X'106'**
: No validation certificates have been associated with this LPAR. Associate the appropriate certificate(s) with this LPAR.

At the time of the wait state, 64-bit general register 2 will contain the module name found to be in error (if the problem is associated with an individual module)

## System action

The system enters a nonrestartable wait state.

### Operator response

Record the wait state PSW. Notify the system programmer, and obtain a stand-alone dump, if requested.

### System programmer response

Determine the reason for the failure, either from time of error registers or, from a stand-alone dump using the IEAVBIPC VERBexit. Correct the problem and re-IPL. You could alternately re-IPL requesting validated boot not in Secure mode and then using the IEAVBPRT program to format the audit records that describe any problems found.

### Source

Initial Program Load (IPL)

## Wait state code to module table

This table correlates wait state codes with module names. For each code, the associated component and detecting module are listed.

If the wait state code you require does not appear in the table, contact the IBM Support Center. Provide the wait state code.

*Table 37. Wait state codes to modules reference*

| Wait state code (Hex) | Component | Detecting module |
|---|---|---|
| 002 | IOS | IEAIPL00 |
| 003 | IPL, IOS | IEAIPL00 IEAIPL03 |
| 004 | IOS | IEAIPL03 |
| 005 | IPL | IEAIPL00 |
| 006 | IPL | IEAIPL00 |
| 007 | Console services | IEAVNPCA IEEVDCIO |
| 009 | NIP | IEAVNPC4 |
| 00A | IPL | IEAVNP03 |
| 00B | Master scheduler | IEEVIPL |
| 00D | Master scheduler | IEEVIPL |
| 00E | IPL | IEAIPL00 |
| 014 | Supervisor control | IEAVEPCO IEAV9PCO |
| 017 | IPL | IEAIPL00 |
| 019 | IPL | IEAIPL00 |
| 01B | RTM | IEESTPRS |
| 01C | Supervisor control | IEAVESPR |
| 020 | Reconfiguration | IEAVNP27 |
| 021 | IOS | IEAVNPM2 |
| 022 | IOS | IECVDAVV |
| 023 | System trace | IEAVNP51 |
| 024 | MCH | IGFPTREC |
| 025 | IPL | IEAIPL41 |
| 028 | IOS | IEAIPL40 |

*Table 37. Wait state codes to modules reference (continued)*

| **Wait state code (Hex)** | **Component** | **Detecting module** |
|---|---|---|
| 02E | ASM | ILRMSG00 |
| 02F | ASM | ILRCMP ILRMSG00 |
| 030 | NIP | IEAVNIP0 |
| 031 | IOS | IEAIPL03 |
| 032 | NIP | IEAVNIPM IEAVNIP0 |
| 033 | NIP | IEAVNIPM IEAVNIP0 |
| 035 | IOS | IEAIPL03 |
| 037 | System Environmental Recording | IEAVNP76 |
| 038 | ASM IPL | IEAIPL00 IEAVNP05 ILRASRIM |
| 039 | IOS | IEAVNPM3 |
| 03A | CSV | IEAVNP05 |
| 03B | CSV | IEAVNP05 |
| 03C | ASM | ILRASRIM ILRIODRV ILRMSG00 |
| 03D | VSM | IEAVNP08 |
| 03E | ASM | ILRTMI00 |
| 03F | NIP | IEAVNPM2 |
| 040 | NIP | IEAVNIPM |
| 044 | NIP | IEAVNIP0 |
| 045 | RTM | IEAVNIPM |
| 046 | NIP | IEAVNIP0 |
| 04A | NIP | IEAVNIP0 |
| 050 | Loadwait | IGFPTSIG |
| 051 | ACR | IGFPTERM |
| 052 | ACR | IGFPTERM |
| 053 | IOS IPL NIP | IEAIPL03 IEAIPL99 IEAVNIP0 IOSIUCB |
| 054 | IPL | IEAIPL02 |
| 055 | IOS IPL | IEAIPL31 IEAIPL41 IEAIPL02 IEAIPL40 |
| 059 | NIP | IEAVNIP0 |
| 05C | DFSMSdfp | IEAVNP11 |
| 05D | DFSMSdfp | IEAVNP11 |
| 05E | DFSMSdfp | IEAVNP11 |
| 05F | DFSMSdfp | IEAVNP11 |
| 060 | ASM | ILRASRIM |
| 061 | ASM | ILRASRIM |

Table 37. Wait state codes to modules reference (continued)

| Wait state code (Hex) | Component | Detecting module |
|---|---|---|
| 062 | IOS | IOSRCHPR |
| 063 | ASM | IEAVNP03 IEAVNP11 IEAVNP19 ILRASRIM |
| 064 | NIP | IEAVNIPM IEAVNIP0 |
| 065 | NIP | IEAVNIPM |
| 06F | IOS | IOSVMSLG |
| 070 | IPL | IEAIPL00 |
| 071 | IPL | IEAIPL41 |
| 072 | IPL, IOS | IEAIPL00 IEAIPL02 IEAIPL30 IEAIPL41 IEAIPL40 IEAIPL43 IEAIPL46 IEAIPL70 IEAIPL71 IEAIPL99 IEAIDRIA IOSISTOR |
| 073 | IPL | IEAIPL00 |
| 074 | IPL,IOS | IEAIPL35 IEAIPL00 IEAIPL02 IEAIPL03 |
| 075 | IPL | IEAIPL00 |
| 076 | IPL | IEAIPL00 |
| 077 | | IEAIPL07 |
| 07B | NIP | IEAVNIP0 |
| 07C | NIP, service processor interface | ISNIRIM IEAIPL99 IEAVNIP0 |
| 07D | NIP | IEAVNPCF |
| 07E | Supervisor Control | IEAVESVC IEAV9SVC |
| 081 | IPL | IEAIPL00 |
| 082 | Console services | IEAVG603 IEAVN703 |
| 083 | Supervisor control | IEAVESAR |
| 084 | RTM | IGFPEMER |
| 085 | ASM | ILRASRIM ILRASRIM |
| 087 | Console services | IEAVG603 IEAVG604 IEAVG605 IEAVG610 IEAVG611 IEAVM605 IEAVM613 IEAVM616 CNZQ1CNQ IEAVMFRR CNZQ1MT2 CNZI1DLI CNZM1TIM CNZM1TSK CNZM1TST CNZQ1DCQ CNZQ1SLG IEAVN701 IEAVN703 IEECVSMA IEEVWAIT |
| 088 | IPL | IEAIPL50 |
| 08A | Console services | IEAVBLWT |
| 08B | IOS | IEAIPL40 |
| 08C | WLM | IWML2LWT |
| 08E | SRM | IEAVNP10 IRARMERR |

*Table 37. Wait state codes to modules reference (continued)*

| Wait state code (Hex) | Component | Detecting module |
|---|---|---|
| 08F | Supervisor Control | IEAVEGR |
| 09x | Loadwait | IEEVEXSN |
| 0A1 | Loadwait | IEEVEXSN |
| 0A2 | XCF | IXCI2IST IXCI2ETX IXCI2PH1 |
| 0A3 | GRS | ISGNLD |
| 0A4 | Timer supervision | IEATESC2 IEATESCH |
| 0B0 | IOS | IEAIPL43 |
| 0B1 | IOS | IOSIOFR IEAIPL43 |
| 0B2 | IOS | IEAIPL71 |
| 0B3 | IPL | IEAIPL49 |
| 0B4 | IOS | IOSIUCB |
| 0E1 | Loadwait/Restart | IEEVSTOP |
| 0E3 | VSM | IEAVNP08 |
| 0E8 | MCH | IEAVNP06 |
| 101 | VSM | IEAVGM00 |
| 102 | VSM | IEAVGM00 |
| 104 | Supervisor control | IEAVESVR |
| 110 | IOS | IOSRHREC |
| 111 | IOS | IOSRHREC |
| 112 | IOS | IOSRHREC |
| 113 | IOS | IOSRCHPR |
| 114 | IOS | IOSRCHPR |
| 115 | IOS | IECVPST IOSVDAVV IECVPST |
| 116 | IOS | IOSVRSTS |
| 140 | IOS | IECVPST |
| 200 | ALC | IEFAB4I0 IEFEB400 |
| 201 | Console services | IEAVN700 |
| 202 | Console services | IEAVN701 |
| 204 | Allocation | IEAIPL08 |
| 206 | Timer | IEAVNP21 |
| 5C7 | Loadwait/Restart | BLWLDWT BLWPTERM BLWPTSIG BLWRSTOP |
| A00 | RTM | RTM |
| A01 | MCH | IGFPMCIH |
| A02 | MCH | IGFPMAIN |

Table 37. Wait state codes to modules reference (continued)

| Wait state code (Hex) | Component | Detecting module |
|---|---|---|
| A19 | IOS | IOSRMCH |
| A1E | Timer supervision NIP | IEATESC2 IEAVRTOD IEATVTOD IEATTSCH IEATTFDH IEATESCH IEAVNIP0 |
| A1F | Timer supervision | IEATESC2 |
| A20 | RSM | IARMN |
| A21 | RSM | IARMN |
| A22 | Master scheduler | IEEVDCSR |
| A23 | MCH | IGFPMAIN |
| A24 | MCH | IGFPMAIN |
| A26 | MCH | IGFPMAIN |
| A27 | Loadwait/Restart | IEEVSTOP |
| A28 | MCH | IGFPMCIH |
| A29 | Loadwait | IGFPTERM |
| A2A | RSM | IEAVNPD8 |
| A2C | RSM | IARMN |
| A2D | RSM | IAXBI |
| A2E | RSM | IAXRR |
| A70 | Console services | IEAVBWTO |
| A7A | Service processor interface | ISNAINIT ISNATACH ISNDAMAG ISNIH ISNIRIM ISNMSI ISNRIM |
| B23 | IOCP | IEAVSTAA |
| CCC | Loadwait | IEEMPS03 |
| D0D | SMF | IEEMB829 |
| FF0 | Installation-provided | N/A |
| FF1 | Installation-provided | N/A |
| FF2 | Installation-provided | N/A |
| FF3 | Installation-provided | N/A |
| FF4 | Installation-provided | N/A |
| FF5 | Installation-provided | N/A |
| FF6 | Installation-provided | N/A |
| FF7 | Installation-provided | N/A |
| FF8 | Installation-provided | N/A |
| FF9 | Installation-provided | N/A |
| FFA | Installation-provided | N/A |

Table 37. Wait state codes to modules reference (continued)

| Wait state code (Hex) | Component | Detecting module |
|---|---|---|
| FFB | Installation-provided | N/A |
| FFC | Installation-provided | N/A |
| FFD | Installation-provided | N/A |
| FFE | Installation-provided | N/A |

# Chapter 11. MVS data areas

The following data areas are updated to support Validated Boot for z/OS. Some data areas might be abbreviated for presentation here. For complete information, see *z/OS MVS Data Areas* or the data area source files in your system's macro library.

## IHAVBA: Validated boot area

This topic describes the IHAVBA data area, which is a new macro in support of Validated Boot for z/OS.

### IHAVBA programming interface information

IHAVBA is a programming interface.

