z/OS
2.4

MVS Interactive Problem Control System (IPCS) Customization

SA23-1383-40

Note

Before using this information and the product it supports, read the information in “Notices” on page 181.

This edition applies to Version 2 Release 4 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2021-06-21

Contents

Figures

Tables

About this information

This information explains how to customize the interactive problem control system (IPCS) for some or all IPCS sessions using the following topics:

- Customizing the IPCS installation package
- Customizing IPCS session parameters
- Customizing dump directory
- Customizing access to IPCS
- Customizing the IPCS dialog

This information also explains:

- Accessing and using IPCS on a target system
- Providing security for IPCS
- Writing and installing IPCS exit routines and various IPCS exit services

Customization can also apply to ABEND/SNAP formatting. IPCS operates in interactive and batch environments supported by TSO/E.

Who should use this information

This information is for system programmers who:

- Use IPCS to format dumps
- Write IPCS exit routines
- Customize IPCS for an installation

z/OS information

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

When possible, this information uses cross document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

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 xv.

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 following deliverable title and order number: z/OS MVS IPCS Customization, SA23-1383-50
- The section title of the specific information to which your comment relates
- 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.

Summary of changes

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

Summary of changes for z/OS Version 2 Release 4 (V2R4)

Next Refresh

- Updated clock value. See "Registers" on page 158.

January 2021

- Customizing Data Privacy for Diagnostics added. See "Customizing Data Privacy for Diagnostics" on page 13.

Summary of changes for z/OS MVS IPCS Customization for Version 2 Release 3 (V2R3)

Changed

- "Editing the BLSCDDIR CLIST" on page 21 was updated to clarify the placement of directories.

Summary of changes for z/OS MVS IPCS Customization for Version 2 Release 2 (V2R2) as updated September 2016

General content changes for z/OS MVS IPCS Customization

The following content is new, changed, or no longer included in V2R2 as updated September 2016.

Changed

The following content is changed.

- References were added to point to more information about IPCS panel customization, specifically for clarification on which functions to use and when. For more information, see "IPCS REXX Execs and CLISTs" on page 3, "IPCS Dialog Programs" on page 4, and "BLSCLIBD CLIST - Activate IPCS Dialog Services" on page 27.
- The design example for the CTRACE buffer find exit routine was updated. For more information, see "Design Example" on page 68.
- The information about editing the BLSCDDIR CLIST was updated. For more information, see "Editing the BLSCDDIR CLIST" on page 21.

z/OS Version 2 Release 1 summary of changes

See the Version 2 Release 1 (V2R1) versions of the following publications for all enhancements related to z/OS V2R1:

- *z/OS Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*

- *z/OS Introduction and Release Guide*

Chapter 1. IPCS Installation Package

This chapter describes the IPCS installation package and introduces customization of the package. Table 1 on page 1 explains the contents of the package.

Table 1. Contents of IPCS installation package

Item in IPCS Installation Package	Purpose
SYS1.PARMLIB members related to IPCS	• Define data areas, symbols, component analysis dialogs, and IBM- and installation-supplied exit routines for IPCS dump formatting and analysis. • Imbed IBM- and installation-supplied parmlib members. • Specify parameters to be used during an IPCS session.
IPCS subcommands	• Analyze, view, and format dump data. • Combine trace data from dump or trace data sets, or both, into a single trace data set. • Debug an IPCS exit routine. • Maintain the IPCS dump directory, symbol table, and storage map. • Merge and display formatted output from multiple application traces and generalized trace facility (GTF) traces. • Perform utility functions, including customization of IPCS sessions.
IPCS restructured extended executor language (REXX) EXECs and command lists (CLISTs)	• Begin an IPCS dialog. • Create a dump directory. • Generate problem screening reports. • Perform initial dump analysis. • Print system storage areas. • Process one or more IPCS subcommands. • Provide examples of IPCS subcommands.
IPCS dialog programs	• Generate the functions associated with the IPCS dialog. • Customize the IPCS dialog.
SYS1.MIGLIB system library	• Allow IPCS and the component formatting and analysis programs in one version of MVS™ to function in another version of MVS. • Install exit routines.
IPCS macros and mapping macros	Allow programmers to map data areas and perform functions needed when writing IPCS exit routines that format and analyze dump data.
IPCS exit services	Provide services to IPCS exit routines necessary for performing tasks related to accessing, formatting, analyzing, printing, or displaying information contained in a dump data set.

SYS1.PARMLIB Members Related to IPCS

IPCS provides parmlib members to control IPCS processing. The BLSCECT, BLSCECTX, and BLSCUSER parmlib members define data areas, symbols, component analysis dialogs, and IBM- and installation-supplied exit routines for IPCS dump formatting and analysis. These parmlib members may also embed IBM- or installation-supplied parmlib members. Note that the data areas, exit routines, and embedded parmlib members can also be used for ABEND and SNAP dump formatting.

For SNAP formatting, IPCS procedure BLSJPRMI initializes IPCS formatting tables. During system initialization, a START command from parmlib member IEACMD00 initiates BLSJPRMI. If PARMLIB members are updated after BLSJPRMI runs, new formatting tables are built, using the current contents of PARMLIB, and the formatters are used by both IPCS and formatted dumps (such as SYSUDUMP and SYSABEND). If BLSJPRMI fails to initialize the IPCS tables for SNAP dump processing, IPCS issues message BLS001E. BLSJPRMI issues the following return codes:

Return Code
Description

0
Normal completion.

4
Attention; condition(s) occurred, but did not prevent successful completion of processing.

8
Error condition(s) occurred, but did not prevent successful completion of processing.

12
Serious conditions occurred; problems pertaining to the syntax or consistent semantic of the parmlib statements) prevented completion of some portion of processing.

16
Terminating conditions occurred; problems pertaining to the processing environment prevented completion of some portion of processing.

The IPCSPRxx members define IPCS session parameters. When starting an IPCS session, most IPCS users enter the **IPCS NOPARM** command to indicate that IPCS is not to use session parameters specified in the IPCSPR00 or any other IPCSPRxx parmlib member.

The parmlib members to control IPCS processing are:

- **BLSCECT**

  BLSCECT is an IBM-supplied default member of SYS1.PARMLIB. This default member embeds BLSCECTX, other IBM-supplied parmlib members, and BLSCUSER.

- **BLSCECTX**

  BLSCECTX is an IBM-supplied default member of SYS1.PARMLIB. This member specifies the format exit routines for IBM programs not in the base system. This member also embeds other IBM-supplied parmlib members used by MVS components to define their IPCS support.

- **BLSCUSER**

  BLSCUSER is an optional parmlib member an installation can create and use for customization. IBM does not supply a BLSCUSER member.

- **IPCSPR00**

  The IBM-supplied default is the IPCSPR00 member in SYS1.PARMLIB. This member contains parameters for an IPCS session.

- **IPCSPRxx**

  Installations can supply one or more IPCSPRxx parmlib members to identify customized parameters to be used during IPCS sessions. These parameters let an installation tailor IPCS sessions to its requirements.

For more information about the topics in the section, see the following references:

- Chapter 2, “Customizing IPCS Session Parameters,” on page 17 for using IPCS parmlib members and alternate parmlibs to customize IPCS session parameters
- “CTRACE Filter/Analysis (CTRF) Exit Routine” on page 69 for using the PANDEF statement in a BLSCUSER parmlib member to create a component analysis dialog
- Chapter 9, “Installing IPCS Exit Routines,” on page 93 for using IPCS parmlib members to install IPCS exit routines

- *z/OS MVS Initialization and Tuning Reference* for the BLSCECT, BLSCUSER, and IPCSPRxx members

IPCS Subcommands

IPCS supplies subcommands to do the following:

- Analyze, view, and format dump data
- Combine trace data from dump or trace data sets, or both, into a single trace data set
- Debug an IPCS exit routine by setting traps through the IPCS debug tool
- Maintain the IPCS dump directory, symbol table, and storage map
- Merge and display formatted output from multiple application traces and GTF traces
- Perform utility functions, including customization of IPCS sessions

For customization, use the following subcommands:

Subcommand
Purpose

COPYDDIR
To allow IPCS users to copy data from one dump directory for a particular dump related to a particular data set into another dump directory.

COPYTRC
To combine trace data from dump or trace data sets, or both, into a single trace data set.

GO
To allow IPCS users to resume IPCS trap processing performed by the IPCS debug tool. The traps can be used to obtain diagnostic input and output information about exit routines that use one of the IPCS exit services.

MERGE
To display formatted output that is merged from multiple component traces and from generalized trace facility (GTF) traces.

NOTE
To allow IPCS users to send messages to the IPCSPRNT data set.

PROFILE
To customize the line widths and the number of lines per printed page of IPCS reports.

SETDEF
To customize IPCS subcommand parameters.

TRAPLIST
To display the status of IPCS-supplied traps.

TRAPOFF
To deactivate an IPCS-supplied trap.

TRAPON
To activate an IPCS-supplied trap.

See the following references for more information:

- Chapter 11, “The IPCS Debug Tool,” on page 161 for information about using the debug tool through the GO, TRAPLIST, TRAPOFF, and TRAPON subcommands
- *z/OS MVS IPCS Commands* for information about the COPYDDIR, COPYTRC, GO, MERGE, NOTE, PROFILE, SETDEF, TRAPLIST, TRAPOFF, and TRAPON subcommands

IPCS REXX Execs and CLISTs

The system library, SYS1.SBLSCLI0, contains copies of the REXX EXECs and CLISTs supplied by IBM. All IPCS REXX EXECs begin with the prefix BLSX. All IPCS CLISTs begin with the prefix BLSC. Use REXX EXECs and CLISTs to:

- Begin an IPCS dialog
- Create a dump directory
- Generate problem screening reports
- Perform initial dump analysis
- Print system storage areas
- Process one or more IPCS subcommands
- Provide examples of IPCS subcommands

For customization:

- Individual IPCS users can invoke the BLSCDDIR CLIST supplying parameters to change dump directory defaults, such as the dump directory volume serial or the number of records provided for dump directories.
- Installations can edit the BLSCDDIR CLIST to customize the dump directory for the entire installation.
- Individual users or installations can run the BLSCLIBD CLIST to begin an IPCS dialog. IBM recommends using this CLIST rather than using the BLSGLIBD dialog program. For more information about starting the IPCS dialog, see "Invoking the IPCS Dialog: Deciding Which Function to Use" on page 26.

See the following references for more information:

- *z/OS MVS IPCS User's Guide* for information about writing REXX EXECs and CLISTs for IPCS and invoking them from an IPCS session
- Chapter 3, "Customizing the Dump Directory," on page 21 for editing the BLSCDDIR CLIST to customize a dump directory for an entire installation
- Chapter 5, "Customizing the IPCS Dialog," on page 37 for using the BLSCLIBD CLIST to start an IPCS dialog
- *z/OS MVS IPCS User's Guide* for using the BLSCDDIR CLIST to customize a dump directory for an individual IPCS user

IPCS Dialog Programs

IPCS supplies dialog programs and a CLIST to generate and customize the IPCS dialog.IPCS supplies the following dialog programs:

Program
Purpose

BLSG
Begins an IPCS dialog. For more information about starting the IPCS dialog, see "Invoking the IPCS Dialog: Deciding Which Function to Use" on page 26.

BLSGLIBD
An alternative way to begin an IPCS dialog. In addition to the function provided by BLSG, BLSGLIBD allocates, through the ISPF LIBDEF service, the following ddnames and associated data sets:

Ddname	Associated Data Set
ISPPLIB	SYS1.SBLSPNL0
ISPMLIB	SYS1.SBLSMSG0
ISPSLIB	SYS1.SBLSKEL0
ISPTLIB	SYS1.SBLSTBL0

Note: IPCS supplies the BLSCLIBD CLIST to start an IPCS dialog. BLSCLIBD performs the same function as the BLSGLIBD dialog program. IBM recommends using the BLSCLIBD CLIST rather than

using the BLSGLIBD dialog program to begin an IPCS dialog. For more information about starting the IPCS dialog, see "Invoking the IPCS Dialog: Deciding Which Function to Use" on page 26.

BLSGSCMD
Causes IPCS to process an IPCS subcommand or CLIST.

BLSLDISP
Browses an IPCS dump data set.

BLSGDCDA
Displays the list of component data analysis routines found in either:

- The BLSCECT parmlib member
- In parmlib members embedded in the BLSCECT parmlib member

BLSGDUIN
Displays a list of dumps found in the dump directory.

Dialog programs BLSG, BLSGLIBD, and BLSGSCMD may be used in a TSO/E batch environment. All others require an interactive ISPF environment where both full screen and line mode output are presented in real time.

To customize the IPCS dialog, use:

- The ISPF LIBDEF service
- ISPEXEC to use the Interactive System Productivity Facility (ISPF) SELECT service with the IPCS dialog programs

See the following references for more information:

- Chapter 5, "Customizing the IPCS Dialog," on page 37 for information about customizing the IPCS dialog by using:
  - The ISPF LIBDEF service
  - The ISPF ISPEXEC and SELECT services with the IPCS dialog programs
- *z/OS ISPF Dialog Tag Language Guide and Reference* for information about the ISPF ISPEXEC, SELECT, and LIBDEF services and dialog programs

SYS1.MIGLIB System Library

IPCS supplies the SYS1.MIGLIB system library to:

- Allow IPCS and the component formatting and analysis programs in one version of MVS to function in another version of MVS.
- Allow you to install installation-provided IPCS exit routines in your system.

See Chapter 6, "Using IPCS on Another System," on page 45 for information about using SYS1.MIGLIB to process multiple levels of dumps. See Chapter 9, "Installing IPCS Exit Routines," on page 93 for information about using SYS1.MIGLIB to install IPCS exit routines.

IPCS Macros and Mapping Macros

The system supplies macros and mapping macros to allow you to:

- Map IPCS data areas
- Create IPCS exit routines
- Invoke and use IPCS exit services within those exit routines

Create IPCS exit routines, using exit services, to:

- Add symptom strings
- Analyze dump data

- Create diagnostic reports of unlimited possibilities
- Format dump data
- Generate titles and table of contents entries
- Locate data in a dump
- Validate data in a dump

Programming Interface Information

IPCS Macro for Invoking an IPCS Exit Service within an IPCS Exit Routine

Mapping macro BLSABDPL maps field ADPLSERV, which contains the address of the exit services router. Use the exit services router to invoke many of the exit services. Table 4 on page 10 indicates which exit services are invoked through the exit services router.

For more information, see the following references:

- See "Invoking with the Exit Services Router" on page 98 for information about the exit services router
- See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary), for a mapping of BLSABDPL.
- See *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for coding the BLSABDPL macro.