### IHAVBA heading information

| | |
|---|---|
| **Common name:** | Validated Boot Area |
| **Macro ID:** | IHAVBA |
| **DSECT name:** | VBA VB_CertExtract VB_AuditArea VB_AA_DSNE VB_AA_ModE |
| **Owning component:** | Supervisor Control (SC1C5) |
| **Eye-catcher ID:** | NONE |
| **Storage attributes:** | Subpool: 245<br>Key: 0<br>Residency: above 16M |
| **Size:** | VBA -- X'0040' bytes<br>VB_CertExtract -- X'0140' bytes<br>VB_AuditArea -- X'0070' bytes<br>VB_AA_DSNE -- X'0070' bytes<br>VB_AA_DSNE_ModE -- X'0100' bytes |
| **Created by:** | IEAIPL99 and various others |
| **Pointed to by:** | SVTVBAA (when not 0, and high bit is 0) |
| **Serialization:** | None required |
| **Function:** | Maps the interface data relevant to validated boot |

### IHAVBA mapping

*Table 38. Structure VBA*

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| 0 | (0) | STRUCTURE | 0 | VBA | SVTVBAA |
| 0 | (0) | CHARACTER | 4 | VBA_ID | |
| 4 | (4) | BITSTRING | 1 | VBA_FLAGS | |
| Bit definitions: | | | | | |
| | | 1... .... | | VBA_ENFORCE | "X'80'" |
| | | .1.. .... | | VBA_AUDIT | "X'40'" |
| 5 | (5) | BITSTRING | 1 | VBA_PAGING_FLAGS | |

Table 38. Structure VBA (continued)

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| Bit definitions: | | | | | |
| | | 1... .... | | VBA_PLPAPAGEDSSPEC | "X'80'" PLPA data set specified |
| | | .1.. .... | | VBA_SCMCANNOTHOLDLPA | "X'40'" |
| 6 | (6) | CHARACTER | 2 | | Use one of these bytes for "version" if we ever expand VBA |
| 8 | (8) | ADDRESS | 4 | VBA_AUDITAREA_ADDR | |
| 12 | (C) | ADDRESS | 4 | VBA_FIRST_GOOD_CX_ADDR | |
| 16 | (10) | SIGNED | 4 | VBA_NUM_GOOD_CX | |
| 20 | (14) | SIGNED | 4 | VBA_NUM_BAD_CX | |
| 24 | (18) | ADDRESS | 4 | VBA_FIRST_BAD_CX_ADDR | |
| 24 | (18) | X'C2C140' | 0 | VBA_ID_CHARS | "C'VBA '" |
| 64 | (40) | X'40' | 0 | VBA_LEN | "*-VBA" |

Table 39. Structure VB_CERTEXTRACT

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| 0 | (0) | STRUCTURE | 0 | VB_CERTEXTRACT | |
| 0 | (0) | CHARACTER | 4 | VB_CX_ID | |
| 4 | (4) | ADDRESS | 4 | VB_CX_NEXTADDR | |
| 8 | (8) | CHARACTER | 4 | | For IBM use only |
| 12 | (C) | SIGNED | 4 | VB_CX_NUMSUCCESSFULUSES | |
| 16 | (10) | CHARACTER | 64 | VB_CX_CERTNAME | |
| 80 | (50) | CHARACTER | 32 | VB_CX_CERTFP | |
| 112 | (70) | CHARACTER | 20 | VB_CX_KEYID | Zeroes if bad key ID length |
| 132 | (84) | CHARACTER | 2 | | For IBM use only |
| 134 | (86) | SIGNED | 2 | VB_CX_REASON_BAD | |
| 136 | (88) | CHARACTER | 8 | VB_CX_STARTTIME | First 8 bytes of ETOD |
| 144 | (90) | CHARACTER | 8 | VB_CX_EXPIRATIONTIME | First 8 bytes of ETOD |
| 152 | (98) | CHARACTER | 2 | | For IBM use only |
| 154 | (9A) | CHARACTER | 6 | | |
| 160 | (A0) | CHARACTER | 80 | VB_CX_X | Zeroes if bad key |
| 240 | (F0) | CHARACTER | 80 | VB_CX_Y | Zeroes if bad key |
| 240 | (F0) | X'C2C3E7' | 0 | VB_CX_ID_CHARS | "C'VBCX'" |
| 240 | (F0) | X'1' | 0 | VB_CX_REASON_NOTSTARTED | "1" |
| 240 | (F0) | X'2' | 0 | VB_CX_REASON_EXPIRED | "2" |
| 240 | (F0) | X'3' | 0 | VB_CX_REASON_BADKEY | "3" |
| 240 | (F0) | X'4' | 0 | VB_CX_REASON_BADKEYTYPE | "4" |
| 240 | (F0) | X'5' | 0 | VB_CX_REASON_BADKEYIDLEN | "5" |
| 240 | (F0) | X'6' | 0 | VB_CX_REASON_BADHASHTYPE | "6" |
| 240 | (F0) | X'7' | 0 | VB_CX_REASON_BADHASHLEN | "7" |
| Reasons for validation failure | | | | | |
| 240 | (F0) | X'1' | 0 | VB_VF_NOTSIGNED | "1" |
| 240 | (F0) | X'2' | 0 | VB_VF_DENOTFOUND | "2" |
| 240 | (F0) | X'3' | 0 | VB_VF_DENOTMATCH | "3" |

*Table 39. Structure VB_CERTEXTRACT (continued)*

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| 240 | (F0) | X'4' | 0 | VB_VF_SIGNOTFOUND | "4" |
| 240 | (F0) | X'5' | 0 | VB_VF_BADHASHALG | "5" |
| 240 | (F0) | X'6' | 0 | VB_VF_BADSIGALG | "6" |
| 240 | (F0) | X'7' | 0 | VB_VF_BADHASHVAL | "7" |
| 240 | (F0) | X'8' | 0 | VB_VF_NOMATCHINGKEYID | "8" |
| 240 | (F0) | X'9' | 0 | VB_VF_SIGVERFAILED | "9" |
| 240 | (F0) | X'A' | 0 | VB_VF_OVERLAYMODULE | "10" |
| 240 | (F0) | X'B' | 0 | VB_VF_BADSIGRECVERSION | "11" |
| 240 | (F0) | X'C' | 0 | VB_VF_MACHLOADERERROR | "12" |
| 240 | (F0) | X'140' | 0 | VB_CERTEXTRACT_LEN | "*-VB_CertExtract" |

*Table 40. Structure VB_AUDITAREA*

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| 0 | (0) | STRUCTURE | 0 | VB_AUDITAREA | |
| 0 | (0) | CHARACTER | 4 | VB_AA_ID | |
| 4 | (4) | SIGNED | 4 | VB_AA_NUMFAILURES | |
| 8 | (8) | CHARACTER | 4 | | For IBM use only |
| 12 | (C) | SIGNED | 4 | VB_AA_NUMFAILURES_NODSNE | Could not allocate a DSNE so just counted the failure |
| 16 | (10) | CHARACTER | 4 | | For IBM use only |
| 20 | (14) | BITSTRING | 1 | VB_AA_FLAGS0 | |
| Bit definitions: | | | | | |
| | | 1... .... | | VB_AA_NOVB | "X'80'" No validated boot possible |
| | | .1.. .... | | VB_AA_NOGOODCERTS | "X'40'" |
| 21 | (15) | BITSTRING | 1 | VB_AA_HASHTABLE_DSNE_DIM | Dimension of the array at VB_AA_HashTable_DSNE_Area |
| 22 | (16) | CHARACTER | 2 | | Use one of these bytes for "version" if we ever expand AuditArea |
| 24 | (18) | SIGNED | 4 | VB_AA_NUM_DSNES | Number of DSNE's |
| 28 | (1C) | SIGNED | 4 | VB_AA_NUM_DSNE_MODES | Number of ModE's |
| 32 | (20) | SIGNED | 4 | VB_AA_NUM_SUCCESS_IPL | Number of successful validations for fetch type IPL |
| 36 | (24) | SIGNED | 4 | VB_AA_NUM_SUCCESS_NUC | Number of successful validations for fetch type NUC |
| 40 | (28) | SIGNED | 4 | VB_AA_NUM_SUCCESS_NIP | Number of successful validations for fetch type NIP |
| 44 | (2C) | SIGNED | 4 | VB_AA_NUM_SUCCESS_LPA | Number of successful validations for fetch type LPA |
| 48 | (30) | CHARACTER | 16 | | Reserved |
| 64 | (40) | CHARACTER | 48 | | Reserved |
| 112 | (70) | CHARACTER | 1 | VB_AA_HASHTABLE_DSNE_AREA(0) | |
| | | | | | Start of array of 4-byte pointers with the dimension in the "_Dim" field. Each pointer, when not 0, points to a chain of areas each mapped by VB_AA_DSNE |
| 112 | (70) | X'C2C1C1' | 0 | VB_AA_ID_CHARS | "C'VBAA'" |
| 112 | (70) | X'70' | 0 | VB_AUDITAREA_LEN | "*-VB_AuditArea" |

Table 41. Structure VB_AA_DSNE

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| 0 | (0) | STRUCTURE | 0 | VB_AA_DSNE | |
| 0 | (0) | CHARACTER | 8 | VB_AA_DSNE_ID | |
| 8 | (8) | ADDRESS | 4 | VB_AA_DSNE_NEXT_ADDR | |
| 12 | (C) | CHARACTER | 4 | | For IBM use only |
| 16 | (10) | CHARACTER | 44 | VB_AA_DSNE_DSNAME | A value of "*" indicates that this entry applies regardless of data set name, such as "no certificate entries". A value of zeroes applies only for IEAIPL00 and indicates not known |
| 60 | (3C) | CHARACTER | 6 | VB_AA_DSNE_VOLID | The volume ID, unless DSName is zeroes or *, indicating not known. |
| 66 | (42) | CHARACTER | 2 | | Use one of these bytes for "version" if we ever expand AA_DSNE |
| 68 | (44) | SIGNED | 4 | VB_AA_DSNE_NUMFAILURES | |
| 72 | (48) | CHARACTER | 4 | | For IBM use only |
| 76 | (4C) | SIGNED | 4 | VB_AA_DSNE_NUMFAILURES_NOMODE | |
| | | | | | Could not allocate a ModE so just counted the failure |
| 80 | (50) | CHARACTER | 4 | | For IBM use only |
| 84 | (54) | CHARACTER | 3 | | |
| 87 | (57) | BITSTRING | 1 | VB_AA_DSNE_HASHTABLE_MODE_DIM | |
| | | | | | Dimension of the array at VB_AA_DSNE_HashTable_Mode_Area |
| 88 | (58) | CHARACTER | 16 | VB_AA_DSNE_FAILTIME | First fail time |
| 104 | (68) | SIGNED | 4 | VB_AA_DSNE_NUM_DSNE_MODES | Number of ModE's for this DSNE |
| 108 | (6C) | CHARACTER | 4 | | |
| 112 | (70) | CHARACTER | 1 | VB_AA_DSNE_HASHTABLE_MODE_AREA(0) | |
| | | | | | Start of array of 4-byte pointers with the dimension in the "_Dim" field. Each pointer, when not 0, points to a chain of areas each mapped by VB_AA_DSNE_ModE |
| 112 | (70) | X'C2C1C1' | 0 | VB_AA_DSNE_ID_CHARS_0TO3 | "C'VBAA'" This is the first 4-byte segment of an 8-byte constant. |
| 112 | (70) | X'E2D5C5' | 0 | VB_AA_DSNE_ID_CHARS_4TO7 | "C'DSNE'" This is the second 4-byte segment of an 8-byte constant. |
| 112 | (70) | X'70' | 0 | VB_AA_DSNE_LEN | "*-VB_AA_DSNE" |

Table 42. Structure VB_AA_DSNE_MODE

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| 0 | (0) | STRUCTURE | 0 | VB_AA_DSNE_MODE | |
| 0 | (0) | CHARACTER | 8 | VB_AA_DSNE_MODE_ID | |
| 8 | (8) | ADDRESS | 4 | VB_AA_DSNE_MODE_NEXT_ADDR | |
| 12 | (C) | CHARACTER | 4 | | For IBM use only |
| 16 | (10) | CHARACTER | 8 | | Use one of these bytes for "version" if we ever expand AA_DSNE_ModE |
| 24 | (18) | CHARACTER | 8 | VB_AA_DSNE_MODE_MODNAME | |
| 32 | (20) | BITSTRING | 1 | VB_AA_DSNE_MODE_FLAGS | |
| Bit definitions: | | | | | |

Table 42. Structure VB_AA_DSNE_MODE (continued)

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| | | 1... .... | | VB_AA_DSNE_MODE_FOUNDSIG | "X'80'" Found the signature record for this module |
| | | .1.. .... | | VB_AA_DSNE_MODE_FOUNDCERT | "X'40'" Found a certificate with a key ID matching the key ID in the signature record. That certificate will be used to attempt validation (as will any other certificate with an identical key ID) |
| | | ..1. .... | | VB_AA_DSNE_MODE_HAVEMACHLOADERERRORS | "X'20'" MachLoaderErrors has information |
| 33 | (21) | BITSTRING | 1 | VB_AA_DSNE_MODE_FETCHTYPE | |
| 34 | (22) | SIGNED | 2 | VB_AA_DSNE_MODE_FAILURE_REASON | |
| 36 | (24) | SIGNED | 4 | VB_AA_DSNE_MODE_NUMFAILURES | |
| 40 | (28) | CHARACTER | 16 | VB_AA_DSNE_MODE_FAILTIME | First fail time |
| 56 | (38) | CHARACTER | 64 | VB_AA_DSNE_MODE_CX_CERTNAME | Valid only when FoundCert |
| 120 | (78) | CHARACTER | 32 | VB_AA_DSNE_MODE_CERTFP | Valid only when FoundSig |
| 152 | (98) | CHARACTER | 20 | VB_AA_UNION | |
| 152 | (98) | CHARACTER | 20 | VB_AA_DSNE_MODE_KEYID | Valid only when FoundSig |
| 152 | (98) | CHARACTER | 6 | VB_AA_DSNE_MODE_MACHLOADERERRORS | Errors found by machine loader. Valid only when haveMachLoaderErrors |
| 152 | (98) | BITSTRING | 4 | VB_AA_DSNE_MODE_MLE_SCLAFED | Secure code loading attribute facilities error details |
| 156 | (9C) | BITSTRING | 2 | VB_AA_DSNE_MODE_MLE_IIEI | IPL Information error indicators |
| 172 | (AC) | CHARACTER | 4 | | For IBM use only |
| 176 | (B0) | CHARACTER | 16 | VB_AA_DSNE_MODE_SIGNTIME | Valid only when FoundSig |
| 192 | (C0) | CHARACTER | 64 | VB_AA_DSNE_MODE_DIGEST | The expected hash value. Valid only when FoundSig |
| 192 | (C0) | X'C2C1C1' | 0 | VB_AA_DSNE_MODE_ID_CHARS_0TO3 | "C'VBAA'" This is the first 4-byte segment of an 8-byte constant. |
| 192 | (C0) | X'D6C4C5' | 0 | VB_AA_DSNE_MODE_ID_CHARS_4TO7 | "C'MODE'" This is the second 4-byte segment of an 8-byte constant. |
| 192 | (C0) | X'0' | 0 | VB_AA_DSNE_MODE_FETCHTYPE_MIN | "0" |
| 192 | (C0) | X'0' | 0 | VB_AA_DSNE_MODE_FETCHTYPE_IPL | "0" |
| 192 | (C0) | X'1' | 0 | VB_AA_DSNE_MODE_FETCHTYPE_NUC | "1" |
| 192 | (C0) | X'2' | 0 | VB_AA_DSNE_MODE_FETCHTYPE_NIP | "2" |
| 192 | (C0) | X'3' | 0 | VB_AA_DSNE_MODE_FETCHTYPE_LPA | "3" |
| 192 | (C0) | X'3' | 0 | VB_AA_DSNE_MODE_FETCHTYPE_MAX | "3" |
| 192 | (C0) | X'100' | 0 | VB_AA_DSNE_MODE_LEN | "*-VB_AA_DSNE_ModE" |

Table 43. Cross Reference for IHAVBA

| Name | Offset | Hex Tag |
|---|---|---|
| VB_AA_DSNE | 0 | |
| VB_AA_DSNE_DSNAME | 10 | |
| VB_AA_DSNE_FAILTIME | 58 | |
| VB_AA_DSNE_HASHTABLE_MODE_AREA | 70 | |
| VB_AA_DSNE_HASHTABLE_MODE_DIM | 57 | |
| VB_AA_DSNE_ID | 0 | |
| VB_AA_DSNE_ID_CHARS_0TO3 | 70 | C2C1C1 |
| VB_AA_DSNE_ID_CHARS_4TO7 | 70 | E2D5C5 |
| VB_AA_DSNE_LEN | 70 | 70 |
| VB_AA_DSNE_MODE | 0 | |
| VB_AA_DSNE_MODE_CERTFP | 78 | |
| VB_AA_DSNE_MODE_CX_CERTNAME | 38 | |
| VB_AA_DSNE_MODE_DIGEST | C0 | |
| VB_AA_DSNE_MODE_FAILTIME | 28 | |
| VB_AA_DSNE_MODE_FAILURE_REASON | 22 | |
| VB_AA_DSNE_MODE_FETCHTYPE | 21 | |
| VB_AA_DSNE_MODE_FETCHTYPE_IPL | C0 | 0 |
| VB_AA_DSNE_MODE_FETCHTYPE_LPA | C0 | 3 |
| VB_AA_DSNE_MODE_FETCHTYPE_MAX | C0 | 3 |
| VB_AA_DSNE_MODE_FETCHTYPE_MIN | C0 | 0 |
| VB_AA_DSNE_MODE_FETCHTYPE_NIP | C0 | 2 |
| VB_AA_DSNE_MODE_FETCHTYPE_NUC | C0 | 1 |
| VB_AA_DSNE_MODE_FLAGS | 20 | |
| VB_AA_DSNE_MODE_FOUNDCERT | 20 | 40 |
| VB_AA_DSNE_MODE_FOUNDSIG | 20 | 80 |
| VB_AA_DSNE_MODE_HAVEMACHLOADERERRORS | 20 | 20 |
| VB_AA_DSNE_MODE_ID | 0 | |
| VB_AA_DSNE_MODE_ID_CHARS_0TO3 | C0 | C2C1C1 |
| VB_AA_DSNE_MODE_ID_CHARS_4TO7 | C0 | D6C4C5 |
| VB_AA_DSNE_MODE_KEYID | 98 | |
| VB_AA_DSNE_MODE_LEN | C0 | 100 |
| VB_AA_DSNE_MODE_MACHLOADERERRORS | 98 | |
| VB_AA_DSNE_MODE_MLE_IIEI | 9C | |
| VB_AA_DSNE_MODE_MLE_SCLAFED | 98 | |
| VB_AA_DSNE_MODE_MODNAME | 18 | |
| VB_AA_DSNE_MODE_NEXT_ADDR | 8 | |
| VB_AA_DSNE_MODE_NUMFAILURES | 24 | |
| VB_AA_DSNE_MODE_SIGNTIME | B0 | |
| VB_AA_DSNE_NEXT_ADDR | 8 | |

Table 43. Cross Reference for IHAVBA (continued)

| Name | Offset | Hex Tag |
|---|---|---|
| VB_AA_DSNE_NUM_DSNE_MODES | 68 | |
| VB_AA_DSNE_NUMFAILURES | 44 | |
| VB_AA_DSNE_NUMFAILURES_NOMODE | 4C | |
| VB_AA_DSNE_VOLID | 3C | |
| VB_AA_FLAGS0 | 14 | |
| VB_AA_HASHTABLE_DSNE_AREA | 70 | |
| VB_AA_HASHTABLE_DSNE_DIM | 15 | |
| VB_AA_ID | 0 | |
| VB_AA_ID_CHARS | 70 | C2C1C1 |
| VB_AA_NOGOODCERTS | 14 | 40 |
| VB_AA_NOVB | 14 | 80 |
| VB_AA_NUM_DSNE_MODES | 1C | |
| VB_AA_NUM_DSNES | 18 | |
| VB_AA_NUM_SUCCESS_IPL | 20 | |
| VB_AA_NUM_SUCCESS_LPA | 2C | |
| VB_AA_NUM_SUCCESS_NIP | 28 | |
| VB_AA_NUM_SUCCESS_NUC | 24 | |
| VB_AA_NUMFAILURES | 4 | |
| VB_AA_NUMFAILURES_NODSNE | C | |
| VB_AA_UNION | 98 | |
| VB_AUDITAREA | 0 | |
| VB_AUDITAREA_LEN | 70 | 70 |
| VB_CERTEXTRACT | 0 | |
| VB_CERTEXTRACT_LEN | F0 | 140 |
| VB_CX_CERTFP | 50 | |
| VB_CX_CERTNAME | 10 | |
| VB_CX_EXPIRATIONTIME | 90 | |
| VB_CX_ID | 0 | |
| VB_CX_ID_CHARS | F0 | C2C3E7 |
| VB_CX_KEYID | 70 | |
| VB_CX_NEXTADDR | 4 | |
| VB_CX_NUMSUCCESSFULUSES | C | |
| VB_CX_REASON_BAD | 86 | |
| VB_CX_REASON_BADHASHLEN | F0 | 7 |
| VB_CX_REASON_BADHASHTYPE | F0 | 6 |
| VB_CX_REASON_BADKEY | F0 | 3 |
| VB_CX_REASON_BADKEYIDLEN | F0 | 5 |
| VB_CX_REASON_BADKEYTYPE | F0 | 4 |
| VB_CX_REASON_EXPIRED | F0 | 2 |

*Table 43. Cross Reference for IHAVBA (continued)*

| Name | Offset | Hex Tag |
|---|---|---|
| VB_CX_REASON_NOTSTARTED | F0 | 1 |
| VB_CX_STARTTIME | 88 | |
| VB_CX_X | A0 | |
| VB_CX_Y | F0 | |
| VB_VF_BADHASHALG | F0 | 5 |
| VB_VF_BADHASHVAL | F0 | 7 |
| VB_VF_BADSIGALG | F0 | 6 |
| VB_VF_BADSIGRECVERSION | F0 | B |
| VB_VF_DENOTFOUND | F0 | 2 |
| VB_VF_DENOTMATCH | F0 | 3 |
| VB_VF_MACHLOADERERROR | F0 | C |
| VB_VF_NOMATCHINGKEYID | F0 | 8 |
| VB_VF_NOTSIGNED | F0 | 1 |
| VB_VF_OVERLAYMODULE | F0 | A |
| VB_VF_SIGNOTFOUND | F0 | 4 |
| VB_VF_SIGVERFAILED | F0 | 9 |
| VBA | 0 | |
| VBA_AUDIT | 4 | 40 |
| VBA_AUDITAREA_ADDR | 8 | |
| VBA_ENFORCE | 4 | 80 |
| VBA_FIRST_BAD_CX_ADDR | 18 | |
| VBA_FIRST_GOOD_CX_ADDR | C | |
| VBA_FLAGS | 4 | |
| VBA_ID | 0 | |
| VBA_ID_CHARS | 18 | C2C140 |
| VBA_LEN | 40 | 40 |
| VBA_NUM_BAD_CX | 14 | |
| VBA_NUM_GOOD_CX | 10 | |
| VBA_PAGING_FLAGS | 5 | |
| VBA_PLPAPAGEDSSPEC | 5 | 80 |
| VBA_SCMCANNOTHOLDLPA | 5 | 40 |

# Chapter 12. RACF data areas

The following RACF data areas are updated to support Validated Boot for z/OS. Some data areas are abbreviated for presentation here. For complete information, see *z/OS Security Server RACF Data Areas*.

## COMP: Common SAF/RACF Parameter List for z/OS UNIX System Services

This topic contains details for changes to the COMP data area for Validated Boot for z/OS. It contains only the section that was changed. For the complete data area, see *z/OS Security Server RACF Data Areas*.

### COMP programming interface information

COMP is a programming interface.

### COMP mapping

*Table 44. Structure COMP*

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| PGSN | | | | | |
| 0 | (0) | STRUCTURE | 16 | PGSN | Mapping for PGSN |
| 0 | (0) | ADDRESS | 4 | PGSN_NUM_PARMS@ | Address of a fullword containing the total number of parameters included in COMP and PGSN. |
| 4 | (4) | ADDRESS | 4 | PGSN_FUNC@ | Address of 2-byte function code. Constants for the function codes are supplied below. |
| 8 | (8) | ADDRESS | 4 | PGSN_FUNC_PARML@ | Address of the function-specific parameter list corresponding to the function code. See *z/OS Security Server RACF Callable Services* for function specific parameter lists for callable service R_PgmSignVer. |
| 12 | (C) | ADDRESS | 4 | PGSN_FUNC_ATTRS@ | Address of a 4-byte variable that contains the attribute flags for the service. |
| PKIS | | | | | |
| 0 | (0) | STRUCTURE | 28 | PKIS | Mapping for PKIS |
| 0 | (0) | ADDRESS | 4 | PKIS_NUM_PARMS@ | Address of a 4-byte variable that contains the number of parameters that follow in the non-request specific portion of the R_PKIServ callable service Parameter List |

*Table 44. Structure COMP (continued)*

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| 4 | (4) | ADDRESS | 4 | PKIS_FUNC@ | Address of a 2-Byte variable that contains the code of the requested function. Constants for the function codes codes are declared below |
| 8 | (8) | ADDRESS | 4 | PKIS_ATTRIBUTES@ | Address of a 4-Byte variable that contains attribute flags for the service |
| 12 | (C) | ADDRESS | 4 | PKIS_LOG_STRING@ | Address of a Variable-Length area that contains the LOG string to be passed to RACROUTE (1 byte for the length followed by up to 255 bytes for the LOG string itself) |
| 16 | (10) | ADDRESS | 4 | PKIS_PARM_VER@ | Address of a 4-Byte variable that contains the version number of the Function Specific Parameter List (PKIS_FUNC_PARML@) |
| 20 | (14) | ADDRESS | 4 | PKIS_FUNC_PARML@ | Address of the FSPL - Function Specific Parameter List (FSPL = the Parameter List that corresponds to the Function Code) |
| | | .... .... | | PKIS_LAST_PARM | Variable length parameter list. This is the last parameter |
| 24 | (18) | ADDRESS | 4 | PKIS_CA_DOMAIN@ | Address of the name of the PKI Services certificate authority instance to be invoked. |

## COMY: 64-bit enabled SAF callable services

This topic contains details for changes to the COMY data area for Validated Boot for z/OS. It contains only the section that was changed. For the complete data area, see *z/OS Security Server RACF Data Areas*.