End Programming Interface Information

Programming Interface Information

IPCS Macros for Using an IPCS Exit Service within an IPCS Exit Routine

IPCS mapping macros are supplied to map exit service parameter lists needed to use IPCS exit services within an IPCS exit routine. Table 2 on page 6 lists the IPCS exit services, and gives the mapping macro that maps the exit service parameter list for each exit service. Find *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Table 2. IPCS Mapping Macros for IPCS Exit Services

IPCS Exit Service	Mapping Macro as Listed In Data Areas	Maps Exit Service Parameter List
Add symptom	BLSADSY	BLSADSY*
Control block formatter	BLSABDPL	ADPLPFMT
Control block status (CBSTAT)	BLSACBSP	BLSACBSP*
Create contention queue element (CQE)	BLSAPCQE	BLSAPCQE*
Equate symbol	BLSRESSY	BLSRESSY*
Exit control table (ECT)	BLSABDPL	ADPLPECT
Expanded print	BLSUPPR2	BLSUPPR2*
Format model processor	BLSABDPL	ADPLPFMT
Get symbol	BLSRESSY	BLSRESSY*
Name	BLSRNAMP	BLSRNAMP*
Name/token lookup	BLSQNTKP	BLSQNTKP*
Select address space identifier (ASID)	BLSABDPL	ADPLPSEL
Standard print	BLSUPPR2	BLSUPPR2*
Storage access	BLSABDPL	ADPLPACC
Storage map	BLSRXMSP	BLSRXMSP*

Table 2. IPCS Mapping Macros for IPCS Exit Services (continued)		
IPCS Exit Service	**Mapping Macro as Listed In Data Areas**	**Maps Exit Service Parameter List**
Symbol service	BLSRXSSP	BLSRXSSP*
WHERE	BLSRPWHS	BLSRPWHS*
Note: * The mapping macro that supports this parameter list permits the user to specify the name of the parameter list. The name shown is the one that appears in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).		

See the following references for more information:

- Chapter 10, "IPCS Exit Services," on page 95 for information about using the mapping macros when using the IPCS exit services
- *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for coding the macros to generate macro mappings
- *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for macro mappings

End Programming Interface Information

Programming Interface Information

IPCS Macros for Writing IPCS Exit Routines

You might need the data areas mapped by the following mapping macros to write IPCS exit routines:

- **BLSQFXL**

  Maps the format exit routine list (FXL) used by model processor formatting (MPF) exit routines. FXL contains the addresses of data of potential interest to the MPF exit routine, as well as a description of the formatted line.

- **BLSRDATC**

  Maps the data area used by IPCS to describe relocatable attributes of a block of storage. This includes the length, data type, offset, plus the dimension and initial subscript for blocks of storage that are described as one-dimensional arrays. The BLSRDATC data area appears within the BLSRESSY data area as field ESSYD; it also appears within the BLSRSASY data area as field SASYF.

- **BLSRDATS**

  Maps the data area used by IPCS to describe an address space in a dump source. BLSRDATS maps the data for the IPCS address space selection parameters.

  The data in BLSRDATS can be for a symbolic literal that describes an address space. When BLSRDATS contains a symbolic literal, the T field contains C'LI' for the literal symbol ZZZASTLI, the 1 field contains a numeric index assigned to the literal when it was created, and the other fields contain reserved values.

  If the numeric index is zero, the address space contains no storage; IPCS services return LITERAL(0) when a caller refers to an address space with no storage.

  The BLSRDATS data area appears within the BLSRESSY data area as field ESSYAS; it also appears within the BLSRSASY data area as field SASYAS. In addition, it appears within a number of parameter lists, such as the WHERE exit service parameter list and the CQE create exit service parameter list.

- **BLSRDATT**

  Maps the data area used by IPCS to describe the data type of a block of storage. IPCS defines equates within the BLSRDATT macro that match the data types:

  Equate
  Type of Data

ZZZDTYA
Pointer

ZZZDTYB
Bit string

ZZZDTYC
Character string

ZZZDTYF
Signed number

ZZZDTYL
Module name

ZZZDTYM
Structure or control block

ZZZDTYU
Area (not a module or control block)

ZZZDTYY
Unsigned number

Use one of these equates to set field ESSYDTY. For example, when retrieving a control block, use the equate ZZZDTYM or when retrieving the symbol for the PRIVATE area, use the equate ZZZDTYU.

The BLSRDATT data area appears within the BLSRDATC data area, and, therefore, within both the BLSRESSY and BLSRSASY data areas. In the BLSRESSY data area, it appears as field ESSYDT; in the BLSRSASY data area, it appears as field SASYFT. It also appears directly within the BLSRSASY data area as field SASYDT. In a storage map that is properly filled in, fields SASYDT and SASYFT usually contain the same data. The two fields may differ when the block in question is part of a group. For example, SASYDT may record that the block is an instance of data type STRUCTURE (UCB) while SASYFT refines that description to record that it, more accurately, is an instance of data type STRUCTURE (UCBTAPE).

- **BLSRDRPX**

  Maps the dump record prefix, which contains the title of the dump and other information needed for interpretation of the dump. BLSRPRD invokes this to map part of the total dump record, but BLSRDRPX can be invoked independently.

- **BLSRPRD**

  Maps the format of dump records that may be accessed by the summary dump access service.

- **BLSRSASY**

  Maps the structure of a dump directory storage address (SA) record. This is a key structure in the interface to scan exits and to the storage map service.

See the following references for more information:

- “Model Processor Formatting (MPF) Exit Routine” on page 81
- “Scan Exit Routine” on page 85
- “Storage Map Service” on page 137
- “Summary Dump Data Access Service” on page 174
- *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for coding the macros to generate the macro mappings
- *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for macro mappings

End Programming Interface Information

Programming Interface Information

IPCS Macros for Creating a Control Block Model

The following macros are supplied to allow you to create a control block model:

- **BLSQMDEF**

  Starts and ends the definition of a formatting model.

- **BLSQMFLD**

  Identifies fields that are to be formatted.

- **BLSQSHDR**

  Defines a text string, called a subheader, to appear as part of the output of the model processor.

See the following references for more information:

- "Format Model Processor Service" on page 120 and "Format Models" on page 122 for information about using the BLSQMDEF, BLSQMFLD, and BLSQSHDR macros to create a control block model
- *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for information about the BLSQMDEF, BLSQMFLD, and BLSQSHDR macros

End Programming Interface Information

Programming Interface Information

Non-IPCS Macros for IPCS Customization

Table 3 on page 9 lists other mapping macros that you may need when customizing IPCS.

Table 3. Non-IPCS Mapping Macros Needed for IPCS Customization

Macro	In Data Areas Books Under:	Used for the Following:
AHLFFAP	FFAP	To write **GTFTRACE formatting appendages** and **GTFTRACE filter/analysis exit routines**.
AHLWKAL	WKAL	To write **GTFTRACE formatting appendages**.
AHLZGTO	GTO	To write **GTFTRACE formatting appendages** and **GTFTRACE filter/analysis exit routines**.
AHLZGTS	GTS	To write **GTF formatting appendages** and **GTFTRACE filter/analysis exit routines**.
CSVMODI	CSVMODI	To retrieve information about loaded modules.
IHASMDLR	SMDLR	To map dump record headers that describe data in the record in a summary dump data contained in an SVC dump. The **summary dump data access service** allows you to access these dump records.
IKJCPPL	CPPL	To write a **BLSUGWDM validity check routine**, which is used to disable access to TSO/E commands.
ITTCTE	ITTCTE	To create an installation-provided CTRACE entry. Use field CTEFMTID (format ID key) to provide an index into the **CTRACE format table**.
ITTCTXI	CTXI	To create **CTRACE filter/analysis (CTRF) exit routines** and **CTRACE format tables**.

Table 3. Non-IPCS Mapping Macros Needed for IPCS Customization (continued)		
Macro	**In Data Areas Books Under:**	**Used for the Following:**
ITTFMTB	See *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*	To create the **CTRACE format table**, used in conjunction with the **CTRACE buffer find exit routine** and the **CTRACE filter/analysis exit routine**. To create unique trace identifiers. Each trace identifier contains: • Mnemonic name to describe the type of event (MNEMONIC parameter) • Entry description (DESCRIPTION parameter) • Formatting program or model for formatting the entry (FORMATNAME, FORMATADDR, MODELNAME, MODELADDR parameters) • Address space identifier (ASID) offsets (OFFSETASID parameter) • JOBNAME offsets (OFFSETJOBNAME parameter) • Exception indicator (EXCEPTION/NOEXCEPTION parameter) • Summary formatting view (VIEWSUMMARY parameter) • Full formatting view (VIEWFULL parameter)

See the following topics for more information:

- Chapter 7, "Providing Security for IPCS," on page 47 for information about creating a BLSUGWDM validity check routine
- Chapter 8, "Writing IPCS Exit Routines," on page 49 for information about using the mapping macros to write IPCS exit routines
- "CTRACE Buffer Find Exit Routine" on page 67 for the CTRACE buffer find exit routine
- "CTRACE Format Table" on page 64 for the CTRACE format table
- *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for the ITTFMTB macro
- For macro mappings: *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

End Programming Interface Information

Programming Interface Information

IPCS Exit Services

Table 4 on page 10 lists the IPCS exit services available to IPCS exit routines. Services that are invoked through the exit services router are preferred to those services invoked in other ways because you can use the debug tool on services invoked through the router; the debug tool is used through the IPCS subcommands GO, TRAPLIST, TRAPOFF, and TRAPON.

Table 4. Invoking Exit Services		
Service	**Recommended?**	**See the following:**
Add symptom	Yes	Chapter 10, "IPCS Exit Services," on page 95
Control block formatter	Yes	Chapter 10, "IPCS Exit Services," on page 95
Control block status (CBSTAT)	Yes	Chapter 10, "IPCS Exit Services," on page 95
Contention queue element (CQE) create	Yes	Chapter 10, "IPCS Exit Services," on page 95
Dump index	No, use table of contents service	Chapter 12, "IPCS Exit Services Supported for Compatibility," on page 167
Equate symbol	Yes	Chapter 10, "IPCS Exit Services," on page 95

Table 4. Invoking Exit Services (continued)

Service	Recommended?	See the following:
Exit control table (ECT)	Yes	Chapter 10, "IPCS Exit Services," on page 95
Expanded print	Yes	Chapter 10, "IPCS Exit Services," on page 95
Format	No, use control block formatter or format model processor service	Chapter 12, "IPCS Exit Services Supported for Compatibility," on page 167
Format model processor	Yes	Chapter 10, "IPCS Exit Services," on page 95
Get symbol	Yes	Chapter 10, "IPCS Exit Services," on page 95
Name	Yes	Chapter 10, "IPCS Exit Services," on page 95
Name/token lookup	Yes	Chapter 10, "IPCS Exit Services," on page 95
Old storage access	No, use storage access service for exit routines that do SNAP or both SNAP and IPCS formatting. Use storage map or symbol service for exit routines that do IPCS formatting.	Chapter 12, "IPCS Exit Services Supported for Compatibility," on page 167
Quiesce IPCS transaction	Yes	Chapter 10, "IPCS Exit Services," on page 95
Print	No, use standard print or expanded print service	Chapter 12, "IPCS Exit Services Supported for Compatibility," on page 167
Select address space identifier (ASID)	Yes	Chapter 10, "IPCS Exit Services," on page 95
Standard print	Yes, but use expanded print service, especially for exit routines that issue messages with prefixes	Chapter 10, "IPCS Exit Services," on page 95
Storage access	Yes	Chapter 10, "IPCS Exit Services," on page 95
Storage map	Yes	Chapter 10, "IPCS Exit Services," on page 95
Summary dump data access	No, use symbol service with an ACC code that identifies the summary dump as desired storage	Chapter 12, "IPCS Exit Services Supported for Compatibility," on page 167
Symbol	Yes	Chapter 10, "IPCS Exit Services," on page 95
Table of contents	Yes	Chapter 10, "IPCS Exit Services," on page 95
Time of day clock	Yes	Chapter 10, "IPCS Exit Services," on page 95
WHERE	Yes	Chapter 10, "IPCS Exit Services," on page 95
17-character time stamp	Yes	Chapter 10, "IPCS Exit Services," on page 95
26-character time stamp	Yes	Chapter 10, "IPCS Exit Services," on page 95

See the following topics for more information:

- "IPCS Macros and Mapping Macros" on page 5 for the macros used to map the exit services parameter lists
- Chapter 8, "Writing IPCS Exit Routines," on page 49 for information about invoking these services in an IPCS exit routine
- Chapter 10, "IPCS Exit Services," on page 95 for information about invoking and using these services
- Chapter 11, "The IPCS Debug Tool," on page 161 for information about using the IPCS debug tool to debug IPCS exit routines that use exit services

End Programming Interface Information

Customizing the IPCS Installation Package

Table 5 on page 12 summarizes the various IPCS customization possibilities.

Table 5. Customizing the IPCS Installation Package

To Customize or Create:	Do the Following:	Reference
Access to IPCS	To simplify or customize access, use any of the following: • A TSO/E LOGON procedure • A customized ISPF/PDF selection panel, if IPCS is set up in a TSO/E environment • User-written CLISTs with the TSO/E ALLOCATE command • JCL	See Chapter 4, "Customizing Access to IPCS," on page 25
Access to the IPCS dialog	Use either a TSO/E LOGON procedure or a customized ISPF/PDF selection panel.	See Chapter 4, "Customizing Access to IPCS," on page 25
IPCS exit routines	To write exits, use the following: • IPCS mapping macros • IPCS macros • IPCS exit services • IPCS debug tool To install exits, use one or more of the following: • SYS1.MIGLIB • SYS1.LPALIB • SYS1.PARMLIB • Alternate parmlibs	See the following: • Chapter 8, "Writing IPCS Exit Routines," on page 49 and Chapter 10, "IPCS Exit Services," on page 95 for information about using IPCS, macros, mapping macros, and exit services to write IPCS exit routines • Chapter 9, "Installing IPCS Exit Routines," on page 93 for information about installing the exit routines • Chapter 11, "The IPCS Debug Tool," on page 161 for information about debugging IPCS exit routines
IPCS dialog	To customize the dialog, do the following: • Use the ISPF LIBDEF service. • Use ISPEXEC to invoke the ISPF SELECT service with the IPCS dialog programs. To create your own dialog, create your own panels.	See the following: • Chapter 5, "Customizing the IPCS Dialog," on page 37 • *z/OS ISPF Dialog Tag Language Guide and Reference* for how to create dialog panels
Dump directory copies	Use the IPCS COPYDDIR subcommand.	See *z/OS MVS IPCS Commands*
Defaults for user and sysplex dump directories	Use the BLSCDDIR CLIST.	See the following: • Chapter 3, "Customizing the Dump Directory," on page 21 • *z/OS MVS IPCS User's Guide*
IPCS session parameters	Create either: • One or more IPCSPRxx members in an alternate parmlib • One or more IPCSPRxx parmlib members in SYS1.PARMLIB	See Chapter 2, "Customizing IPCS Session Parameters," on page 17
IPCS subcommand parameters	Use the IPCS SETDEF subcommand.	See *z/OS MVS IPCS Commands*

Table 5. Customizing the IPCS Installation Package (continued)		
To Customize or Create:	**Do the Following:**	**Reference**
Line widths for IPCS reports	Use the IPCS PROFILE subcommand.	See *z/OS MVS IPCS Commands*
Security for IPCS sessions	Create a BLSUGWDM validity check module.	See Chapter 7, “Providing Security for IPCS,” on page 47

Customizing Data Privacy for Diagnostics

The Data Privacy for Diagnostics Analyzer provides the facilities to scan and identify data within dumps that may be sensitive personal information (SPI). The Data Privacy for Diagnostics Analyzer runs via batch jobs and utilizes the zFS file system to retain all its required input, and as a repository for its reports. Required inputs include dictionaries used to identify SPI. Reports can be generated to help a user understand what caused pages to be flagged as containing sensitive data. Users will be able to provide feedback by updating this information in the file system and running the feedback analysis tool to improve the SPI data detection/analysis.