### COMY programming interface information

COMY is a programming interface.

### COMY heading information

| | |
|---|---|
| **Common name:** | SAF Common Security Parameter List (64 bit) |
| **Macro ID:** | IRRPCOMY |
| **DSECT name:** | COMY, PGSN64, RAUX64, PKIS64 |
| **Owning component:** | Resource Access Control Facility (SC1BN) |
| **Eye-catcher ID:** | None |

**Storage attributes:**

**Subpool**
N/A

**Key**
Any

**Residency**
Invoker's primary address space

**Size:**

| **Section** | **Size** |
|---|---|
| **COMY** | 56 bytes |
| **PGSN64** | 32 bytes |
| **PKIS64** | 60 bytes |
| **RAUX64** | 80 bytes |

**Created by:** Invoker of 64-bit enabled callable services

**Pointed to by:** Address of COMY is passed in register 1 when invoking 64-bit enabled callable services

**Serialization:** None

**Function:** Maps the common input parameter list for the 64-bit RACF and SAF callable services routers

## COMY mapping

*Table 45. Structure COMY*

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| 0 | (0) | STRUCTURE | 0 | COMY | SAF enabled callable services. |
| 0 | (0) | ADDRESS | 8 | COMY_WORKA_STOR@ | Address of 1024 byte work area |
| 8 | (8) | ADDRESS | 8 | COMY_SAFRC_ALET@ | Address of ALET for SAF return code |
| 16 | (10) | ADDRESS | 8 | COMY_SAFRC_STOR@ | Address of SAF return code |
| 24 | (18) | ADDRESS | 8 | COMY_RACRC_ALET@ | Address of ALET for RACF return code |
| 32 | (20) | ADDRESS | 8 | COMY_RACRC_STOR@ | Address of RACF return code |
| 40 | (28) | ADDRESS | 8 | COMY_RACSC_ALET@ | Address of ALET for RACF reason code |
| 48 | (30) | ADDRESS | 8 | COMY_RACSC_STOR@ | Address of RACF reason code |
| PGSN64 | | | | | |
| 0 | (0) | STRUCTURE | 24 | PGSN64 | Mapping for PGSN64 |
| 0 | (0) | ADDRESS | 8 | PGSN64_NUM_PARMS@ | Address of a fullword containing the total number of parameters included in COMY and PGSN64. |

*Table 45. Structure COMY (continued)*

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| 8 | (8) | ADDRESS | 8 | PGSN64_FUNC@ | Address of a 2-byte function code. See data area COMP for the function code constants. |
| 16 | (10) | ADDRESS | 8 | PGSN64_FUNC_PARML@ | Address of the function specific parameter list corresponding to the function code. See *z/OS Security Server RACF Callable Services* for function specific parameter lists for callable service R_PgmSignVer. |
| 24 | (18) | ADDRESS | 8 | PGSN64_FUNC_ATTR@ | Address of a 4-byte variable that contains the attribute flags for the service. |

RAUX64

| | | | | | |
|---|---|---|---|---|---|
| 0 | (0) | STRUCTURE | 152 | RAUX64 | |
| 0 | (0) | ADDRESS | 8 | RAUX64_NUM_PARMS@ | Address of a fullword containing the total number of parameters included in COMY and RAUX64. |
| ... | ... | ... | | ... | ... |

## RCVT: RACF Communication Vector Table

This topic contains details for changes to the RCVT data area for Validated Boot for z/OS. It contains only the section that was changed. For the complete data area, see *z/OS Security Server RACF Data Areas*.

### RCVT programming interface information

RCVT is **NOT a programming interface.** The following fields are the only intended Programming Interfaces in RCVT:

- RCVT
- RCVTAPTR
- RCVTCDTL
- RCVTDATP
- RCVTDNL
- RCVTENVP
- RCVTFLGS
- RCVTFLG1
- RCVTFLG3
- RCVTFRCP
- RCVTGENT
- RCVTGLBL
- RCVTID
- RCVTIDPV
- RCVTINAC

- RCVTISTL
- RCVTJALL
- RCVTJCHK
- RCVTJSYS
- RCVTJUND
- RCVTJXAL
- RCVTLNOD
- RCVTMFLG
- RCVTMFL1
- RCVTML2F
- RCVTPALG
- RCVTPINV
- RCVTPNL0
- RCVTPTGN
- RCVTRCVI
- RCVTRELS
- RCVTREXP
- RCVTRL
- RCVTRNA
- RCVTROFF
- RCVTRVOK
- RCVTSTAT
- RCVTSTA1
- RCVTTAPE
- RCVTTDSN
- RCVTVERS
- RCVTVRN
- RCVTVRMN
- RCVTWARN
- RCVTWUID

**Application Programmers:**

The RCVT fields listed above are Programming Interfaces for input only, with the following exceptions:

- RCVTISTL and RCVTAPTR can be both input and output
- RCVTREXP and RCVTFRCP are not part of the application programming interface.

**Notes:**

1. The 118th bit of the RCVTVCPR field is a programming interface for input only. It can be used to quickly check if the SECLABEL class is active. If the bit is on, the class is active.
2. For external security managers (ESMs) such as RACF or ESMs that are functionally compatible with RACF: The RCVT fields listed above are Programming Interfaces for both input and output. The ESM is responsible for creating the RCVT, attaching it to the communication vector table (CVT), and putting appropriate data into these fields in order to be compatible with RACF and the way that IBM products use the RCVT.

## RCVT heading information

| | |
|---|---|
| **Common name:** | RACF communication vector table |
| **Macro ID:** | ICHPRCVT |
| **DSECT name:** | RCVT |
| **Owning component:** | Resource Access Control Facility (SC1BN) |
| **Eye-catcher ID:** | RCVT (Offset: 0, Length: 4) |
| **Storage attributes:** | **Subpool** SQA **Key** 0 |
| **Size:** | 2308 bytes |
| **Created by:** | RACF initialization or equivalent |
| **Pointed to by:** | CVTRAC |
| **Serialization:** | None |
| **Function:** | Communication area for information global to RACF functions (or equivalent product functions) |

## RCVT mapping

*Table 46. Structure RCVT*

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| 0 | (0) | STRUCTURE | 2038 | RCVT | LOCATED THROUGH CVT |
| 0 | (0) | CHARACTER | 4 | RCVTID | EBCDIC ID |
| 4 | (4) | ADDRESS | 4 | RCVTDCB | PTR DCB OF RACF DATA SET |
| 8 | (8) | ADDRESS | 4 | RCVTDEB | PTR DEB OF RACF DATA SET |
| 12 | (C) | ADDRESS | 4 | RCVTINDX | PTR RACF RESIDENT INDEX TABLE OR ZERO IF NO INDEX BLOCKS RESIDENT |
| ... | ... | ... | ... | ... | ... |
| 633 | (279) | BITSTRING | 1 | RCVTFLG3 | Miscellaneous flags |
| | | 1... .... | | RCVTDCDT | Dynamic CDT is active |
| | | .1.. .... | | RCVTPLC | Allow lower case passwords |
| | | ..1. .... | | RCVTCFLD | Custom Fields are in effect |
| | | ...1 .... | | RCVTAUTU | Authority used is available to authorization exits |
| | | .... 1... | | RCVTPSC | Special characters are allowed in passwords |
| | | .... .1.. | | RCVTXPWD | Extended password support is available |
| | | .... ..1. | | RCVTMFA | MFA functions are available |

*Table 46. Structure RCVT (continued)*

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| | | .... ...1 | | RCVTMAIL | E-mail support available |
| 634 | (27A) | SIGNED | 1 | RCVTPMIN | Minimum days between password changes |
| 635 | (27B) | UNSIGNED | 1 | RCVTPALG | Password algorithm in effect: 0 = Existing algorithm as indicated by ICHDEX01 (masking, DES, or installation-defined) 1 = KDFAES |
| 636 | (27C) | UNSIGNED | 2 | RCVTPMEM | Password algorithm memory factor. |
| 638 | (27E) | UNSIGNED | 2 | RCVTPREP | Password algorithm iteration factor. |
| 640 | (280) | BITSTRING | 1 | RCVTFLG4 | Function availability bits |
| | | 1... .... | | RCVTRPFF | Indicates that the R_Password fast-fail option is available |
| | | .1.. .... | | RCVTMFA3 | MFA3 Functions (OA20930) are available. |
| | | ..1. .... | | RCVTIDT | IDT Functions (OA55926) are available. |
| | | ...1 .... | | RCVTEPT | Enhanced PassTicket Functions (OA59196) are available. |
| | | .... 1... | | RCVTPHIA | Phrase interval functions (OA61951) are available. |
| | | .... .1.. | | * | Reserved. |
| | | .... ..1. | | RCVTVABT | Validated Boot for z/OS functions (OA63507) are available. |
| | | .... ...1 | | * | Reserved |
| 696 | (2B8) | CHARACTER | 8 | RCVTJSYS | USER-ID from the SETROPTS command JES(NJEUSERID(user-id)) |
| 704 | (2C0) | CHARACTER | 8 | RCVTJUND | USER-ID from the SETROPTS command JES(UNDEFINEDUSER(user-id)) |
| 712 | (2C8) | ADDRESS | 4 | RCVTTMP2 | ADDRESS OF RDS TEMPLATES |
| 716 | (2CC) | ADDRESS | 4 | RCVTRCK4 | ADDRESS OF IRRRCK04 |
| 720 | (2D0) | ADDRESS | 4 | RCVTSVC0 | ADDRESS OF ICHSVC00 |
| 724 | (2D4) | ADDRESS | 4 | RCVTPTGN | ADDRESS OF THE PASSTICKET ROUTINE |
| 728 | (2D8) | ADDRESS | 4 | RCVTFRX4 | ADDRESS OF FASTAUTH POST-PROCESSING INSTALLATION EXIT FOR DATASPACE (ICHRFX04) |
| 732 | (2DC) | ADDRESS | 4 | RCVTDX11 | ADDRESS OF ICHDEX11 |
| 736 | (2E0) | ADDRESS | 4 | RCVTXLT0 | ADDRESS OF IRRRXT02 |

*Table 46. Structure RCVT (continued)*

| Offset Dec | Offset Hex | Type | Len | Name(Dim) | Description |
|---|---|---|---|---|---|
| 740 | (2E4) | ADDRESS | 4 | RCVTGLS6 | ADDRESS OF ICHGLS06 |
| 744 | (2E8) | ADDRESS | 4 | RCVTDPTB | ADDRESS OF DYNAMIC PARSE TABLE |
| 748 | (2EC) | ADDRESS | 4 | RCVTRCK2 | ADDRESS OF IRRRCK02 |
| 752 | (2F0) | ADDRESS | 4 | RCVTRX10 | Address of IRRRXT10 |
| 756 | (2F4) | ADDRESS | 4 | RCVTRX11 | Address of IRRRXT11 |
| 760 | (2F8) | ADDRESS | 4 | RCVTDSPC | Address of IRRDSP00 |
| 764 | (2FC) | BITSTRING | 1 | RCVTFL2X | RACF SETROPTS options |
| | | 1... .... | | RCVTCMPM | SETROPTS COMPATMODE IS ACTIVE |
| | | .1.. .... | | RCVTMLSF | 1 - SETROPTS MLS (FAILURES) IS IN EFFECT<br>0 - SETROPTS MLS (WARNING) IS IN EFFECT |
| | | ..1. .... | | RCVTMLAF | 1 - SETROPTS MLACTIVE (FAILURES) IS IN EFFECT<br>0 - SETROPTS MLACTIVE (WARNING) IS IN EFFECT |
| | | ...1 .... | | RCVTCATF | 1 - SETROPTS CATDSNS (FAILURES) IS IN EFFECT<br>0 - SETROPTS CATDSNS (WARNING) IS IN EFFECT |
| | | .... 1... | | RCVTAAPL | SETROPTS APPLAUDIT IS ACTIVE |
| | | .... .1.. | | RCVTNADC | SETROPTS NOADDCREATOR IS IN EFFECT |
| | | .... ..1. | | RCVTGNOE | SETROPTS ENHANCEDGENERICOWNER -ON if active. RCVTGNOW also ON if RCVTGNOE is ON. |
| | | .... ...1 | | * | Reserved |
| 765 | (2FD) | BITSTRING | 1 | RCVTNJEF | NJE Flags |
| | | 1... .... | | RCVTJWTO | Flag indicating WTO has been issued for NJE, if "ON" - (1) |
| | | .111 1111 | | * | Reserved |
| ... | ... | ... | | ... | ... |
| 2308 | (904) | CHARACTER | | * | END OF RCVT |

# Chapter 13. Load module formats

This topic contains general-use programming interface and associated guidance information.