In order to use the Analyzer, some initial set up must be performed. To help with this set up, a sample batch job has been provided. The batch job will create and initialize the file system, mount it to the desired mount point, and run an initialization shell script. See 'SYS1.SAMPLIB(BLSDPJIN)' for instructions on how to modify the sample batch job example to run it on a different system(s).

One consideration that should be given is to the access control for this file system. Some of the sub-directories may contain sensitive data that has been extracted from dumps, or data that has been ingested by your customization for dump analysis. This data may be in reports, in files after feedback has been given and data has been ingested. Therefore, you want to ensure that only intended personnel have access to these folders. One option for you is to use FSACCESS control user access to this data. See "**z/OS Security Server RACF Security Administrator's Guide**" for more information on using FSACCESS to control access to file systems.

Once this initial set up is complete, a user will want to ensure that the new file system is always mounted on the systems where the analysis will be run. The user may choose to update the appropriate BPXPRMxx SYS1.PARMLIB members to ensure that the mount processing occurs.

Programming Interface Information

The Data Privacy for Diagnostics Analyzer File System

The set-up job BLSDPJIN performs the following steps:

- Creates the file system
- Formats the file system
- Creates the home directory
- Mounts the file system to the home directory
- Runs the initialization shell script (blsdpdp.sh note)

The file system has a required folder structure. The following describes these sub-directories and their contents:

/<directory>/knowledgebase
This folder is used to store the ingested knowledge and user feedback.

/<directory>/knowledgebase/ingested/
This folder stores the ingested knowledge such as user provided directories and regular expressions, and is populated by the INGEST function.

/<directory>/knowledgebase/feedback/
This folder stores the processed user feedback and is populated by the FEEDBACK function.

/<directory>/configuration
This folder stores the configuration which is used for various operations carried out by Data Privacy for Diagnostics Analyzer. This folder will contain the following configuration files (which correspond to the ANALYZE, INGEST and EXTRACT modes of operations).

/<directory>/configuration/analysis_config.json
This file contains configuration about the sensitivity analysis to be carried out on dump. It allows customizing which built-in identifiers and ingested information that should be used for analysis. It also allows you to customize which combination of identifiers should be present together for data to be considered sensitive. See the z/OS MVS IPCS User's Guide for additional information about the analysis_config.json file including parameters and examples.

/<directory>/configuration/extract_config.json
This file contains configuration about identifiers which are to be extracted to a file. It allows the user to display the current pattern or dictionary associated with a built-in or customer identifier that is available for the ANALYZE function in determining which data is to be marked as sensitive by the Analyzer. See the z/OS MVS IPCS User's Guide for additional information about the extract_config.json file including parameters and examples.

/<directory>/configuration/ingestion_config.json
This file contains configuration about user provided data to be ingested. The ingested data is then available for use in future ANALYZE runs. See the z/OS MVS IPCS User's Guide for additional information about the ingestion_config.json file including parameters and examples.

/<directory>/reports
This folder is used to store reports generated by Data Privacy for Diagnostics Analyzer. A subdirectory is created for each dump on which Data Privacy for Diagnostics Analyzer ANALYZE processing is requested. The following folder structure is generated for each dump:

/<directory>/reports/<dump-name-1>/
This folder stores reports from each invocation of ANALYZE.

/<directory>/reports/<dump-name-1>/<timestamp-of-Data Privacy for Diagnostics Analyzer-ANALYZE-invocation>/
Stores reports of a single ANALYZE invocation. It contains the following files:

../concise_sensitive_report_<i>
This file is generated by each thread spawned by ANALYZE to process the dump.

../sensitive_token_log_<i>
This file is generated by each thread spawned by ANALYZE to containing all of the sensitive tokens identified in the dump. These files are generated by each thread spawned by the ANALYZE function if the value of the SENSITIVE REPORT field is Y on the IPCS ANALYZE panel or if the log_sensitive_tokens value is set to TRUE in the BLSJDPA JCL that invokes the ANALYZE function.

../non_sensitive_tokens
This file contains all the non-sensitive tokens identified in the dump along with their count. This file is generated when REPORT is requested after ANALYZE that requested token level redaction. Token level redaction can be requested by specifying the value N for the ALLOW PAGE LEVEL option on the ANALYZE IPCS panel or by specifying the value 2 for the analysis_mode option in the BLSJDPA JCL. This file can be modified to provide feedback about tokens which are incorrectly marked as non-sensitive.

End Programming Interface Information

Programming Interface Information

Data Privacy for Diagnostics Analyzer Directory Maintenance

As more and more dumps are run through the various Data Privacy for Diagnostics Analyzer functions, the file system usage will grow. You should periodically determine if older dump analysis directories are no longer required, and remove those sub-directories and their contents.

Note: The blsdpdp.sh initialization shell script that runs at the end of the BLSDPJIN setup job will not update the analysis_config.json, ingestion_config.json and extract_config.json files in <directory>/configuration if those files are present in that directory when the script is run. To obtain the latest changes, you can preserve the contents of your current configuration files by renaming them or simply delete them if not needed. Then run the shell script to create newly updated versions of those configuration files. Lastly, merge the installation's changes from the renamed files into the resulting files as the Analyzer will only use the configuration files with the names analysis_config.json, ingestion_config.json and extract_config.json

End Programming Interface Information

Chapter 2. Customizing IPCS Session Parameters

IPCSPRxx parmlib members contain the parameters that control an IPCS session. This chapter describes the session parameters.

Session Parameters

IPCS session parameters appear in an IPCSPRxx parmlib member and can specify the values shown in Table 6 on page 17.

Table 6. IPCS Session Parameters Specified in an IPCSPRxx Parmlib Member

Session Parameter	Specifies:	Default in IPCSPR00:
DSD(*dsn*)	The data set name of the IPCS data set directory, which is used for problem and data set management.	Not supplied
NODSD	The suppression of problem and data set management use.	NODSD
PDR(*dsn*)	The data set name of the IPCS problem directory.	Not supplied
NOPDR	The suppression of problem and data set management use.	NOPDR
PROBIDPREFIX(*prefix*)	The 3-character value used to form a problem identifier.	Not supplied
SYSTEM(*system-id*)	The default system identifier.	Not supplied
GROUP(*group-id*)	The default group identifier.	Not supplied
ADMINAUTHORITY (*userid-list*)	The TSO/E user IDs of the persons with IPCS administrative authority.	Not supplied
DELETEAUTHORITY (*userid-list*)	The TSO/E userids of the persons with IPCS delete authority.	Not supplied
LINELENGTH(*value*)	The default logical record length (LRECL) for the IPCS print output data set.	LINELENGTH(137)
PAGESIZE(*value*)	The default number of lines per page for the IPCS print output data set.	PAGESIZE(60)

See *z/OS MVS Initialization and Tuning Reference* for more information about the session parameters contained in a IPCSPRxx parmlib member.

The session parameters include:

- Problem and data set management parameters. These parameters are usually not needed if the installation uses the recommended IBM Information/Family programs for problem management.
- IPCS print data set defaults. These are needed only if you want to print IPCS output.

IPCS supplies default session parameters in IPCSPR00, including the print data set defaults, but not the problem and data set management functions. When starting an IPCS session, most IPCS users enter the IPCS NOPARM command to indicate that IPCS is not to use session parameters specified in IPCSPR00 or any other IPCSPRxx parmlib member. Note that NOPARM does not affect the use of the BLSCECT parmlib member. If NOPARM is specified, you cannot use the IPCS problem management or data set management subcommands.

Customizing Session Parameters

If you choose to use session parameters other than the defaults, IPCS allows installations to create one or more IPCSPRxx parmlib members to customize some or all IPCS sessions. To customize the IPCS session parameters, do one, or all, of the following:

- Customize the IBM-supplied IPCSPR00 parmlib member.
- Create one or more IPCSPRxx parmlib members.
- Create an alternate parmlib containing one or more IPCSPRxx members.

Indicate which IPCSPRxx parmlib member IPCS is to use for session parameters on the IPCS command that begins an IPCS session. See "Indicating Which IPCSPRxx Member IPCS Should Use" on page 18.

Creating an Alternate Parmlib

To create an alternate parmlib containing one or more IPCSPRxx members, do the following:

1. Make sure that you are not in an IPCS session.
2. Edit a copy of IPCSPRxx into an alternate parmlib called DEPTXYZ.PARMLIB, for example. Call the copy IPCSPR05, for example.
3. Specify the parameters in the IPCSPR05 member of DEPTXYZ.PARMLIB.
4. Enter the following TSO/E command:

   ```
   allocate file(ipcsparm) dsn('deptxyz.parmlib' 'sys1.parmlib') shr
   ```

   The result of this allocate command is that IPCS will use IPCSPR05 in DEPTXYZ.PARMLIB for the session parameters.
5. Begin an IPCS session by entering the following command so that IPCS will syntax-check IPCSPR05:

   ```
   IPCS PARM(05)
   ```

All users needing special session parameters can store customized IPCSPRxx members in DEPTXYZ.PARMLIB. Users do not need update authority to SYS1.PARMLIB to tell IPCS which problem and data set directories have been established for users.

Indicating Which IPCSPRxx Member IPCS Should Use

If you choose to customize the IPCSPR00 parmlib member or create an IPCSPRxx member containing session parameters, you can indicate which member IPCS should use for session parameters as follows:

- **To Bypass the Use of Any IPCSPRxx Member**

  Begin your IPCS session by entering **IPCS NOPARM**. IPCS will not use session parameters.
- **To Use the IBM-Supplied or Installation-Customized IPCSRP00 Member**

  Begin your IPCS session by entering **IPCS**. Do not enter the **PARM** or **NOPARM** parameter on the IPCS command. IPCS will use the IPCSPR00 parmlib member for session parameters.
- **To Use an Installation-Provided IPCSPRxx Member**

  Begin your IPCS session by entering **IPCS PARM(*xx*)**, where ***xx*** is the suffix of the IPCSPR***xx*** parmlib member containing the session parameters to be used.

See *z/OS MVS IPCS User's Guide* for information about the IPCS command used to start an IPCS session.

Using Problem and Data Set Management Facilities

IPCS users who previously depended upon the IBM-supplied IPCSPR00 to specify the data set name for IPCS problem and data set management and the IPCS problem directory will have to add the following lines to the IPCSPR00 or a IPCSPRxx parmlib member:

```
DSD(IPCS.DATA.SET.DIRECTRY)
PDR(IPCS.PROBLEM.DIRECTRY)
```

These data sets are usually not needed and should not be allocated if the installation uses the recommended IBM Information/Family programs for problem management.

Chapter 3. Customizing the Dump Directory

Users or installations who use IPCS extensively or want to improve IPCS performance can customize the dump directories:

- **User dump directory:** Each IPCS user has a user dump directory, which lists the sources the user can process with IPCS. IPCS uses it while processing the user's subcommands and commands.
- **Sysplex dump directory:** This directory lists the SVC dumps produced by systems in the sysplex and can list other sources. A user can transfer the source description for a dump, trace, or data set from the sysplex dump directory to the user's current user dump directory.

The directory used by a subcommand is the directory allocated with ddname IPCSDDIR.

Customizing the Directories

To customize all user dump directories created by IPCS users at an installation or to customize the sysplex dump directory, the installation can alter the dump directory defaults by:

- Running the BLSCDDIR CLIST with appropriate DSNAME, RECORDS, or VOLUME parameter values before users invoke IPCS
- Editing the BLSCDDIR CLIST to specify the name, size, or volume for user dump directories

In TSO/E, the IPCS command uses the defaults in BLSCDDIR to dynamically create a user dump directory for an IPCS session.

See the *z/OS MVS IPCS User's Guide* for information about starting an IPCS session, including creating a dump directory, and the TSO/E commands used to create, initialize, and allocate a dump directory. See *z/OS MVS IPCS Commands* for invoking the BLSCDDIR CLIST and descriptions of the DSNAME, RECORDS, or VOLUME parameters.

Editing the BLSCDDIR CLIST

The BLSCDDIR CLIST is provided in the installation data set SBLSCLI0. Do not edit the BLSCDDIR CLIST directly as it is SMP/E managed, and if the part is updated by IBM, installation updates may be lost. Instead, either apply the update as a USERMOD to allow for notification of changes, or copy the CLIST to a non-installation directory and perform updates there.

An installation can edit a copied BLSCDDIR CLIST to change dump directory defaults supplied by the BLSCDDIR CLIST, such as:

- The dump directory name
- The dump directory size
- The dump directory volume

It is recommended that you use storage management subsystem (SMS) constructs established at your installation, such as management class and data class, in preference to using the VOLUME parameter in the IBM supplied BLSCDDIR CLIST.

You can place a directory in a basic format, large format or extended format sequential data set. The directory must be a key-sequenced data set (KSDS). It can be extended format but it cannot be in compressed format.

To change the user dump directory defaults in BLSCDDIR CLIST, do one or more of the following:

- Change the DSNAME assignment in the BLSCDDIR CLIST to contain a new data set name for the directory. The name should be fully-qualified.

- Change the RECORDS assignment in the BLSCDDIR CLIST to specify a new number of records to maintain a larger dump directory. The default dump directory size is 5000 records. The number should be the records accommodated by the primary and secondary dump directory allocation.

  For most processing, determine the space needed for a user dump directory by using the following rule:

  Number of tracks allocated = 5% of the total number of tracks in the dumps to be represented in the directory.

  An installation should try to find a default for space that will provide the least-active IPCS users with enough space to do their work with a directory that uses just one extent. Then the most active IPCS users' directories should be examined to verify that they are no more than 256 times as large as the default you chose. If not, either:

  1. Allocate space for dump directories using an amount that will accommodate the more active IPCS users. This may be quite acceptable in installations where tools like the hierarchical storage manager (HSM) are used to archive little-used data sets. Because HSM uses data compression techniques to save space on archival storage, the over-allocated user dump directories of the least active IPCS users will not cause storage problems.
  2. Warn the more active IPCS users that they should change the defaults provided by BLSCDDIR when they allocate replacement user dump directories, using the procedure in the *z/OS MVS IPCS User's Guide*.
- Change the VOLUME assignment in the BLSCDDIR CLIST to specify a new volume serial for user dump directories. The default volume serial for directories is VSAM01.

  VSAM allows 256 extents of a data set to be allocated on each volume, so you will probably be able to find a value that satisfies both casual and extensive IPCS users.

  If all user dump directories cannot be placed on a single volume, add the following to the BLSCDDIR CLIST:

```
IF &LENGTH(&VOLUME) = 0 THEN +
  IF &SUBSTR(1:1,&SYSUID)=A THEN SET VOLUME=VOLA
  ELSE IF &SUBSTR(1:1,&SYSUID)=B THEN SET VOLUME=VOLB
  .
  .
  .
```

Dump Directories and Performance

IPCS uses a user dump directory to process dumps efficiently. IPCS determines the most efficient method of mapping storage and creating symbols for its use. IPCS and VSAM, however, sometime do not agree on the most efficient use of the dump directory. You cannot alter the IPCS code, but you can do several VSAM-related things to your dump directory to try to improve performance.

- **Using System-Managed Storage**

  To use system-managed storage, specify in the BLSDDIR CLIST the parameters: DATACLAS, MGMTCLAS, and/or STORCLAS.
- **Changing the Control Interval Size**

  One of the potential bottlenecks in IPCS processing is the number of control interval splits in the dump directory. Periodically review the rate of control interval splits your IPCS users experience. To minimize splits, adjust the NDXCISZ parameter value in the BLSCDDIR CLIST to tune the size of the index and data in the dump directory.
- **Copying Source Descriptions**

  The COPYDDIR subcommand allows VSAM to reorganize the source descriptions as it copies descriptions from one dump directory to another. To take advantage of COPYDDIR, you can modify your installation's dump management procedures as follows:

  1. Have the jobs that produce initial problem screening reports save the source descriptions in the sysplex dump directory.

2. Have individual users invoke COPYDDIR to copy the source description to their own user dump directories. Not only to the users avoid IPCS initialization processing, but the source description is less fragmented.

- **Refreshing Dump Directories**

  You or an individual user can invoke the TSO/E REPRO command to copy the contents of a dump directory to a temporary data set, then back into the dump directory. This allows VSAM to reorganize the data in the dump directory. See *z/OS MVS IPCS User's Guide* for a sample CLIST to refresh the dump directory.

- **Changing the Buffer Space**

  You or an individual user can use the TSO/E ALTER command to change the amount of virtual storage IPCS uses to buffer the data portion of a dump directory. IPCS will override the value specified if space for less than 16 control intervals or more than 1024 control intervals is specified.

Chapter 4. Customizing Access to IPCS

This chapter describes how to customize access to IPCS, and is written primarily for the system programmer who wants to customize access to IPCS for users at the installation. However, individual users can also follow these procedures to customize access to IPCS for themselves.

IBM recommends that all installations customize access for IPCS users. By providing customized access, you give users a convenient way to start IPCS and ensure that the method for starting IPCS is consistent with the standards of the installation.

Planning for Customized Access

Before you begin any actual customization, you need to make two decisions related to setting up an IPCS session for the user:

- The method to use for starting IPCS
- The function to use for invoking the IPCS dialog.

Regardless of the method for starting IPCS or the function for invoking the IPCS dialog, IBM recommends that you:

- Add an IPCS option to an ISPF selection panel.

  As supplied by IBM, there is no option for selecting the IPCS dialog on any ISPF panel. To provide a simple means for users to start the IPCS dialog, you should add an option for the IPCS dialog to an ISPF selection panel.
- Create a TSO/E logon procedure that the user can invoke to set up the environment for the IPCS session.

  This customization might also involve providing an accompanying CLIST that is invoked by the logon procedure. The logon procedure and CLIST can automate various functions for the user, such as:

  – Concatenating the IPCS libraries
  – Starting IPCS in line mode
  – Starting an ISPF session.

There is one other point that you should consider when providing customized access to IPCS: If the definition for the dump directory is not managed through the storage management subsystem (SMS), IBM recommends that you edit the BLSCDDIR CLIST to specify the default volume for the dump directory. See Chapter 3, “Customizing the Dump Directory,” on page 21 for further information.

Note: You can use different release levels of IPCS on the same system. The examples in this chapter show only how to access the level of IPCS that matches the system's release level (assuming IPCS libraries are not renamed at your installation). See Chapter 6, “Using IPCS on Another System,” on page 45 for information on using different release levels of IPCS.

Decisions to Make Before Starting Customization

Before you begin any actual customization, you should decide which method for starting IPCS you want to provide for the user. There are two basic methods, and there are certain advantages to each.

You should also decide which function to use for invoking the IPCS dialog. Part of the recommended customization involves adding an IPCS option to an ISPF selection panel. The option that you provide will use one of the functions for invoking the IPCS dialog.

This section describes each of these decisions, explains the choices available, and helps you determine which choices are best suited to your requirements. The specific customization that you provide depends on the decisions you make.

Starting IPCS: Deciding Which Method to Provide

There are two basic ways for users to start the IPCS dialog:

- **Start the IPCS dialog directly from ISPF without starting IPCS in line mode**

  In this case, the user invokes the IPCS dialog directly from ISPF, without entering the IPCS command before starting ISPF.

  Requirement

  To start IPCS without entering the IPCS command, you must run IPCS under the terminal monitor program supplied with TSO/E Version 2.4.
- **Start IPCS in line mode before starting ISPF**.

  In this case, the user must perform the following steps to start the IPCS dialog:

  1. Enter the IPCS command to start IPCS line mode before starting ISPF.
  2. Start an ISPF session.
  3. Invoke the IPCS dialog from ISPF.

This section describes the two methods and compares their relative advantages and disadvantages.

Comparison of the Two Methods

In most cases, starting the IPCS dialog directly from ISPF (without first starting IPCS in line mode) is probably the best method to use. This method makes the most efficient use of system resources, in that the storage needed to run IPCS is not required until the user actually invokes the IPCS dialog.

In contrast, starting IPCS line mode before beginning an ISPF session causes system resources for IPCS to be held for the entire ISPF session. This use of resources is of no real consequence if the user plans to use IPCS for most of the time the ISPF session is active. Most users, however, do not need to use IPCS for a large portion of their ISPF session. For such users, entering the IPCS command before starting ISPF causes unnecessary use of system resources, and a corresponding decrease in performance for other ISPF services.

However, there are certain situations in which it might be desirable to start IPCS in line mode before starting ISPF:

- The user plans to use IPCS many different times during an ISPF session.

  The IPCS dialog starts more quickly, once invoked, when the IPCS command has been entered to start IPCS before starting the ISPF session. For users who plan to invoke the IPCS dialog many times during the course of an ISPF session, the benefit of improved response time might outweigh the disadvantage of reduced performance for other ISPF services.
- The user plans to use IPCS in line mode.

  Entering the IPCS command is the only way to start IPCS in line mode.
- The user needs the problem management and data set management functions.

  These functions are available only by entering the IPCS command with the PARM(nn) parameter. See *z/OS MVS IPCS User's Guide* and *z/OS MVS IPCS Commands* for further information on managing problems and data sets.

Even if a customized access procedure has started an ISPF session without entering the IPCS command, users can still enter the IPCS command explicitly. The user can exit ISPF, enter the IPCS command, and either use IPCS in line mode or re-enter ISPF to invoke the IPCS dialog.

Invoking the IPCS Dialog: Deciding Which Function to Use

There are three functions that you can use to invoke the IPCS dialog:

- The BLSCLIBD CLIST

- The BLSG dialog program
- THE BLSGLIBD dialog program

The BLSCLIBD CLIST is usually the preferred function for invoking the IPCS dialog when users do not need to run different release levels of IPCS on the same system. Most people use IPCS for only a small portion of their ISPF session, and do not start and exit the IPCS dialog more than a few times during an ISPF session. With BLSCLIBD, the libraries required for the IPCS dialog are not concatenated until the user actually invokes the IPCS dialog. This method of concatenating the IPCS libraries provides an overall improvement in the performance of ISPF services for most IPCS users.

Where it is necessary to run different release levels of IPCS on the same system, the BLSG dialog program must be used to invoke the IPCS dialog. BLSG might also be preferred in cases where the user intends to enter and exit the dialog frequently during an ISPF session. The libraries required for the IPCS dialog must be concatenated before BLSG is invoked, and therefore they do not have to be concatenated each time the user invokes BLSG. As a result, there is improved response time when starting and exiting the IPCS dialog. BLSG also allows you to use private versions of IPCS dialog panels.

The BLSGLIBD dialog program performs a function similar to that of the BLSCLIBD CLIST. The difference is that BLSGLIBD does not set up SYS1.SBLSCLI0 as an ALTLIB application CLIST library for the IPCS dialog session, whereas BLSCLIBD does. Therefore, in most cases, IBM recommends that you use the BLSCLIBD CLIST instead of the BLSGLIBD dialog program. You might, however, want to use BLSGLIBD if you have modified the IPCS dialog to include private versions of IPCS panels, but want a function similar to that of BLSCLIBD. Because you cannot use the BLSCLIBD CLIST to access these panels, you need to write your own CLIST. In this case, include BLSGLIBD as the step of your CLIST that invokes the IPCS dialog.

The following sections describe the functions for invoking the IPCS dialog and the LIBDEF service.

BLSCLIBD CLIST - Activate IPCS Dialog Services

The BLSCLIBD CLIST requires the ISPF LIBDEF service introduced in ISPF Version 2 Release 2.

Invoke the BLSCLIBD CLIST through one of the following methods:

- Use the TSO/E EXECUTE command, entered on the TSO commands option in the ISPF dialog:

```
EX 'SYS1.SBLSCLI0(BLSCLIBD)'
```

- Add an option to an ISPF panel that users can select to start the IPCS dialog. The line to add to the processing section of the panel is:

```
I,'CMD(%BLSCLIBD) NEWAPPL(BLSG) PASSLIB'
```

BLSCLIBD invokes the BLSCALTL CLIST. This CLIST invokes the TSO/E ALTLIB command to set up SYS1.SBLSCLI0 as an ALTLIB application CLIST library for the IPCS dialog session.

BLSCLIBD requests the following ISPF services and sets up the allocations for these data sets through the LIBDEF service:

```
ISPEXEC LIBDEF ISPMLIB DATASET ID('SYS1.SBLSMSG0') COND
ISPEXEC LIBDEF ISPPLIB DATASET ID('SYS1.SBLSPNL0') COND
ISPEXEC LIBDEF ISPSLIB DATASET ID('SYS1.SBLSKEL0') COND
ISPEXEC LIBDEF ISPTLIB DATASET ID('SYS1.SBLSTBL0') COND
```

Through these allocations, BLSCLIBD supplements the allocations made in a logon procedure or in a startup CLIST. With BLSCLIBD, the libraries and data sets required for IPCS are not allocated until the user actually requests the IPCS dialog. The fact that BLSCLIBD allocates these resources only when needed is an advantage for most people, who typically use IPCS for only a small portion of their ISPF session. By contrast, when a logon procedure or startup CLIST allocates the libraries and data sets required for IPCS, these resources are searched for materials during periods when other facilities are being used. The extra time required for this search can slow the performance of ISPF services for other facilities. Also, allocations of data sets may be impacted, if the user has many other data sets to allocate and the installation has defined a maximum number of allocations per user.

By using BLSCLIBD, the libraries and data sets needed for IPCS are not allocated until the IPCS dialog is actually requested.

Restriction: You can define and use up to 10 global variables in CLISTs invoked through the IPCS dialog, if CLIST BLSCLIBD started the IPCS dialog. IPCS does not restrict the number of global variables you can define when the IPCS dialog is started using other approved methods. If CLIST BLSCLIBD started the IPCS dialog, and if you require more than 10 global variables, create your own copy of CLIST BLSCALTL and add more global variables. Modify CLIST BLSCLIBD to point to your copy of BLSCALTL rather than to SYS1.SBLSCLI0(BLSCALTL). For information about defining and using global variables, see *z/OS TSO/E CLISTs*.

For more information about BLSCLIBD, see "Customizing Access when Using the BLSCLIBD CLIST" on page 31.

BLSG Dialog Program - Activate IPCS Dialog Services

Invoke the BLSG dialog program through one of the following methods:

- Use ISPEXEC with the ISPF SELECT service. From ISPF, enter:

```
ISPEXEC SELECT PGM(BLSG) PARM(PANEL(BLSPPRIM)) NEWAPPL(BLSG) PASSLIB
```

Using Alternate Panels: You can use panels other than BLSPPRIM as the entry to the IPCS dialog, and you can use command and program logic to determine which panel to display at the time the dialog is activated. To specify an alternate panel, replace PANEL(BLSPPRIM) with another value that the ISPF SELECT service accepts, as shown in the following examples:

Example 1: Specify that ISPF is to display an installation-written panel called PANLIPCS instead of BLSPPRIM as the initial panel of the IPCS dialog.

```
ISPEXEC SELECT PGM(BLSG) PARM(PANEL(PANLIPCS)) NEWAPPL(BLSG) PASSLIB
```

Example 2: Specify that ISPF is to process an installation-written command called CMDIPCS that determines which panel is displayed as the initial panel of the IPCS dialog.

```
ISPEXEC SELECT PGM(BLSG) PARM(CMD(CMDIPCS)) NEWAPPL(BLSG) PASSLIB
```

This option might be useful if IPCS users at the installation specialize in different areas, and you want to provide panels tailored to the specific needs of each area. This command can be implemented using a CLIST or REXX exec, or as an unauthorized TSO/E command.

Example 3: Specify that ISPF is to process an installation-written dialog program called PGMIPCS that displays installation-developed dialog panels.

```
ISPEXEC SELECT PGM(BLSG) PARM(PGM(PGMIPCS)) NEWAPPL(BLSG) PASSLIB
```

Alternate Panels and Performance: If you are using alternate panels to create a customized dialog, keep in mind that the point at which you invoke BLSG can affect performance. While BLSG is active, it serves as the owner of data sets used to support ISPF logical screens. BLSG closes these data sets when the SELECT service returns control to BLSG and BLSG ends. You can structure the dialog using either of the following methods:

- Invoke BLSG before displaying the primary option panel
- Display the primary option panel first, and then invoke BLSG when the user requests a specific IPCS function from the primary option panel.

The first method is more efficient, in that IPCS releases the resources it has accumulated only when the user exits the IPCS dialog. With the second method, IPCS releases resources after each request completes. Thus, with the second method, response time is longer when the user requests different functions, because IPCS must prepare and release resources as part of each request. You might want to

use the second method, however, if other services that you want to provide during the ISPF session will not operate effectively while IPCS holds resources.