This topic contains load record formats (see "SYM record (load module)" through "Record format of load module IDRs-part 3").

## Input conventions

Load modules to be processed in a single execution of the binder must conform with a number of input conventions. Violations of the following are treated as errors by the binder:

- All the ESD records must precede the text records.
- The end of every load module must be marked by an EOM flag.
- RLD items must appear after the text record containing the adcons which they describe.
- All SYM records must be placed at the beginning of the load module.
- During a single execution of the binder, if two or more control sections having the same name are read in, only the first control section is accepted; the subsequent control sections are deleted
- The binder interprets common (CM) entries in the ESD (blank or with the same name) as references to a single control section whose length is the maximum length specified in the CM items of that name (or blank). No text can be contained in a common control section

## Record formats

Figure 1 on page 129 through Figure 7 on page 135 are the load module record formats for the linkage editor.

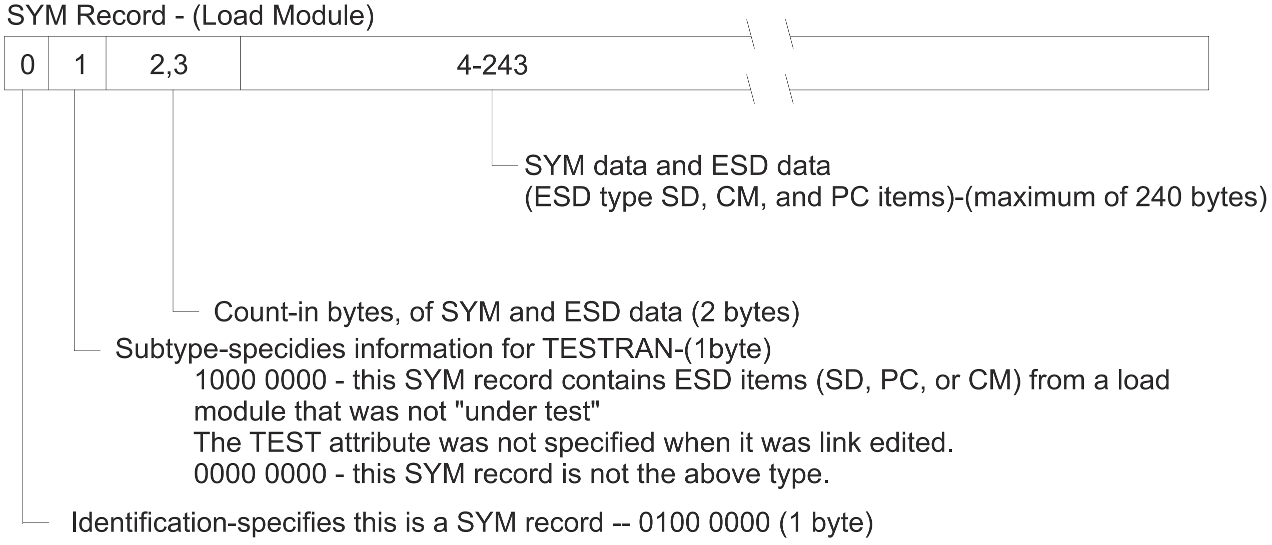
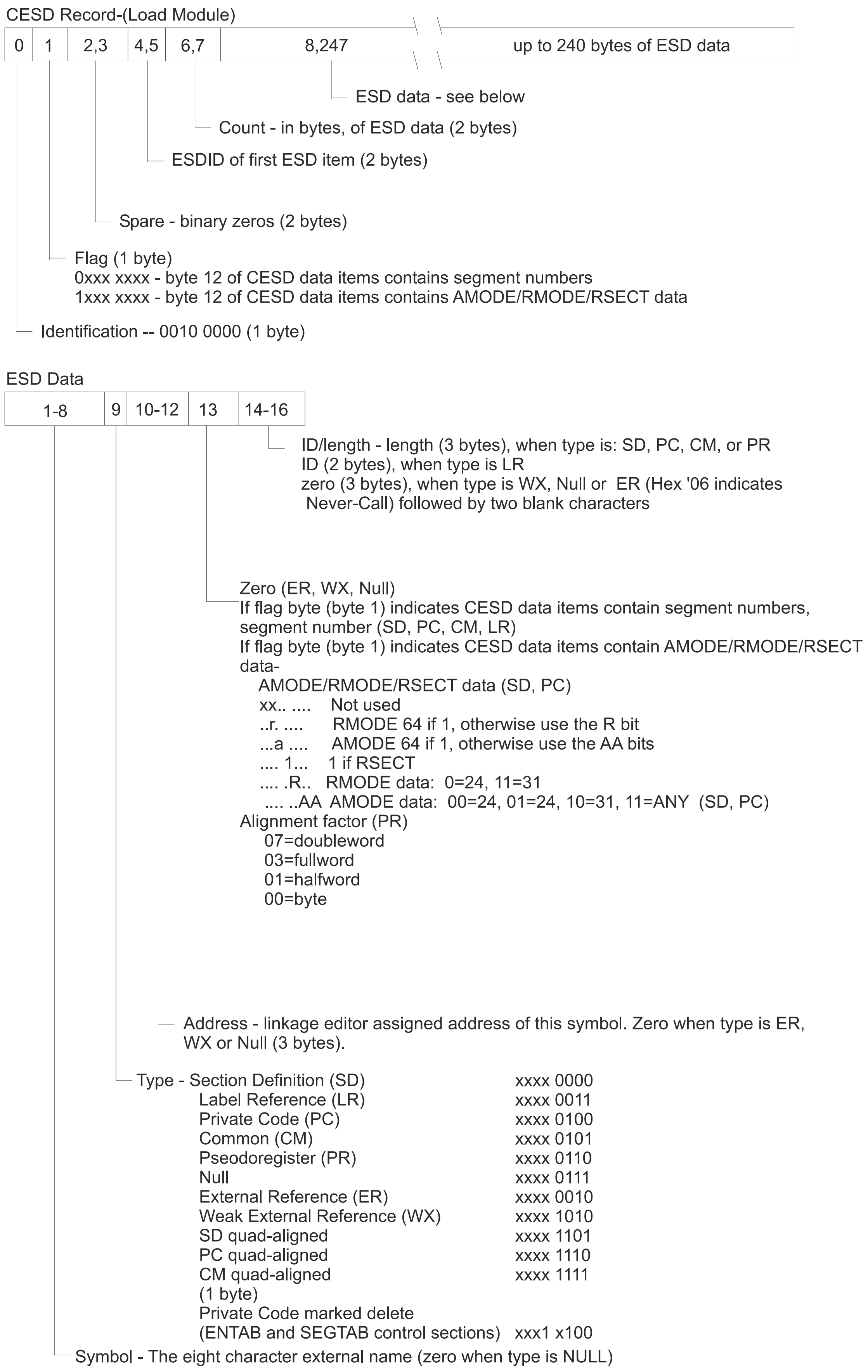

SYM Record - (Load Module)
0
1
2,3
4-243
SYM data and ESD data
(ESD type SD, CM, and PC items)-(maximum of 240 bytes)
Count-in bytes, of SYM and ESD data (2 bytes)
Subtype-specidies information for TESTRAN-(1byte)
1000 0000 - this SYM record contains ESD items (SD, PC, or CM) from a load module that was not "under test"
The TEST attribute was not specified when it was link edited.
0000 0000 - this SYM record is not the above type.
Identification-specifies this is a SYM record -- 0100 0000 (1 byte)

*Figure 1. SYM record (load module)*



CESD Record-(Load Module)

| 0 | 1 | 2,3 | 4,5 | 6,7 | 8,247 | up to 240 bytes of ESD data |
|---|---|---|---|---|---|---|

ESD data - see below

Count - in bytes, of ESD data (2 bytes)

ESDID of first ESD item (2 bytes)

Spare - binary zeros (2 bytes)

Flag (1 byte)
0xxx xxxx - byte 12 of CESD data items contains segment numbers
1xxx xxxx - byte 12 of CESD data items contains AMODE/RMODE/RSECT data

Identification -- 0010 0000 (1 byte)

ESD Data

| 1-8 | 9 | 10-12 | 13 | 14-16 |
|---|---|---|---|---|

ID/length - length (3 bytes), when type is: SD, PC, CM, or PR
ID (2 bytes), when type is LR
zero (3 bytes), when type is WX, Null or ER (Hex '06 indicates Never-Call) followed by two blank characters

Zero (ER, WX, Null)
If flag byte (byte 1) indicates CESD data items contain segment numbers, segment number (SD, PC, CM, LR)
If flag byte (byte 1) indicates CESD data items contain AMODE/RMODE/RSECT data-
AMODE/RMODE/RSECT data (SD, PC)

```
xx.. ....   Not used
..r. ....   RMODE 64 if 1, otherwise use the R bit
...a ....   AMODE 64 if 1, otherwise use the AA bits
.... 1...   1 if RSECT
.... .R..   RMODE data:  0=24, 11=31
.... ..AA   AMODE data:  00=24, 01=24, 10=31, 11=ANY  (SD, PC)
```

Alignment factor (PR)
07=doubleword
03=fullword
01=halfword
00=byte

Address - linkage editor assigned address of this symbol. Zero when type is ER, WX or Null (3 bytes).

| | |
|---|---|
| Type - Section Definition (SD) | xxxx 0000 |
| Label Reference (LR) | xxxx 0011 |
| Private Code (PC) | xxxx 0100 |
| Common (CM) | xxxx 0101 |
| Pseodoregister (PR) | xxxx 0110 |
| Null | xxxx 0111 |
| External Reference (ER) | xxxx 0010 |
| Weak External Reference (WX) | xxxx 1010 |
| SD quad-aligned | xxxx 1101 |
| PC quad-aligned | xxxx 1110 |
| CM quad-aligned | xxxx 1111 |
| (1 byte) | |
| Private Code marked delete (ENTAB and SEGTAB control sections) | xxx1 x100 |

Symbol - The eight character external name (zero when type is NULL)