- Use the ISPSTART command. From TSO/E READY or from IPCS line mode, enter:

```
ISPSTART PGM(BLSG) NEWAPPL(BLSG) PASSLIB PARM(PANEL(BLSPPRIM))
```

- Add an option to an ISPF panel that users can select to start the IPCS dialog. The line to add to the processing section of the panel is:

```
I,'PGM(BLSG) PARM(PANEL(BLSPPRIM)) NEWAPPL(BLSG) PASSLIB'
```

Note: Specify PASSLIB only if both of the following are true:

1. You also specify NEWAPPL(BLSG).
2. You are running ISPF Version 2 Release 2 or later.

Return Codes

Table 7. BLSG Dialog Program Return Codes

Code	Explanation
00	Successful completion.
16	Ending error with two possible meanings: • The program invoked BLSG in an environment where one or more of the following is true: – IPCS is used recursively. – There is no supported level of TSO/E installed. – BLSG is invoked without first starting IPCS line mode, but TSO/E Version 2.4 is not installed. • The ISPF SELECT service generated a return code of 16.
other	Return code from the ISPF SELECT service. See *z/OS ISPF Dialog Tag Language Guide and Reference* for more information.

BLSGLIBD Dialog Program - Activate IPCS Dialog Services

The BLSGLIBD dialog program, like the BLSCLIBD CLIST, requires the ISPF LIBDEF service.

Invoke the BLSGLIBD dialog program through one of the following methods:

- Use ISPEXEC with the ISPF SELECT service. From ISPF, enter:

```
ISPEXEC SELECT PGM(BLSGLIBD) PARM(PANEL(BLSPPRIM)) NEWAPPL(BLSG) PASSLIB
```

Using Alternate Panels: You can use panels other than BLSPPRIM as the entry to the IPCS dialog, and you can use command and program logic to determine which panel to display at the time the dialog is activated. To specify an alternate panel, replace PANEL(BLSPPRIM) with another value that the ISPF SELECT service accepts, as shown in the following examples:

Example 1: Specify that ISPF is to display an installation-written panel called PANLIPCS instead of BLSPPRIM as the initial panel of the IPCS dialog.

```
ISPEXEC SELECT PGM(BLSGLIBD) PARM(PANEL(PANLIPCS)) NEWAPPL(BLSG) PASSLIB
```

Example 2: Specify that ISPF is to process an installation-written command called CMDIPCS that determines which panel is displayed as the initial panel of the IPCS dialog.

```
ISPEXEC SELECT PGM(BLSGLIBD) PARM(CMD(CMDIPCS)) NEWAPPL(BLSG) PASSLIB
```

This option might be useful if IPCS users at the installation specialize in different areas, and you want to provide panels tailored to the specific needs of each area. This command can be implemented using a CLIST or REXX exec, or as an unauthorized TSO/E command.

Example 3: Specify that ISPF is to process an installation-written dialog program called PGMIPCS that displays installation-developed dialog panels.

```
ISPEXEC SELECT PGM(BLSGLIBD) PARM(PGM(PGMIPCS)) NEWAPPL(BLSG) PASSLIB
```

Alternate Panels and Performance: If you are using alternate panels to create a customized dialog, keep in mind that the point at which you invoke BLSGLIBD can affect performance. While BLSGLIBD is active, it serves as the owner of data sets used to support ISPF logical screens. BLSGLIBD closes these data sets when the SELECT service returns control to BLSGLIBD and BLSGLIBD ends. You can structure the dialog using either of the following methods:

- Invoke BLSGLIBD before displaying the primary option panel
- Display the primary option panel first, and then invoke BLSGLIBD when the user requests a specific IPCS function from the primary option panel.

The first method is more efficient, in that IPCS releases the resources it has accumulated only when the user exits the IPCS dialog. With the second method, IPCS releases resources after each request completes. Thus, with the second method, response time is longer when the user requests different functions, because IPCS must prepare and release resources as part of each request. You might want to use the second method, however, if other services that you want to provide during the ISPF session will not operate effectively while IPCS holds resources.

- Use the ISPSTART command. From TSO/E READY or IPCS line mode, enter:

```
ISPSTART PGM(BLSGLIBD) NEWAPPL(BLSG) PASSLIB PARM(PANEL(BLSPPRIM))
```

- Add an option to an ISPF panel that users can select to start the IPCS dialog. The line to add to the processing section of the panel is:

```
I,'PGM(BLSGLIBD) PARM(PANEL(BLSPPRIM)) NEWAPPL(BLSG) PASSLIB'
```

Note: Specify PASSLIB only if both of the following are true:

1. You also specify NEWAPPL(BLSG).
2. You are running ISPF Version 2 Release 2 or later (required for BLSGLIBD).

BLSGLIBD requests the following ISPF services and sets up the allocations for these data sets through the LIBDEF service:

```
ISPEXEC LIBDEF ISPMLIB DATASET ID('SYS1.SBLSMSG0') COND
ISPEXEC LIBDEF ISPPLIB DATASET ID('SYS1.SBLSPNL0') COND
ISPEXEC LIBDEF ISPSLIB DATASET ID('SYS1.SBLSKEL0') COND
ISPEXEC LIBDEF ISPTLIB DATASET ID('SYS1.SBLSTBL0') COND
```

Like the BLSCLIBD CLIST, BLSGLIBD supplements the allocations made in a logon procedure or in a CLIST, and thus saves system resources by doing these allocations only when they are needed. Thus, the libraries and data sets needed for IPCS are not allocated until the IPCS dialog is actually requested.

BLSGLIBD then proceeds to perform the same processing as does BLSG, assuming that the same parameters are passed to both programs.

Return Codes

Table 8. BLSGLIBD Dialog Program Return Codes	
Code	**Explanation**
00	Successful completion.
16	Ending error with two possible meanings: • The program invoked BLSGLIBD in an environment where one or more of the following is true: – IPCS is used recursively. – There is no supported level of TSO/E installed. – BLSG is invoked without first starting IPCS line mode, but TSO/E Version 2.4 is not installed. • The ISPF SELECT service generated a return code of 16.
other	Return code from the ISPF SELECT service. See *z/OS ISPF Dialog Tag Language Guide and Reference* for more information.

ISPF LIBDEF Service

You can use the ISPF LIBDEF service to access IPCS messages, panels, and data set-tailoring skeletons. The use of LIBDEF is not required. However, effective use of LIBDEF can improve the performance of your IPCS dialogs. You should consider the following points about using LIBDEF:

1. Performance gains depend on the activities that the installation's IPCS users do most often.
   - LIBDEF improves performance when:
     – Users logon to TSO/E; four fewer allocations take place.
     – Users run dialogs other than the IPCS dialog; fewer libraries are searched.
   - LIBDEF slows performance when:
     – Users enter the IPCS dialog; ISPF dynamically allocates four libraries and opens them.
     – Users leave the IPCS dialog; ISPF closes and frees the four libraries.

   If users enter and leave the IPCS dialog frequently, the use of LIBDEF might slow performance.
2. LIBDEF cannot be used to access IPCS program libraries concatenated to the ISPLLIB data set.

The BLSCLIBD CLIST and the BLSGLIBD dialog program both use the LIBDEF service.

See *z/OS ISPF Dialog Tag Language Guide and Reference* for more information on the LIBDEF service.

Customizing Access

At this point, you are ready to begin customizing access to IPCS. This section describes the actual customization that IBM recommends when using either the BLSCLIBD CLIST or the BLSG dialog program as the function for invoking the IPCS dialog. The description for each of these functions includes an explanation of how to provide either method for starting IPCS with that particular function.

You should note that the examples in this section do not represent the only ways to customize access to IPCS. There are a great number of methods that you could use. The methods shown in these examples are intended to provide a simple approach to customization, an approach that meets the requirements of most installations.

Customizing Access when Using the BLSCLIBD CLIST

This section shows how to customize access to IPCS when using the BLSCLIBD CLIST to invoke the IPCS dialog. There are two primary tasks involved in this customization, as described in the following sections:

Adding an ISPF Option to Invoke the BLSCLIBD CLIST

Figure 1 on page 32 shows two lines added to the ISPF/PDF primary option menu, ISR@PRIM, to provide customized access to the IPCS primary option menu. The two lines are underlined in the figure to highlight their placement in the ISPF/PDF primary option menu. Note that the following selection is added to the part of the menu that ISPF displays on the user's screen:

```
%  I +IPCS        - IPCS problem analysis services
```

A corresponding line is added to the processing section of the menu. This line invokes the BLSCLIBD CLIST to start the IPCS dialog.

```
I,'CMD(%BLSCLIBD) NEWAPPL(BLSG) PASSLIB'
```

```
%----------------------  ISPF/PDF PRIMARY OPTION MENU  ------------------------
%OPTION  ===]_ZCMD                                                            +
%                                                        +USERID    - &ZUSER
%  0 +ISPF PARMS  - Specify terminal and user parameters   +T IME     - &ZTIME
%  1 +BROWSE      - Display source data or output listings +T ERMINAL - &ZTERM
%  2 +EDIT        - Create or change source data           +P F KEYS  - &ZKEYS
%  3 +UTILITIES   - Perform utility functions
%  4 +FOREGROUND  - Invoke language processors in foreground
%  5 +BATCH       - Submit job for language processing
%  6 +COMMAND     - Enter TSO/E command or CLIST
%  7 +DIALOG TEST - Perform dialog testing
%  8 +
.LM UTILITIES- Perform library management utility functions
%  C +CHANGES     - Display summary of changes for this release
%  I +IPCS        - IPCS problem analysis services
%  T +TUTORIAL    - Display information about ISPF/PDF
%  X +EXIT        - Terminate ISPF using log and list defaults
%
+Enter%END+command to terminate ISPF.
%
)INIT
  .HELP = ISR00003
  &ZPRIM = YES        /* ALWAYS A PRIMARY OPTION MENU      */
  &ZHTOP = ISR00003   /* TUTORIAL TABLE OF CONTENTS        */
  &ZHINDEX = ISR91000 /* TUTORIAL INDEX - 1ST PAGE         */
  VPUT (ZHTOP,ZHINDEX) PROFILE
)PROC
  &ZSEL = TRANS( TRUNC (&ZCMD,'.')
                0,'PANEL(ISPOPTA)'
                1,'PGM(ISRBRO) PARM(ISRBRO01)'
                2,'PGM(ISREDIT) PARM(P,ISREDM01)'
                3,'PANEL(ISRUTIL)'
                4,'PANEL(ISRFPA)'
                5,'PGM(ISRJB1) PARM(ISRJPA) NOCHECK'
                6,'PGM(ISRPTC)'
                7,'PGM(ISRYXDR) NOCHECK'
                8,'PANEL(ISRLPRIM)'
                C,'PGM(ISPTUTOR) PARM(ISR00005)'
                I,'CMD(%BLSCLIBD) NEWAPPL(BLSG) PASSLIB'
                T,'PGM(ISPTUTOR) PARM(ISR00000)'
              ' ',' '
                X,'EXIT'
                *,'?' )
  &ZTRAIL = .TRAIL
)END
```

Figure 1. An ISPF Panel for Starting the IPCS Dialog through BLSCLIBD

After you have modified the panel, the user simply selects the option to invoke the IPCS dialog.

Providing a TSO/E Logon Procedure and CLIST

When you use BLSCLIBD to invoke the IPCS dialog, IBM recommends that you provide a customized TSO/E logon procedure that, in turn, invokes a CLIST. The CLIST can then perform functions such as the following:

- Enter the TSO/E ALTLIB command to concatenate the SYS1.SBLSCLI0 library to the SYSPROC data set. SYS1.SBLSCLI0 contains IPCS CLISTs and REXX execs. IBM recommends the addition of SYS1.SBLSCLI0 to the SYSPROC concatenation.

  The BLSCLIBD CLIST uses the LIBDEF service to concatenate other IPCS libraries required for running the IPCS dialog. See "BLSCLIBD CLIST - Activate IPCS Dialog Services" on page 27 for further information.
- Enter the IPCS command (if you want to start IPCS in line mode for users).
- Enter the ISPF command (if you want to start ISPF for users).

Examples

The examples in Figure 2 on page 33 and Figure 3 on page 33 show a statement in a logon procedure and an associated CLIST for use with BLSCLIBD.

```
//*============ TSO/E LOGON PROCEDURE FOR USE WITH BLSCLIBD =========**
//PROC1    EXEC PGM=IKJEFT01,ROLL=(NO,NO),DYNAMNBR=100,
//              PARM='EXEC 'IPCSU1.IPCS.CLIST(START1)'
//SYSPROC  DD   DSN=COMPCTR.CLIST,DISP=SHR     Installation CLISTs
//         DD   DSN=ISR.V310.ISRCLIB,DISP=SHR  ISPF Version 3 CLISTs
//ISPMLIB  DD   DSN=COMPCTR.ISPMLIB,DISP=SHR   Installation messages
//         DD   DSN=ISR.V310.ISRMLIB,DISP=SHR  ISPF Version 3 messages
//ISPPLIB  DD   DSN=COMPCTR.ISPPLIB,DISP=SHR   Installation panels
//         DD   DSN=ISR.V310.ISRPLIB,DISP=SHR  ISPF Version 3 panels
//ISPSLIB  DD   DSN=COMPCTR.ISPSLIB,DISP=SHR   Installation skeletons
//         DD   DSN=ISR.V310.ISRSLIB,DISP=SHR  ISPF Version 3 skeletons
//ISPTLIB  DD   DSN=COMPCTR.ISPTLIB,DISP=SHR   Installation tables
//         DD   DSN=ISR.V310.ISRTLIB,DISP=SHR  ISPF Version 3 tables
//SYSHELP  DD   DSN=COMPCTR.HELP,DISP=SHR      Installation help text
//         DD   DSN=SYS1.HELP,DISP=SHR         IBM-supplied help text
```

Figure 2. TSO/E Logon Procedure for Use with BLSCLIBD

```
                    /* IPCSU1.IPCS.CLIST(START1) */

PROC 0
CONTROL LIST
ALTLIB ACTIVATE +
         APPLICATION(CLIST) +
         DA('SYS1.SBLSCLI0')            /* CLISTS AND REXX EXECS */
ISPF                                    /* START ISPF            */
```

Figure 3. CLIST for use with PROC1

Note: The allocation to SYSHELP is not required to start the IPCS dialog, but does provide IBM-supplied help text for users.

To make the procedure available to users, do the following:

1. Add the procedure to a JCL procedure library from which JES2 or JES3 retrieves TSO/E logon procedures.
2. Have the TSO/E administrator for the installation authorize individuals who require access to IPCS to use the logon procedure.

When the user logs on using PROC1, the logon procedure invokes the CLIST in Figure 3 on page 33. The CLIST then concatenates SYS1.SBLSCLI0 to SYSPROC and starts ISPF. Thus, by logging on with PROC1, the user is automatically placed in an ISPF session. From that point, a user who wants to start IPCS can simply select the IPCS option from the modified ISPF panel.