*Figure 2. CESD record (load module)*

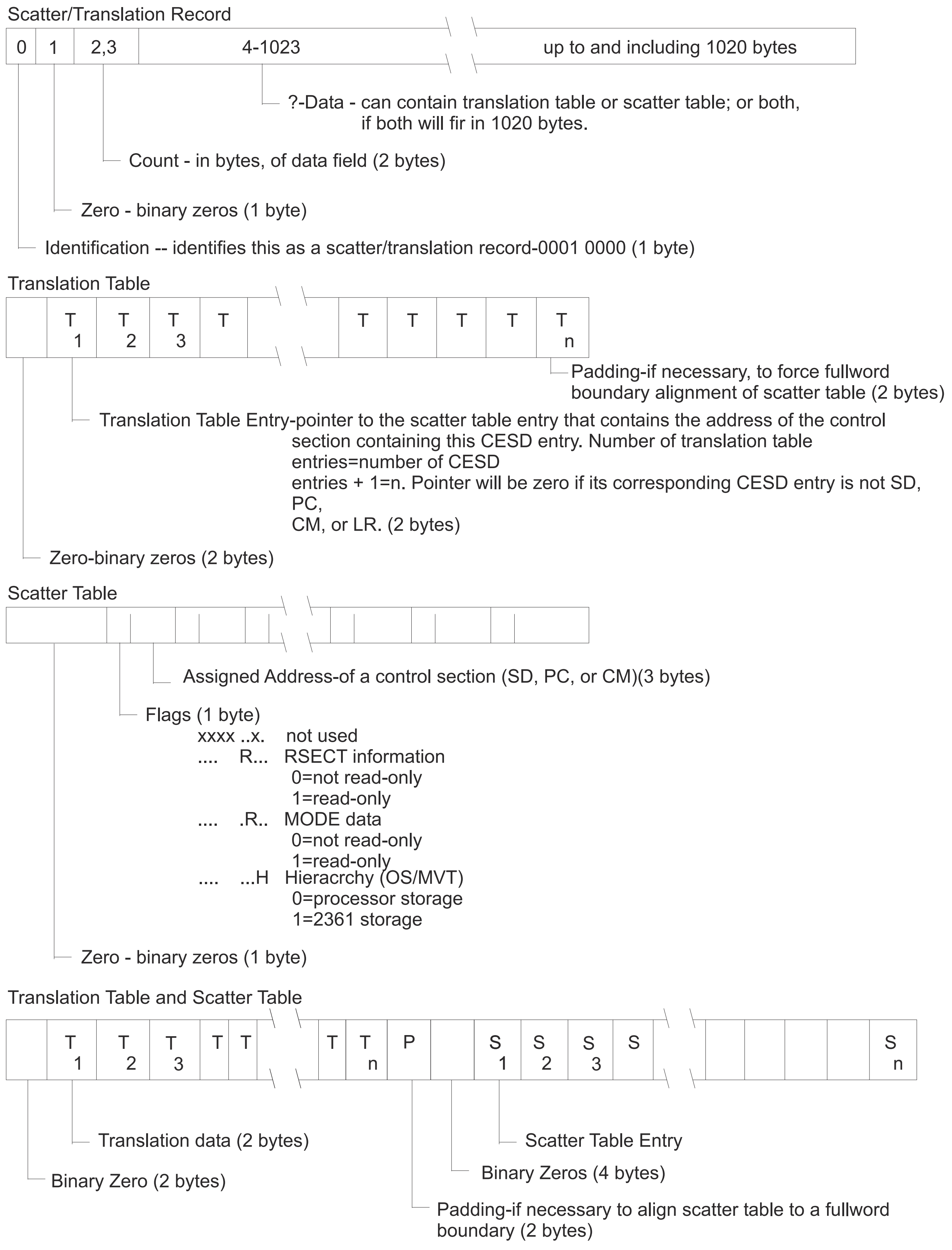

Scatter/Translation Record
0
1
2,3
4-1023
up to and including 1020 bytes
?-Data - can contain translation table or scatter table; or both, if both will fir in 1020 bytes.
Count - in bytes, of data field (2 bytes)
Zero - binary zeros (1 byte)
Identification -- identifies this as a scatter/translation record-0001 0000 (1 byte)
Translation Table
T 1
T 2
T 3
T
T
T
T
T
T n
Padding-if necessary, to force fullword boundary alignment of scatter table (2 bytes)
Translation Table Entry-pointer to the scatter table entry that contains the address of the control section containing this CESD entry. Number of translation table entries=number of CESD entries + 1=n. Pointer will be zero if its corresponding CESD entry is not SD, PC, CM, or LR. (2 bytes)
Zero-binary zeros (2 bytes)
Scatter Table
Assigned Address-of a control section (SD, PC, or CM)(3 bytes)
Flags (1 byte)
xxxx ..x. not used
.... R... RSECT information
0=not read-only
1=read-only
.... .R.. MODE data
0=not read-only
1=read-only
.... ...H Hieracrchy (OS/MVT)
0=processor storage
1=2361 storage
Zero - binary zeros (1 byte)
Translation Table and Scatter Table
T 1
T 2
T 3
T
T
T
T n
P
S 1
S 2
S 3
S
S n
Translation data (2 bytes)
Binary Zero (2 bytes)
Scatter Table Entry
Binary Zeros (4 bytes)
Padding-if necessary to align scatter table to a fullword boundary (2 bytes)

*Figure 3. Scatter/Translation record*

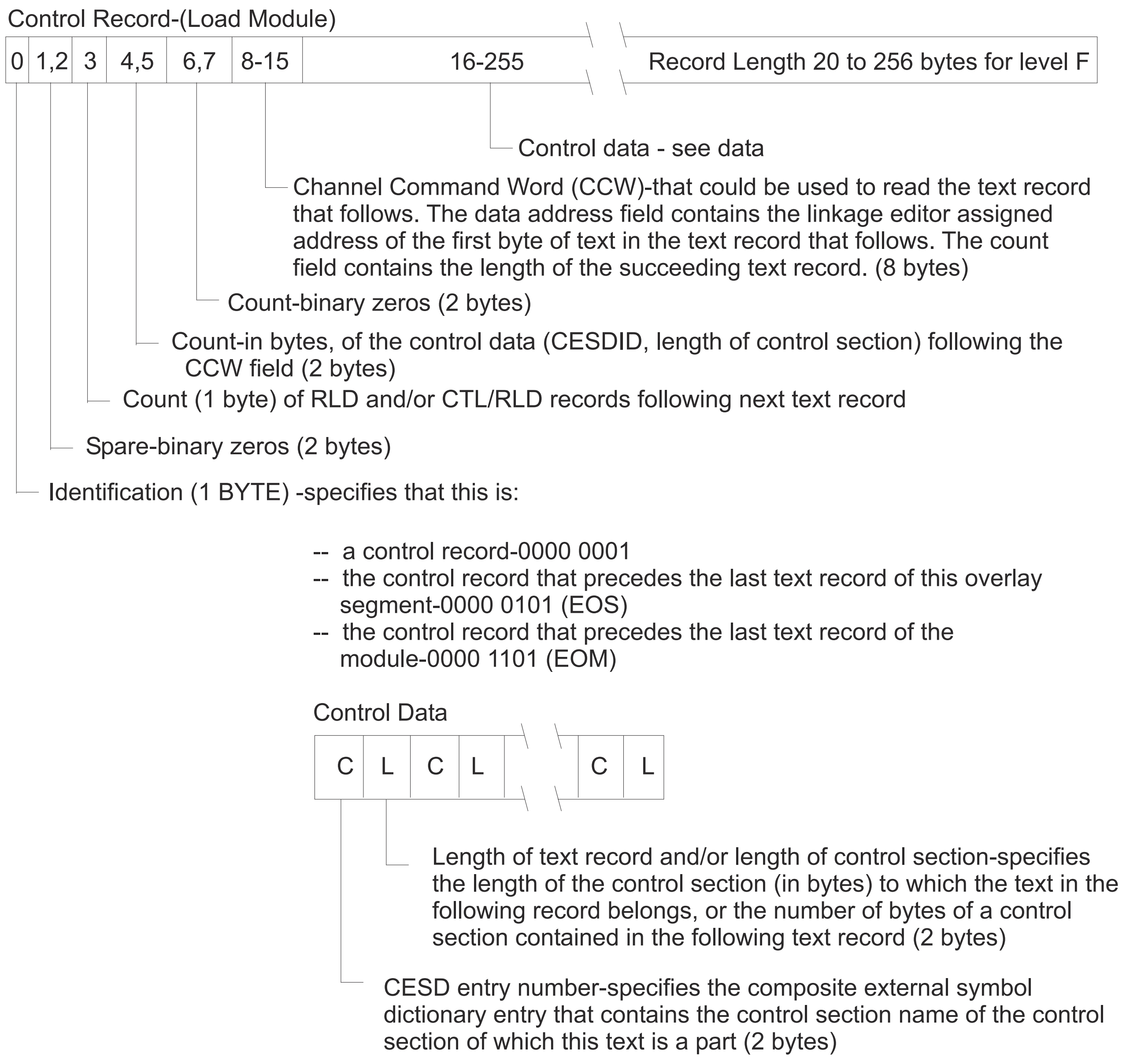
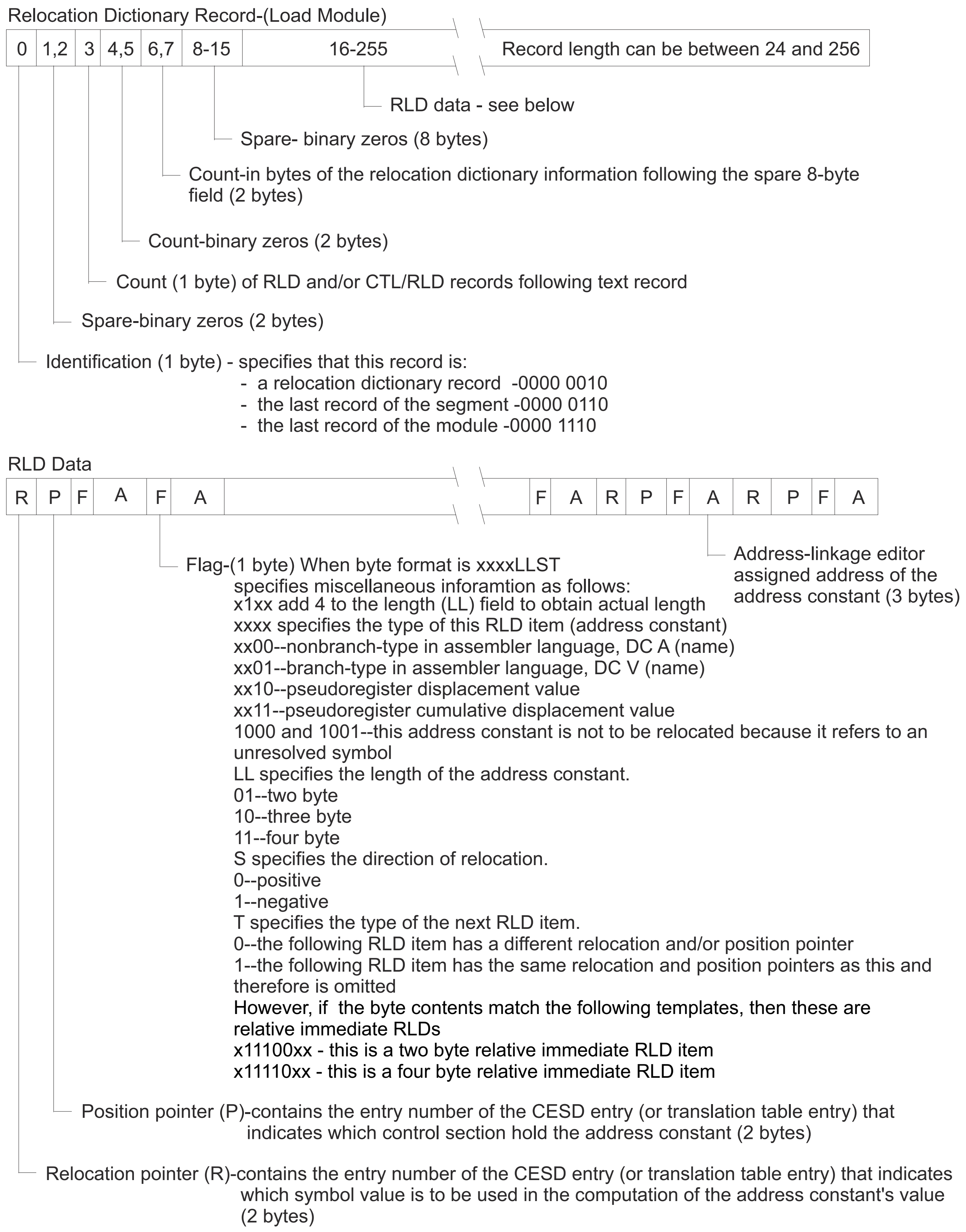