Methods for Starting IPCS

Note that the CLIST in Figure 3 on page 33 does not enter the IPCS command. If you want to make IPCS available to the user through the method that starts IPCS in line mode, you can add the IPCS command to the CLIST. To have the CLIST enter the IPCS command, add the line shown in Figure 4 on page 34 to the CLIST in Figure 3 on page 33, directly before the ISPF command.

```
IPCS                                  /* START IPCS LINE MODE  */
```

Figure 4. IPCS Command for IPCSU1.IPCS.CLIST(START1)

Customizing Access when Using the BLSG Dialog Program

This section shows how to customize access to IPCS when using the BLSG dialog program to invoke the IPCS dialog. There are two primary tasks involved in doing this customization, as described in the following sections:

- "Adding an ISPF Option to Invoke the BLSG Dialog Program" on page 34
- "Providing a TSO/E Logon Procedure and CLIST" on page 35

Adding an ISPF Option to Invoke the BLSG Dialog Program

Figure 5 on page 35 shows two lines added to the ISPF/PDF primary option menu, ISR@PRIM, to provide customized access to the IPCS primary option menu. The two lines are underlined in the figure to highlight their placement in the ISPF/PDF primary option menu. Note that the following selection is added to the part of the menu that ISPF displays on the user's screen:

```
%  I +IPCS        - IPCS problem analysis services
```

A corresponding line is added to the processing section of the menu. This line invokes the BLSG dialog program to start the IPCS dialog.

```
I,'PGM(BLSG) PARM(PANEL(BLSPPRIM)) NEWAPPL(BLSG) PASSLIB'
```

```
%-----------------------  ISPF/PDF PRIMARY OPTION MENU  ------------------------
%OPTION  ===]_ZCMD                                                             +
%                                                        +USERID    - &ZUSER
%  0 +ISPF PARMS  - Specify terminal and user parameters   +T IME     - &ZTIME
%  1 +BROWSE      - Display source data or output listings +T ERMINAL - &ZTERM
%  2 +EDIT        - Create or change source data           +P F KEYS  - &ZKEYS
%  3 +UTILITIES   - Perform utility functions
%  4 +FOREGROUND  - Invoke language processors in foreground
%  5 +BATCH       - Submit job for language processing
%  6 +COMMAND     - Enter TSO/E command or CLIST
%  7 +DIALOG TEST - Perform dialog testing
%  8 +
.LM UTILITIES- Perform library management utility functions
%  C +CHANGES     - Display summary of changes for this release
%  I +IPCS        - IPCS problem analysis services
%  T +TUTORIAL    - Display information about ISPF/PDF
%  X +EXIT        - Terminate ISPF using log and list defaults
%
+Enter%END+command to terminate ISPF.
%
)INIT
  .HELP = ISR00003
  &ZPRIM = YES        /* ALWAYS A PRIMARY OPTION MENU      */
  &ZHTOP = ISR00003   /* TUTORIAL TABLE OF CONTENTS        */
  &ZHINDEX = ISR91000 /* TUTORIAL INDEX - 1ST PAGE         */
  VPUT (ZHTOP,ZHINDEX) PROFILE
)PROC
  &ZSEL = TRANS( TRUNC (&ZCMD,'.')
                0,'PANEL(ISPOPTA)'
                1,'PGM(ISRBRO) PARM(ISRBRO01)'
                2,'PGM(ISREDIT) PARM(P,ISREDM01)'
                3,'PANEL(ISRUTIL)'
                4,'PANEL(ISRFPA)'
                5,'PGM(ISRJB1) PARM(ISRJPA) NOCHECK'
                6,'PGM(ISRPTC)'
                7,'PGM(ISRYXDR) NOCHECK'
                8,'PANEL(ISRLPRIM)'
                C,'PGM(ISPTUTOR) PARM(ISR00005)'
                I,'PGM(BLSG) PARM(PANEL(BLSPPRIM)) NEWAPPL(BLSG) PASSLIB'
                T,'PGM(ISPTUTOR) PARM(ISR00000)'
              ' ' ' '
                ,
                X,'EXIT'
                *,'?' )
  &ZTRAIL = .TRAIL
)END
```

Figure 5. An ISPF Panel for Starting the IPCS Dialog through BLSG

After you have modified the panel, the user simply selects the option to invoke the IPCS dialog. See *z/OS ISPF Planning and Customizing*.

Providing a TSO/E Logon Procedure and CLIST

When you use BLSG to invoke the IPCS dialog, IBM recommends that you provide a customized TSO/E logon procedure that:

- Concatenates the following IPCS libraries to ISPF data sets:

IPCS Library	ISPF Data Set Concatenation	Library Contents
SYS1.SBLSCLI0	SYSPROC	IPCS CLISTs and REXX execs
SYS1.SBLSMSG0	ISPMLIB	IPCS messages
SYS1.SBLSPNL0	ISPPLIB	IPCS panels
SYS1.SBLSKEL0	ISPSLIB	IPCS skeletons
SYS1.SBLSTBL0	ISPTLIB	IPCS tables

  The addition of SYS1.SBLSCLI0 to the SYSPROC concatenation is recommended. The library concatenations for the IPCS messages, panels, skeletons, and tables are required for the IPCS dialog.

- Invokes a CLIST, if you want to do either of the following:
  - Enter the IPCS command (to start IPCS in line mode for users)
  - Enter the ISPF command (to start ISPF for users).

Examples

Figure 6 on page 36 and Figure 7 on page 36 show examples of a logon procedure and CLIST for use with BLSG.

```
//*============ TSO/E LOGON PROCEDURE FOR USE WITH BLSG =============**
//PROC2    EXEC PGM=IKJEFT01,ROLL=(NO,NO),DYNAMNBR=100,
//              PARM='EXEC 'IPCSU1.IPCS.CLIST(START2)'
//SYSPROC  DD   DSN=COMPCTR.CLIST,DISP=SHR     Installation CLISTs
//         DD   DSN=ISR.V310.ISRCLIB,DISP=SHR  ISPF Version 3 CLISTs
//         DD   DSN=SYS1.SBLSCLI0,DISP=SHR     IPCS CLISTs
//ISPMLIB  DD   DSN=COMPCTR.ISPMLIB,DISP=SHR   Installation messages
//         DD   DSN=ISR.V310.ISRMLIB,DISP=SHR  ISPF Version 3 messages
//         DD   DSN=SYS1.SBLSMSG0,DISP=SHR     IPCS messages
//ISPPLIB  DD   DSN=COMPCTR.ISPPLIB,DISP=SHR   Installation panels
//         DD   DSN=ISR.V310.ISRPLIB,DISP=SHR  ISPF Version 3 panels
//         DD   DSN=SYS1.SBLSPNL0,DISP=SHR     IPCS panels
//ISPSLIB  DD   DSN=COMPCTR.ISPSLIB,DISP=SHR   Installation skeletons
//         DD   DSN=ISR.V310.ISRSLIB,DISP=SHR  ISPF Version 3 skeletons
//         DD   DSN=SYS1.SBLSKEL0,DISP=SHR     IPCS skeletons
//ISPTLIB  DD   DSN=COMPCTR.ISPTLIB,DISP=SHR   Installation tables
//         DD   DSN=ISR.V310.ISRTLIB,DISP=SHR  ISPF Version 3 tables
//         DD   DSN=SYS1.SBLSTBL0,DISP=SHR     IPCS tables
//SYSHELP  DD   DSN=COMPCTR.HELP,DISP=SHR      Installation help text
//         DD   DSN=SYS1.HELP,DISP=SHR         IBM-supplied help text
```

Figure 6. TSO/E Logon Procedure for Use with BLSG

Note: The allocation to SYSHELP is not required to start the IPCS dialog, but does provide IBM-supplied help text for users.

```
                    /* IPCSU1.IPCS.CLIST(START2) */

PROC 0
CONTROL LIST
ISPF                                  /* START ISPF           */
```

Figure 7. CLIST for use with PROC2

To make the logon procedure available to users, do the following:

1. Add the procedure to a JCL procedure library from which JES2 or JES3 retrieves TSO/E logon procedures.
2. Have the TSO/E administrator for the installation authorize individuals who require access to IPCS to use the logon procedure.

When the user logs on using PROC2, the logon procedure concatenates the IPCS libraries and invokes the CLIST in Figure 7 on page 36. The CLIST then starts ISPF. Thus, by logging on with PROC2, the user is automatically placed in an ISPF session. From that point, a user who wants to start IPCS can simply select the IPCS option from the modified ISPF panel.

Methods for Starting IPCS

Note that the CLIST in Figure 7 on page 36 does not enter the IPCS command. If you want to make IPCS available to the user through the method that starts IPCS in line mode, you can add the IPCS command to the CLIST. To have the CLIST enter the IPCS command, add the line shown in Figure 8 on page 36 to the CLIST in Figure 7 on page 36, directly before the ISPF command.

```
IPCS                                  /* START IPCS LINE MODE  */
```

Figure 8. IPCS Command for IPCSU1.IPCS.CLIST(START2)

Chapter 5. Customizing the IPCS Dialog

Installations can customize the IPCS full-screen dialog panels for a number of reasons. For example:

- An installation might want to customize one of the menu panels to integrate support for other IBM products besides MVS.
- An option provided on a low-level panel, such as the COMPONENT option of the IPCS MVS Analysis of Dump Contents panel, is used frequently by the installation and would be more accessible if placed in the IPCS Primary Option Menu.
- An installation may want to add options to invoke other IPCS subcommands.

See the following:

- Chapter 4, “Customizing Access to IPCS,” on page 25 and *z/OS MVS IPCS User's Guide* for information about accessing the IPCS dialog.
- *z/OS ISPF Dialog Tag Language Guide and Reference* for information about the IPCS dialog programs and the ISPF SELECT service.

Using the ISPF SELECT Service with IPCS Dialog Programs

If you intend to tailor the IPCS dialog to satisfy user requirements or if you intend to implement a separate dialog using IPCS services, use the dialog programs in this chapter.

Recursive Invocations of the ISPF and IPCS Dialogs

You can design your ISPF installation dialogs to use BLSG recursively. When BLSG is entered recursively, it yields its resource ownership responsibilities to the first copy of BLSG activated under the dialog task. It does not close dump data sets or end the IPCS dialog when it ends in a recursive context.

An example of how you might design your ISPF installation dialogs to use BLSG recursively follows:

1. Alter the ISPF/PDF Primary Option Menu as described in “Adding an ISPF Option to Invoke the BLSG Dialog Program” on page 34.
2. Alter SYS1.SBLSPNL0(BLSPPRIM) to contain the following line in the body:

```
%   I +ISPF         - ISPF/PDF PRIMARY OPTION MENU
```

3. Alter SYS1.SBLSPNL0(BLSPPRIM) to contain the following line in the)PROC section:

```
I,'PANEL(ISR@PRIM) NEWAPPL(ISR)'
```

This permits the user of these panels to do the following:

1. Activate the IPCS dialog from ISPF by entering:

```
I
```

2. Keep the IPCS dialog active and display the ISPF/PDF Primary Option Menu again by entering:

```
I
```

3. Activate the IPCS dialog recursively by entering:

```
I
```

Tailoring the IPCS Dialog to Identify the IPCS Level

Starting with z/OS Release 2, IPCS identifies the z/OS release for which the version of IPCS being used was intended. This information appears on the IPCS dialog primary option menu. Users who need to customize this panel can access the following ISPF variables to show the release of IPCS being used. The variables are defined in the ISPF shared pool for application BLSG by z/OS R2 and subsequent releases.

Table 9. ISPF Variables to Identify IPCS Release

ISPF Variable	Length	Content of variable
BLSYSFMI	8	Product FMID
BLSYSMOD	2	Product modification level
BLSYSNAM	16	Product name
BLSYSPID	8	PID number
BLSYSREL	2	Product release
BLSYSVER	2	Product version

Dumps of IPCS

If a dump is generated while z/OS R2 or later release of IPCS is active, load module BLSU will always be active. CSECT IEASYSID will always be present in the load module. IEASYSID contains all of the information shown in Table 9 on page 38. It can be used to confirm whether a mismatch between the level of IPCS and the materials being processed played a role in the problem leading to the dump.

BLSGDCDA Dialog Program - Display Component Data Analysis

Use dialog program BLSGDCDA to display the list of analysis exit routines in the BLSCECT parmlib member. You must invoke either the BLSGLIBD or the BLSG dialog program to begin an IPCS dialog before you invoke the BLSGDCDA dialog program.

Customizing Use of BLSGDCDA

Use ISPEXEC with the ISPF SELECT service to invoke BLSGDCDA directly, as follows:

```
ISPEXEC SELECT PGM(BLSGDCDA) [NEWAPPL(BLSG) PASSLIB]
```

Note:

1. Instead of using ISPEXEC to invoke the ISPF SELECT service, you can use other equivalent means to request the ISPF SELECT service.
2. BLSGDCDA does not expect any parameters when it is invoked.
3. The **NEWAPPL** and **PASSLIB** options may be omitted if BLSGDCDA is invoked when **NEWAPPL(BLSG)** is already active.
4. Specify PASSLIB only if both of the following conditions exist:
   a. You also specify NEWAPPL(BLSG).
   b. You are running ISPF Version 2 Release 2 or later.

BLSGDUIN Dialog Program - Display Dump Inventory

Use dialog program BLSGDUIN to display the list of dumps in the dump directory. You must invoke either the BLSGLIBD or the BLSG dialog program to begin an IPCS dialog before you invoke the BLSGDUIN dialog program.

Customizing Use of BLSGDUIN

Use ISPEXEC with the ISPF SELECT service to invoke BLSGDUIN directly, as follows:

```
ISPEXEC SELECT PGM(BLSGDUIN) [NEWAPPL(BLSG) PASSLIB] [PARM(source)]
```

Note:

1. Instead of using ISPEXEC to invoke the ISPF SELECT service, you can use other equivalent means to request the ISPF SELECT service.
2. BLSGDUIN does not expect any parameters when it is invoked.
3. The **NEWAPPL** and **PASSLIB** options may be omitted if BLSGDUIN is invoked when **NEWAPPL(BLSG)** is already active.
4. Specify PASSLIB only if both of the following conditions exist:
   a. You also specify NEWAPPL(BLSG).
   b. You are running ISPF Version 2 Release 2 or later.
5. Use the PARM parameter to place the specified source at the top of the screen initially displayed. The *source* is as follows; see the SETDEF subcommand in *z/OS MVS IPCS Commands* for coding the values.

   ```
   { ACTIVE | MAIN | STORAGE          }
   { DSNAME(dsname) | DATASET(dsname) }
   { FILE(ddname) | DDNAME(ddname)    }
   ```

   If the specified source is not in the dump directory, the source that would collate immediately before the specified source is shown at the top of the screen. If no source would collate before the specified source, the first source in the dump directory is shown at the top of the screen.

BLSGSCMD Dialog Program - Process an IPCS Subcommand or Command Procedure

Use dialog program BLSGSCMD to cause IPCS to process an IPCS subcommand or CLIST. You must invoke either the BLSGLIBD or the BLSG dialog program to begin an IPCS dialog before you invoke the BLSCSCMD dialog program.