Control Record-(Load Module)
0
1,2
3
4,5
6,7
8-15
16-255
Record Length 20 to 256 bytes for level F
Control data - see data
Channel Command Word (CCW)-that could be used to read the text record that follows. The data address field contains the linkage editor assigned address of the first byte of text in the text record that follows. The count field contains the length of the succeeding text record. (8 bytes)
Count-binary zeros (2 bytes)
Count-in bytes, of the control data (CESDID, length of control section) following the CCW field (2 bytes)
Count (1 byte) of RLD and/or CTL/RLD records following next text record
Spare-binary zeros (2 bytes)
Identification (1 BYTE) -specifies that this is:
-- a control record-0000 0001
-- the control record that precedes the last text record of this overlay segment-0000 0101 (EOS)
-- the control record that precedes the last text record of the module-0000 1101 (EOM)
Control Data
C
L
C
L
C
L
Length of text record and/or length of control section-specifies the length of the control section (in bytes) to which the text in the following record belongs, or the number of bytes of a control section contained in the following text record (2 bytes)
CESD entry number-specifies the composite external symbol dictionary entry that contains the control section name of the control section of which this text is a part (2 bytes)

*Figure 4. Control record (load module)*



Relocation Dictionary Record-(Load Module)

| 0 | 1,2 | 3 | 4,5 | 6,7 | 8-15 | 16-255 | Record length can be between 24 and 256 |
|---|---|---|---|---|---|---|---|

- 16-255: RLD data - see below
- 8-15: Spare- binary zeros (8 bytes)
- 6,7: Count-in bytes of the relocation dictionary information following the spare 8-byte field (2 bytes)
- 4,5: Count-binary zeros (2 bytes)
- 3: Count (1 byte) of RLD and/or CTL/RLD records following text record
- 1,2: Spare-binary zeros (2 bytes)
- 0: Identification (1 byte) - specifies that this record is:
  - a relocation dictionary record -0000 0010
  - the last record of the segment -0000 0110
  - the last record of the module -0000 1110

RLD Data

| R | P | F | A | F | A | | F | A | R | P | F | A | R | P | F | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

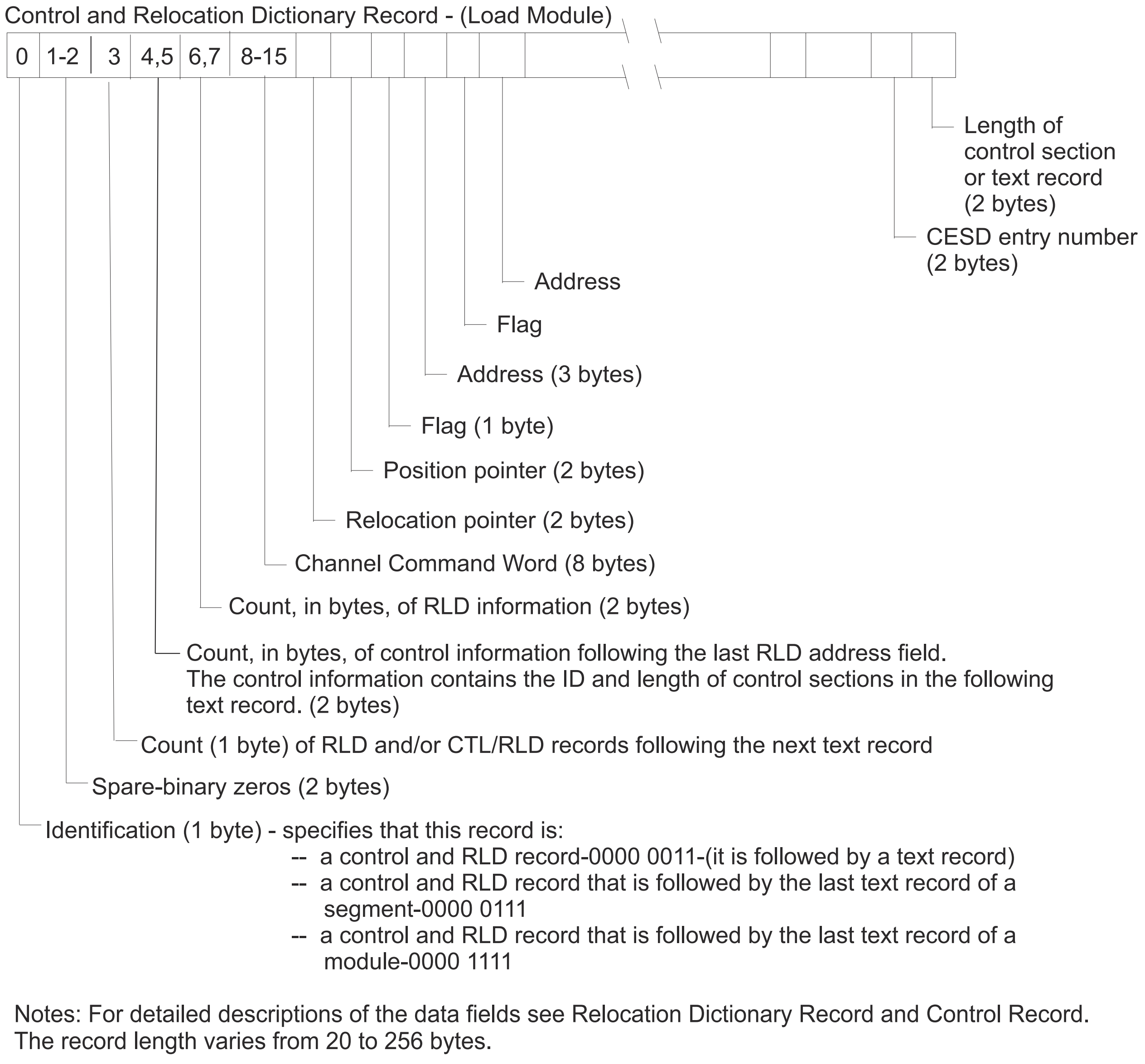
A: Address-linkage editor assigned address of the address constant (3 bytes)

F: Flag-(1 byte) When byte format is xxxxLLST specifies miscellaneous inforamtion as follows:
x1xx add 4 to the length (LL) field to obtain actual length
xxxx specifies the type of this RLD item (address constant)
xx00--nonbranch-type in assembler language, DC A (name)
xx01--branch-type in assembler language, DC V (name)
xx10--pseudoregister displacement value
xx11--pseudoregister cumulative displacement value
1000 and 1001--this address constant is not to be relocated because it refers to an unresolved symbol
LL specifies the length of the address constant.
01--two byte
10--three byte
11--four byte
S specifies the direction of relocation.
0--positive
1--negative
T specifies the type of the next RLD item.
0--the following RLD item has a different relocation and/or position pointer
1--the following RLD item has the same relocation and position pointers as this and therefore is omitted
However, if the byte contents match the following templates, then these are relative immediate RLDs
x11100xx - this is a two byte relative immediate RLD item
x11110xx - this is a four byte relative immediate RLD item

P: Position pointer (P)-contains the entry number of the CESD entry (or translation table entry) that indicates which control section hold the address constant (2 bytes)

R: Relocation pointer (R)-contains the entry number of the CESD entry (or translation table entry) that indicates which symbol value is to be used in the computation of the address constant's value (2 bytes)

*Figure 5. Relocation dictionary record (load module)*



Control and Relocation Dictionary Record - (Load Module)
0
1-2
3
4,5
6,7
8-15
Length of control section or text record (2 bytes)
CESD entry number (2 bytes)
Address
Flag
Address (3 bytes)
Flag (1 byte)
Position pointer (2 bytes)
Relocation pointer (2 bytes)
Channel Command Word (8 bytes)
Count, in bytes, of RLD information (2 bytes)
Count, in bytes, of control information following the last RLD address field. The control information contains the ID and length of control sections in the following text record. (2 bytes)
Count (1 byte) of RLD and/or CTL/RLD records following the next text record
Spare-binary zeros (2 bytes)
Identification (1 byte) - specifies that this record is:
-- a control and RLD record-0000 0011-(it is followed by a text record)
-- a control and RLD record that is followed by the last text record of a segment-0000 0111
-- a control and RLD record that is followed by the last text record of a module-0000 1111
Notes: For detailed descriptions of the data fields see Relocation Dictionary Record and Control Record. The record length varies from 20 to 256 bytes.

*Figure 6. Control and relocation dictionary record (load module)*

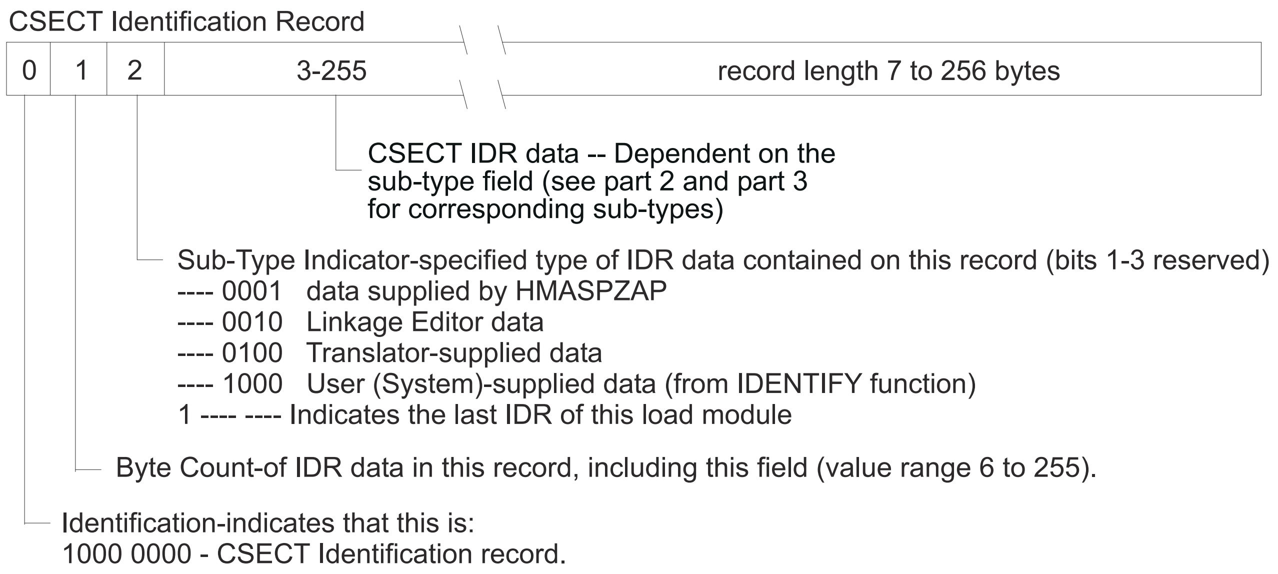
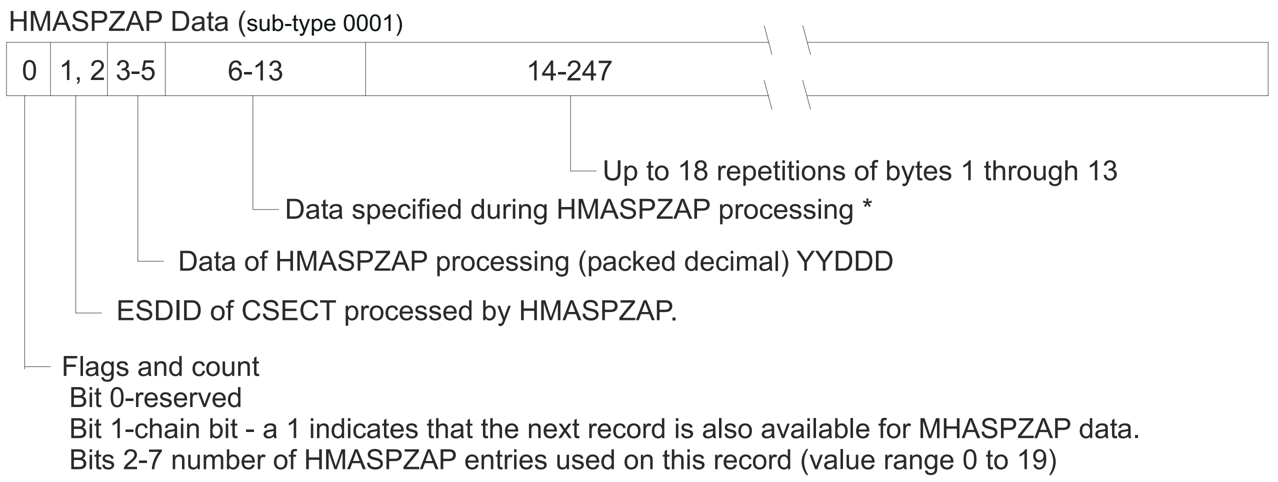
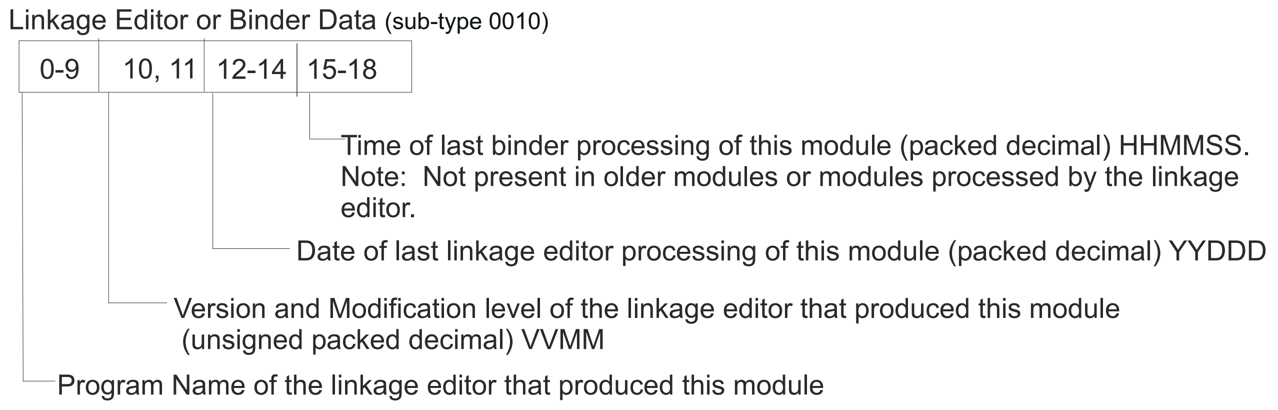