Customizing Use of BLSGSCMD

Use ISPEXEC with the ISPF SELECT service to invoke BLSGSCMD directly, as follows:

```
ISPEXEC SELECT PGM(BLSGSCMD) PARM(text) [NEWAPPL(BLSG)] PASSLIB
```

Where *text* is the subcommand invocation string. ISPF limits *text* to no more than 100 characters. If your subcommand invocation string may be longer than that limit you may place the string in an ISPF SHARED pool variable and use the following invocation of BLSGSCMD:

```
ISPEXEC SELECT PGM(BLSGSCMD) PARM(GETVAR) [NEWAPPL (BLSG)] PASSLIB
```

or

```
ISPEXEC SELECT PGM(BLSGSCMD) PARM(GETVAR(var-name)) [NEWAPPL(BLSG)] PASSLIB
```

When GETVAR is specified alone, BLSGSCMD expects to find the subcommand invocation string in SHARED pool variable BLSSCMDP. Otherwise, you must supply the SHARED pool variable name as *var-name*.

Use dialog program BLSGSCMD to allow a function other than IPCS to use an IPCS subcommand or CLIST. A selection menu processed by the ISPF SELECT service is a common example of this situation.

You can use dialog program BLSGSCMD when you want any report written to the terminal by the IPCS subcommand or CLIST to be treated as a complete report. BLSGSCMD will discard any report written to the terminal before returning control to the invoker.

- Consider an example. Assume that the following ISPF SELECT service request invokes a CLIST.

  ```
  ISPEXEC SELECT CMD('%ISPFPROC')
  ```

  ISPF processes CLIST ISPFPROC. CLIST ISPFPROC can invoke BLSGSCMD multiple times. Each request is a request to write a report to the terminal. Each report will be processed separately, presented as a complete report to the user by IPCS, and discarded when the END primary command is entered. ISPFPROC generates each report separately. ISPFPROC can present selection or data entry panels to the user between each report to determine which report the user desires next.
- Contrast the previous example with another. Assume that the following ISPF SELECT service request invokes a CLIST:

  ```
  ISPEXEC SELECT PGM(BLSGSCMD) PARM('%IPCSPROC')
  ```

  IPCS processes CLIST IPCSPROC. CLIST IPCSPROC can invoke many IPCS subcommands, each of which write part of a single report to the terminal. IPCS can process many subcommands within IPCSPROC to satisfy a single **DOWN** primary command entered on the IPCS dump display reporter panel.

  CLIST IPCSPROC can complete a report in two ways:

  1. CLIST IPCSPROC can end. Dialog program BLSGSCMD will recognize the end of the report.
  2. CLIST IPCSPROC can invoke BLSGSCMD recursively, using a request such as

     ```
     ISPEXEC SELECT PGM(BLSGSCMD) PARM(IEFBR14)
     ```

Note:

1. Instead of using ISPEXEC to invoke the ISPF SELECT service, you can use other equivalent means to request the ISPF SELECT service.
2. The *text* passed as a parameter to dialog program BLSGSCMD is the subcommand or CLIST invocation to be processed.
3. Dialog program BLSGSCMD must run under ISPF application identifier BLSG. Specify the ISPF SELECT service option `NEWAPPL(BLSG)` if another application identifier might be active when the SELECT service is requested.
4. Specify PASSLIB only if both of the following conditions exist:
   a. You also specify NEWAPPL(BLSG).
   b. You are running ISPF Version 2 Release 2 or later.

Return Codes

Code	Explanation
00	Successful completion.
other	Return code from the ISPF SELECT service. See *z/OS ISPF Dialog Tag Language Guide and Reference* for more information.
16	Ending error with two possible meanings: • The program invoked BLSG in an inappropriate environment. • The ISPF SELECT service generated a return code of 16.

BLSGSETD Dialog Program - Check Defaults

Use dialog program BLSGSETD to check the values entered on the Defaults Option Data Entry panel. You must invoke either the BLSGLIBD or the BLSG dialog program to begin an IPCS dialog before you invoke the BLSGSEDT dialog program.

You are not required to customize BLSGSETD.

The BLSGSETD dialog program should be used instead of the BLSCSETD CLIST; BLSGSETD checks values more thoroughly than BLSCSETD does. However, an installation that has customized BLSCSETD may continue to use it.

A REXX exec, BLSXSETD, also provides the same function as the BLSGSETD dialog program and the BLSCSETD CLIST.

Customizing Use of BLSGSEDT

Use ISPEXEC with the ISPF SELECT service to invoke BLSGSETD directly, as follows:

```
ISPEXEC SELECT PGM(BLSGSETD) [NEWAPPL(BLSG) PASSLIB]
```

Note:

1. Instead of using ISPEXEC to invoke the ISPF SELECT service, you can use other equivalent means to request the ISPF SELECT service.
2. BLSGSETD does not expect any parameters when it is invoked.
3. The **NEWAPPL** and **PASSLIB** options may be omitted if BLSGSETD is invoked when **NEWAPPL(BLSG)** is already active.
4. Specify PASSLIB only if both of the following conditions exist:
   a. You also specify NEWAPPL(BLSG).
   b. You are running ISPF Version 2 Release 2 or later.

BLSLDISP Dialog Program - Browse an IPCS Dump Data Set

Use dialog program BLSLDISP to browse an IPCS dump data set. You must invoke either the BLSGLIBD or the BLSG dialog program to begin an IPCS dialog before you invoke the BLSLDISP dialog program.

- **Customizing Use of BLSLDISP**

  Use ISPEXEC with the ISPF SELECT service to invoke BLSLDISP directly, as follows:

  ```
  ISPEXEC SELECT PGM(BLSLDISP) PARM(text) [NEWAPPL(BLSL)] PASSLIB
  ```

  You can invoke IPCS dialog program BLSLDISP with a parameter that causes the immediate display of either the BROWSE option pointer panel or the BROWSE option storage panel. The syntax of the optional parameter, *text*, follows.

  Problem analysis dialogs that are designed to show storage during their operation can bypass the IPCS dialog BROWSE option entry panel.

  ```
  [PANEL( { POINTER } ) ]
  [       { STORAGE }   ]

  [ ACTIVE | MAIN | STORAGE          ]
  [ DSNAME(dsname) | DATASET(dsname) ]
  [ FILE(ddname) | DDNAME(name)      ]
  ```

PANEL(POINTER)
PANEL(STORAGE)
Specifies the panel that IPCS is to display on entry to IPCS BROWSE processing in place of the BROWSE option entry panel (BLSPOPT).

PANEL(POINTER)
Specifies that the BLSLDISP dialog program is to display the pointer panel. If a dump source is specified and the PANEL parameter is omitted, the pointer panel is displayed when BROWSE is requested on the IPCS Primary Option Menu.

PANEL(STORAGE)
Specifies that the storage panel is to be displayed by the BLSLDISP dialog program upon entry.

Symbol X for the dump is used to determine which storage is to be initially displayed. Symbol X is the name of the symbol which always contains the address of the last address referenced. If symbol X is not defined, location 0 in the default address space for the dump is initially displayed.

Enter the END primary command on the storage panel to display the pointer panel. This allows using the pointer panel in BROWSE processing initiated with the PANEL(STORAGE) option.

Enter the CANCEL primary command to exit BROWSE processing directly from the storage panel.

ACTIVE | MAIN | STORAGE
Specifies the main storage for the address space in which IPCS is currently running and allows you to access that active storage as the dump source. You can access private storage and any common storage accessible by an unauthorized program.

See the SETDEF subcommand in the *z/OS MVS IPCS Commands* for more information about the ACTIVE, MAIN, and STORAGE parameters.

DSNAME(*dsname*) | DATASET(*dsname*)
FILE(*ddname*) | DDNAME(*name*)
Specifies the dump source to be browsed.

If a dump source is specified and the PANEL parameter is omitted, PANEL(POINTER) is assumed.

If the parameter list passed to BLSLDISP is incorrect, then IPCS displays the IPCS BROWSE TERMINATED panel (see Figure 9 on page 42). This panel shows the parameter list and an error message to describe the problem.

```
-------------- IPCS BROWSE TERMINATED --------------- INVALID PANEL
 COMMAND ===] _

  IPCS BROWSE processing is being terminated at 10:15 on 10/25/86.

  The reason for termination is indicated in the message area of
  this panel.
  This may be related to the parameters passed to IPCS BROWSE which
  are shown below:

  PARAMETERS ===] PANEL(POINTED) ACTIVE

  Press the ENTER key to terminate IPCS BROWSE processing.
```

Figure 9. Example - IPCS Browse Terminated Panel

Note:

1. Instead of using ISPEXEC to invoke the ISPF SELECT service, you can use other equivalent means to request the ISPF SELECT service.
2. The *text* passed as a parameter to dialog program BLSLDISP is explained in the Customizing Use of BLSLDISP section.
3. Dialog program BLSLDISP must run under ISPF application identifier BLSL. Specify the ISPF SELECT service option **NEWAPPL(BLSL)** when requesting the SELECT service.

4. Specify PASSLIB only if both of the following conditions exist:
   a. You also specify NEWAPPL(BLSL).
   b. You are running ISPF Version 2 Release 2 or later.

Chapter 6. Using IPCS on Another System

The IPCS of z/OS must be used to format dump and trace data sets from z/OS systems.

Similarly:

- The IPCS of MVS/ESA SP 5 must be used to format dump and trace data sets from MVS/ESA SP 5.
- The IPCS of MVS/ESA SP 4 must be used to format dump and trace data sets from MVS/ESA SP 4.
- The IPCS of MVS/SP Version 3 must be used to format dump and trace data sets from MVS/SP Version 3.
- The IPCS of MVS/XA must be used to format MVS/XA dump data sets.
- The IPCS of MVS/370 must be used to format MVS/370 dump data sets.

During transition from one system to another, the new IPCS may run under your previous system. Situations such as the following require consideration:

- You were running a sysplex with two or more releases on various systems, generating traces on each. A problem occurred, and now you want to merge the traces. To accomodate you in this situation, IPCS, the central tracing components, and most components of z/OS permit the use of IPCS in the most recent release to process **trace data sets**. You may need to use COPYTRC to extract traces from **dumps** if that was the mechanism used to capture some of the traces, and you should confirm that the tracing support supplied by the components that wrote the trace entries supports this use.
- You are a support programmer, and you are presented with a dump. The presenter cannot pinpoint the release that generated the dump. To accomodate you in this situation, IPCS from the most recent release will attempt dump initialization and basic processing sufficient to identify the release that generated the dump. Do **not** attempt to do any analysis beyond identifying the generating release, and do be prepared for some messages that result from data area changes between releases. Use DROPDUMP to remove all analysis records from the dump directory before performing analysis using IPCS from the correct release.

The procedure in Figure 10 on page 46 allows IPCS to process dumps produced by system SY1 on any MVS system that can access both the dumps produced by SY1 and the data sets that support SY1's IPCS, all of which are assumed to reside on 3380 SY1PAK.

Rather than using the system catalog on SY2 to locate IPCS data sets, a process that would cause SY2's IPCS data sets to be used, the LOGON procedure specifies the unit and volume serial number of the volume on which SY1's IPCS data sets have been placed.

```
//********* TSO/E LOGON PROCEDURE FOR USING SYS1.MIGLIB **************
//IPCSPROC EXEC PGM=IKJEFT01,DYNAMNBR=70,REGION=3072K
//STEPLIB  DD   DSN=SYS1.MIGLIB, DISP=SHR,     Steplib for SYS1.MIGLIB
//          UNIT=3380,VOL=SER=SY1PAK           defining alternate system
//SYSPROC  DD   DSN=COMPCTR.CLIST,DISP=SHR     Installation CLISTs
//         DD   DSN=ISR.V210.ISRCLIB,DISP=SHR  ISPF Version 2 CLISTs
//         DD   DSN=SYS1.SBLSCLI0,DISP=SHR,    IPCS CLISTs
//          UNIT=3380,VOL=SER=SY1PAK            define alternate system
//ISPMLIB  DD   DSN=COMPCTR.ISPMLIB,DISP=SHR   Installation messages
//         DD   DSN=ISR.V210.ISRMLIB,DISP=SHR  ISPF Version 2 messages
//         DD   DSN=SYS1.SBLSMSG0,DISP=SHR,    IPCS messages
//          UNIT=3380,VOL=SER=SY1PAK            define alternate system
//ISPPLIB  DD   DSN=COMPCTR.ISPPLIB,DISP=SHR   Installation panels
//         DD   DSN=ISR.V210.ISRPLIB,DISP=SHR  ISPF Version 2 panels
//         DD   DSN=SYS1.SBLSPNL0,DISP=SHR,    IPCS panels
//          UNIT=3380,VOL=SER=SY1PAK            define alternate system
//ISPSLIB  DD   DSN=COMPCTR.ISPSLIB,DISP=SHR   Installation skeletons
//         DD   DSN=ISR.V210.ISRSLIB,DISP=SHR  ISPF Version 2 skeletons
//         DD   DSN=SYS1.SBLSKEL0,DISP=SHR,    IPCS skeletons
//          UNIT=3380,VOL=SER=SY1PAK            define alternate system
//ISPTLIB  DD   DSN=COMPCTR.ISPTLIB,DISP=SHR   Installation tables
//         DD   DSN=ISR.V210.ISRTLIB,DISP=SHR  ISPF Version 2 tables
//         DD   DSN=SYS1.SBLSTBL0,DISP=SHR,    IPCS tables
//          UNIT=3380,VOL=SER=SY1PAK            define alternate system
//SYSHELP  DD   DSN=COMPCTR.HELP,DISP=SHR      Installation help text
//         DD   DSN=SYS1.HELP,DISP=SHR,        IBM-supplied help text
//          UNIT=3380,VOL=SER=SY1PAK            define alternate system
//IPCSPARM DD   DSN=SYS1.PARMLIB,DISP=SHR,     Parmlib to use when
//          UNIT=3380,VOL=SER=SY1PAK            processing dumps
```

Figure 10. A TSO/E LOGON Procedure for Using SYS1.MIGLIB

The data sets for SY1 (or copies of them) that contain IPCS code and related materials are the following:

- SYS1.HELP - TSO/E HELP command data
- SYS1.MIGLIB - Code
- SYS1.PARMLIB - Parmlib data
- SYS1.SBLSCLI0 - CLISTs
- SYS1.SBLSKEL0 - ISPF skeletons
- SYS1.SBLSMSG0 - ISPF messages
- SYS1.SBLSPNL0 - ISPF panels
- SYS1.SBLSTBL0 - ISPF tables

Make sure that you place all installation-provided IPCS exit routines in SYS1.MIGLIB.

See Chapter 9, “Installing IPCS Exit Routines,” on page 93 for information about installing IPCS exit routines in SYS1.MIGLIB.

Chapter 7. Providing Security for IPCS

Provide security for IPCS by using:

- The z/OS SecureWay Security Server, which includes the Resource Access Control Facility (RACF®)
- BLSUGWDM validity check module to disable access to TSO/E commands

Providing z/OS Security Server Protection

Security of IPCS-formatted dumps is handled in the same way as in the MVS system—through Security Server RACF. You can use the Security Server to provide additional security, as required.

Note: IBM does not recommend use of data set passwords, or the use of TSO security that is separate from RACF.