CSECT Identification Record
0
1
2
3-255
record length 7 to 256 bytes
CSECT IDR data -- Dependent on the sub-type field (see part 2 and part 3 for corresponding sub-types)
Sub-Type Indicator-specified type of IDR data contained on this record (bits 1-3 reserved)
---- 0001 data supplied by HMASPZAP
---- 0010 Linkage Editor data
---- 0100 Translator-supplied data
---- 1000 User (System)-supplied data (from IDENTIFY function)
1 ---- ---- Indicates the last IDR of this load module
Byte Count-of IDR data in this record, including this field (value range 6 to 255).
Identification-indicates that this is:
1000 0000 - CSECT Identification record.

*Figure 7. Record format of load module IDRs–part 1*


HMASPZAP Data (sub-type 0001)
0
1, 2
3-5
6-13
14-247
Up to 18 repetitions of bytes 1 through 13
Data specified during HMASPZAP processing *
Data of HMASPZAP processing (packed decimal) YYDDD
ESDID of CSECT processed by HMASPZAP.
Flags and count
Bit 0-reserved
Bit 1-chain bit - a 1 indicates that the next record is also available for MHASPZAP data.
Bits 2-7 number of HMASPZAP entries used on this record (value range 0 to 19)

* May be a PTF number or up to eight bytes of variable user data specified on an HMASPZAP IDRDATA control statement.


Linkage Editor or Binder Data (sub-type 0010)
0-9
10, 11
12-14
15-18
Time of last binder processing of this module (packed decimal) HHMMSS.
Note: Not present in older modules or modules processed by the linkage editor.
Date of last linkage editor processing of this module (packed decimal) YYDDD
Version and Modification level of the linkage editor that produced this module (unsigned packed decimal) VVMM
Program Name of the linkage editor that produced this module

**Note:** Date and time fields contents of "65001F0000000F" are generated by the binder when copying a module lacking a binder IDR record.

*Figure 8. Record format of load module IDRs–part 2*

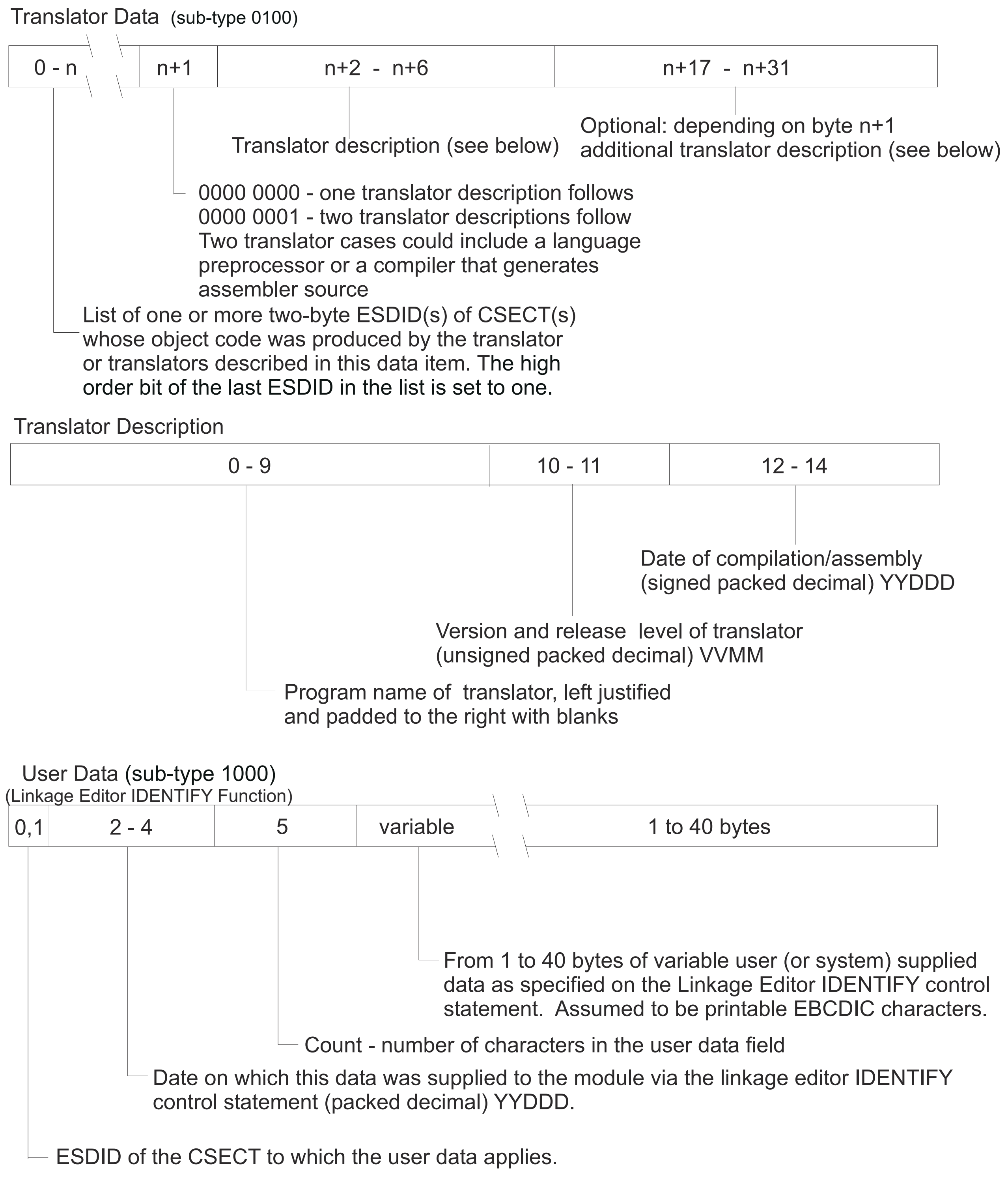

Translator Data (sub-type 0100)
0 - n
n+1
n+2 - n+6
n+17 - n+31
Translator description (see below)
Optional: depending on byte n+1 additional translator description (see below)
0000 0000 - one translator description follows
0000 0001 - two translator descriptions follow
Two translator cases could include a language preprocessor or a compiler that generates assembler source
List of one or more two-byte ESDID(s) of CSECT(s) whose object code was produced by the translator or translators described in this data item. The high order bit of the last ESDID in the list is set to one.
Translator Description
0 - 9
10 - 11
12 - 14
Date of compilation/assembly (signed packed decimal) YYDDD
Version and release level of translator (unsigned packed decimal) VVMM
Program name of translator, left justified and padded to the right with blanks
User Data (sub-type 1000)
(Linkage Editor IDENTIFY Function)
0,1
2 - 4
5
variable
1 to 40 bytes
From 1 to 40 bytes of variable user (or system) supplied data as specified on the Linkage Editor IDENTIFY control statement. Assumed to be printable EBCDIC characters.
Count - number of characters in the user data field
Date on which this data was supplied to the module via the linkage editor IDENTIFY control statement (packed decimal) YYDDD.
ESDID of the CSECT to which the user data applies.

*Figure 9. Record format of load module IDRs–part 3*

## Signing records after the nominal last record

For a signed load module, there are multiple extended records after the nominal last record. They are called signing records.

There are two kinds of signing records:

- Directory entry record, which saves a directory entry for a signed primary member and its aliases.
- Signature record, which saves signatures.

Signing records have a common 8-byte extended header described in the following table.

*Table 47. Mapping of record header REC_HEADER*

| Field name | Length | Description |
|---|---|---|
| REC_PREFIX | 1 | This field must be 0x88. |
| REC_SUBTYPE | 1 | This field can be one of the following: 0x00: directory entry record, or 0x01: signature record. |
| REC_VERSION | 1 | This field must be 0x01. |
| REC_FLAGS | 1 | Bits 6-7 can be one of the following: **00** This record is the initial record of the designated type (it is not a continuation record). It is not continued on the succeeding record. **01** This record is the initial record of the designated type (it is not a continuation record). It is continued on the succeeding record. **10** This record is a continuation of the previous record of this type, and it is not continued on the following record. **11** This record is a continuation of the previous record of this type, and it is continued on the following record. Other bits are reserved. |
| REC_LENGTH | 2 | Length of the current record, including this header. The record length is 1024 at most. |
| * | 2 | RESERVED |

For the mapping of the directory entry record, refer to "Mapping of the directory entry record" on page 138. For the mapping of the signing record, refer to "Mapping of the signature record" on page 139

## Mapping of the directory entry record

This record contains directory entries for a load module, including one for a primary member and others for its aliases.

*Table 48. Mapping of the directory entry record*

| Field name | Length | Description |
|---|---|---|
| Header | 8 | For the header description, refer to Table 47 on page 138. |
| ENTRY_NUM | 2 | The number of directory entries in this record. |
| ENTRY_DATA | variable | A listing of all directory entries in this record. The length of each entry is determined by its self-defined length in the entry. One entry is followed by the next without gaps. |

## Mapping of the signature record

The following table contains the mapping for the signature record.

*Table 49. Mapping of the signature record*

| Field name | Length | Description |
|---|---|---|
| Header | 8 | For the header description, refer to Table 47 on page 138. |
| Timestamp | 16 | 16-byte STCKE data converted to GMT |
| SIGN_TYPE | 1 | **0** Signature for the module fetch<br>**1** Signature for the scatter load<br>**2** Signature for the binder |
| SIGN_VERSION | 1 | **1** The hash algorithm is SHA2-512 and the signing algorithm is Elliptic Curve ECDSA P521. |
| SIGN_LEN | 2 | Length of signature data saved in SIGN_DATA. |
| * | 32 | Reserved |
| SIGN_DATA | SIGN_LEN | Signature data returned by the RACF signing service. |

# Appendix A. Accessibility

Accessible publications for this product are offered through IBM Documentation (www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact the z/OS team web page (www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

# Notices

## Terms and conditions for product documentation

### Applicability

### Personal use

### Commercial use

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

## Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

## Trademarks

# Index

Product Number: 5650-ZOS