Your installation must continue to take specific action to prevent unauthorized access to dumps containing sensitive data.

Using BLSUGWDM to Disable Access to TSO/E Commands

For security, create a validity check module named BLSUGWDM. Place in this module your command name validation routine. Use the routine to disable access to TSO/E commands. Providing the validation routine prevents unauthorized users from accessing certain programs. IPCS calls BLSUGWDM during processing for the IPCS TSO subcommand and all TSO/E commands invoked under IPCS. IPCS passes BLSUGWDM the command processor parameter list (CPPL) and the command scan output area (CSOA), as filled in by the TSO/E IKJSCAN routine.

BLSUGWDM can change CSOA bits to indicate whether either:

- A TSO/E command is incorrect (bit CSOABAD is set on)
- A TSO/E command is valid only as a CLIST (bit CSOAEXEC is set on)

If the CSOAEXEC bit is on upon entry to BLSUGWDM, the routine should not turn it off.

See *z/OS TSO/E System Diagnosis: Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for the data areas.

Note: IBM supplies a sample BLSUGWDM installation exit with the name of BLSXGWDM in the SYS1.SAMPLIB parmlib member. The sample will help you understand the capabilities and limitations of this interface.

Chapter 8. Writing IPCS Exit Routines

You can customize and enhance IPCS by providing exit routines.

Exit Routines

The IPCS exit routines are:

Exit Routine Description	Use the Routine When You Want to:
"ANALYZE Exit Routine" on page 57	Generate data for contention analysis.
"Address Space Control Block (ASCB) Exit Routine" on page 59	Generate information related to the address space or ASCB being processed.
"Control Block Formatter Exit Routine" on page 71	Assist in formatting a control block.
"Control Block Status (CBSTAT) Exit Routine" on page 73	Perform analysis and generate condensed output describing information relevant to the debugging process.
"CTRACE Buffer Find Exit Routine" on page 67	Locate the component trace buffers in a dump for a particular component.
"CTRACE Filter/Analysis (CTRF) Exit Routine" on page 69	• Perform statistical analysis of the component trace. • Provide additional component trace filtering. • Limit the number of component trace entries (CTE) processed.
"Find Exit Routine" on page 76	Associate a symbol with an AREA or STRUCTURE in a dump.
"GTFTRACE Filter/Analysis Exit Routine" on page 78	• Do statistical analysis of GTF trace records. • Provide additional GTF trace record filtering. • Limit the number of GTF trace records processed.
"GTFTRACE Formatting Appendage" on page 79	Obtain formatted output of GTF trace records containing a particular FID and an EID in the user range.
"Model Processor Formatting (MPF) Exit Routine" on page 81	Dynamically interact with the formatting service to augment its function.
"Obtaining Information About Loaded Modules" on page 150	Retrieve information associated with a loaded module from a dump.
"Post-Formatting Exit Routine" on page 83	Supply a routine for any type of structure that can be described by a parmlib data statement.
"Scan Exit Routine" on page 85	Check the validity of an AREA or STRUCTURE in a dump.
"Task Control Block (TCB) Exit Routine" on page 88	Generate information related to the task control block (TCB) being processed.
"Verb Exit Routine" on page 90	Print a title for each major report and generate table of contents entries. Exploit exit services to perform a variety of installation or user functions.

General Information about Writing an IPCS Exit Routine

You can write an IPCS exit routine for two types of formatting:

- **IPCS formatting:** This type of formatting is for unformatted dumps and traces processed by IPCS:
  - SVC dumps
  - Stand-alone dumps
  - ABEND dumps written to SYSMDUMP data sets
  - CTRACE trace data
  - GTFTRACE trace data.
- **ABEND/SNAP formatting:** This type of formatting is used for formatted dumps. These dumps are:
  - ABEND dumps written to SYSABEND data sets
  - ABEND dumps written to SYSUDUMP data sets
  - SNAP dumps.

The following topics describe these items that you should be aware of when writing an exit routine:

- Conditions on entry to an IPCS exit routine
- Services available to an IPCS exit routine
- Restrictions and limitations of an IPCS exit routine
- Discontinuing processing for an interactive user
- Communication between IPCS exit routines
- External routines invoked by IPCS exit routines
- IPCS data areas and mapping macros to be used by IPCS exit routines
- Conventions for return to the caller of an IPCS exit routine
- Making load libraries available to IPCS
- Using IPCS exit routines to format installation-supplied application trace data

Programming considerations that differ for IPCS formatting and ABEND/SNAP formatting are described.

Conditions on Entry to an IPCS Exit Routine

- **For IPCS and ABEND/SNAP Formatting**

  The exit routine receives control for either type of formatting with:

  - A program status word (PSW) in primary mode and enabled, pageable, in task mode
  - The addressing and residence modes specified when the exit was link-edited

  The exit routine should obtain working storage from the subpool specified in field ADPLSBPL, by coding:

  ```
  GETMAIN ... SP=adplsbpl_value
  ```

  adplsbpl_value is the value in field ADPLSBPL, contained in BLSABDPL.

  The exit routine must release all resources it acquired before the routine completes processing.

  For both ABEND/SNAP and IPCS formatting, the system provides recovery for an exit routine. Use the ESTAE and ESTAEX macros to provide additional recovery if desired, but do not have recovery routines request retry. To ensure appropriate resource clean-up, permit percolation to the recovery routines supplied by the system.

- **For IPCS Formatting**

  An exit routine runs in PSW key 8 and problem state under a task control block (TCB) established for IPCS or ISPF logical screen processing.

- **For ABEND/SNAP Formatting**

  An exit routine runs in PSW key 0 and supervisor state under any TCB for which SNAP (direct request) or ABEND (indirect request) processing has been requested. The routine should be carefully audited to ensure that it introduces neither security nor integrity exposures into the system.

  No locks are held when the exit routine receives control. The exit routine cannot issue the ENQ macro or obtain any locks.

Services Available to an IPCS Exit Routine

- **For IPCS Formatting**

  An exit routine can use the following services:

  – **System Services:**

    - Any system service available to programs that are enabled and in task mode. Use such service carefully so that the exit routine exits in the same state in which it was entered.
    - Any system service available to unauthorized programs.

  – **IPCS Exit Services:**

    - Add symptom
    - Control block formatter
    - Control block status
    - Exit control table (ECT)
    - Equate symbol
    - Expanded print
    - Format model processor
    - Get symbol
    - Name
    - Name/token lookup
    - Select ASID
    - Standard print
    - Storage access
    - Storage map
    - Symbol
    - Table of contents
    - WHERE

  – **TSO/E Services:** IPCS makes a command processor parameter list, CPPL (mapped by mapping macro IKJCPPL), available to all exit routines to facilitate the use of TSO/E services. IPCS initializes all fields in the CPPL except for CPPLCBUF upon entry to all exit routines. IPCS initializes the CPPLCBUF field only upon entry to verb exit routines.

    Note:

    1. The use of IKJPARS and IKJSCAN services of TSO/E is recommended for analyzing the syntax of free-form user input text passed to verb exit routines.
    2. The STACK service and the TSO/E EXEC command can be used to queue subcommands and CLISTs, respectively, to be processed by IPCS immediately after completion of the current subcommand.

  – **Other Services:**

    - ISPF services. Use the ISPF SELECT service to invoke the IPCS dialog programs.
    - PDF services, if installed.

- DATABASE 2 (DB2®) services.
- Graphical data display manager (GDDM) services.
- Other application services.

- **For ABEND/SNAP Formatting**

  An exit routine can use the following services:

  – **System Services:**
    - Any system service available to programs that are enabled and in task mode. Use such service carefully so that the exit routine exits in the same state in which it was entered.
    - Any system service available to authorized programs.

  – **IPCS Exit Services:**
    - Control block formatter
    - Expanded print
    - Format model processor
    - Standard print
    - Storage access

Restrictions and Limitations of an IPCS Exit Routine

- **For IPCS and ABEND/SNAP Formatting**

  An exit routine can use system services that cause TCBs and service request blocks (SRB) other than the original TCB to perform processing. Such TCBs and SRBs cannot request dump-processing services.

  Entry to exit routines must be in primary mode, and all services invoked by the exit routine expect to receive control in primary mode. In between, however, an exit routine may switch to AR mode for any processing that might require AR mode.

- **For IPCS Formatting**

  Exit routines should be reentrant and recursively enterable. Existing verb exit routines that do not satisfy this criterion can be used with the restriction that the IPCS dialog user cannot call for their recursive use. In this case, the verb exit routines should be marked with neither the RENT nor the REUS linkage editor attributes so that the user of the IPCS dialog can cause multiple, independent copies of the exit routine to be loaded and run under each ISPF logical screen task.

  When using any of the IPCS print services, follow these guidelines to design messages issued by IPCS:

  1. Do not use EBCDIC codes that cannot be represented on all the media to which IPCS messages may be transmitted. Lowercase EBCDIC letters may be treated as an exception to this rule because IPCS will translate lowercase letters to uppercase if the user requests.
  2. IPCS limits the EBCDIC codes that you may use by translating all unprintable codes to EBCDIC periods. IBM does not recommend that you use richer fonts than the CHARS=DUMP font, but IPCS does not prevent you from using these fonts.

     See *z/OS MVS IPCS User's Guide* for information about how IPCS treats different print fonts.
  3. Do not design reports in which lines of text refer to other parts of the report through page numbers. For reports that are quite extensive, consider using the table of contents service to help the report user find the parts of the report that are of interest. The table of contents service allows the report user to:

     – Supply a data set specifically for the table of contents information

     – Direct the IPCS print output to another data set where IPCS can include page headers that include the referenced page numbers.

     If the report user decides not to use the table of contents entries, IPCS discards them.

When a report is written to a terminal instead of a print data set, no page headers (and, therefore, no page numbers) are displayed. Thus, references to page numbers are of limited value in a report written to a terminal; the page number references merely provide some indication of the relative locations of various items in a report.

- **For ABEND/SNAP Formatting**

  An exit routine should not alter the environment being dumped. Resources that are allocated for processing by an the routine should be returned to their original state before the routine ends processing.

Discontinuing Processing for an Interactive User

An exit routine should be prepared to handle the request of an interactive user tp terminate report processing. IPCS handles requests for transaction termination by turning on bit ADPLSYNO as soon as the request is recognized. IPCS then denies subsequent requests for services until the end of the transaction.

For exits that retrieve a small amount of data, format it immediately, and then iterate the process, IPCS responds quickly to a request to terminate the transaction. However, for exits that perform a significant amount of data analysis before actually providing any output, some additional provision should be made to the exit processing. Such exits should include tests of ADPLSYNO in their processing when IPCS services are denied. If ADPLSYNO is on in these circumstances, the exit should terminate as quickly as possible.

Communication Between IPCS Exit Routines

- **For IPCS and ABEND/SNAP Formatting**

  One exit routine can communicate with another by invoking the other exit routine and supplying parameters. No mechanism is supplied to remember data between invocations or to share information between separate exit routines when an exit routine is invoked multiple times.

- **For IPCS Formatting**

  To avoid repetitive processing, IPCS provides the following functions through the dump directory:

  – Definitions of symbols in the IPCS symbol table save repetitive look-up operations for such blocks as the communications vector table (CVT). Centralizing the look-up process for these blocks helps independently-written functions come to the same conclusions regarding whether a usable instance of a block can be found.

    Use the equate symbol service, get symbol service, and symbol service to exploit this communication mechanism.

    Defining types of data and naming conventions for IPCS symbols and supplying find routines to locate instances of that data by name will extend the communications mechanism.

  – Entries in the IPCS storage map are designed to save repetitive validation of data areas, such as the CVT. IPCS storage map entries have other uses as well.

    Use the get symbol service, storage map service, and symbol service to exploit the IPCS storage map.

    Extend the communications mechanism by defining types of data and supplying scan exit routines to determine whether instances of the data appear to be usable.

  For information about the exit services you can use for communication between exit routines, see the following:

  – “Equate Symbol Service” on page 111
  – “Get Symbol Service” on page 123
  – “Storage Map Service” on page 137
  – “Symbol Service” on page 140

External Routines Invoked by IPCS Exit Routines

- **For IPCS Formatting**

  An exit routine can invoke external routines freely. The system provides the external routine IEAVTFRD.

- **For ABEND/SNAP Formatting**

  An exit routine should use only external routines accessible in the link pack area (LPA) for performance reasons.

IPCS Data Areas, Macros, and Mapping Macros to be Used by IPCS Exit Routines

- **For IPCS and ABEND/SNAP Formatting**

  When writing an exit routine, do not define any persistent data areas. Any storage that the routine obtains with the GETMAIN macro should be released with a FREEMAIN macro before the routine completes processing.

  IPCS provides mapping macros to map data areas needed to write IPCS exit routines and access IPCS exit services. IPCS also provides macros used to write certain IPCS exit routines.

  See the following:

  – "IPCS Macros and Mapping Macros" on page 5 for information about the IPCS macros and mapping macros needed to write IPCS exit routines
  – *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for information coding the macros
  – For macro mappings, see *z/OS MVS Data Areas* and *z/OS TSO/E System Diagnosis: Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

- **For ABEND/SNAP Formatting**

  Data areas may be defined by coding the DATA statement in the BLSCUSER parmlib member. To format BLSCUSER-defined data areas, code a SNAP exit routine.

  See *z/OS MVS Installation Exits* for information about writing SNAP exit routines.

Conventions for Return to Caller for an IPCS Exit Routine

- **For IPCS and ABEND/SNAP Formatting**

  All exit routines must return control to their calling routine using standard linkage. Register 15 contains a return code that is used by some of the IPCS functions that call exit routines. Return code conventions differ from one type of exit routine to another and are discussed as part of the "Output" topic for each type of exit routine.

Making Load Libraries Available to IPCS

- **For IPCS Formatting**

  You can make load libraries available to IPCS through any of the following methods:

  – Bring a load module into the job pack area through the use of the ATTACH, LINK, LOAD, or XCTL macros.
  – Define your own library, which contains exit routines, and identify it to IPCS through one of the following methods:
    - Specify the TASKLIB parameter as part of the TSO/E IPCS command used to start IPCS.
    - Make DD statement ISPLLIB available at the time the ISPF session is started.
    - Use the TSOLIB command of TSO/E. See detailed information in *z/OS TSO/E Command Reference*.
  – Use JOBLIB or STEPLIB DD statements in the JCL of a LOGON procedure.
  – Place a load module into the link pack area (LPA).

– Use SYS1.LINKLIB, LNKLSTxx data sets, or SYS1.MIGLIB.

For all but the first method, the linkage editor places load modules into the library.

- **For ABEND/SNAP Formatting**

  The load module must be available in the link pack area (LPA) or in SYS1.LINKLIB, LNKLSTxx data sets, or SYS1.MIGLIB. Use of the LPA is preferred because you do not need to use the private area.

Managing Storage for IPCS Exit Routines

z/OS Release 2 provides macros that can be used to manage blocks of storage in IPCS and SNAP/ABDUMP environments. Table 10 on page 55 summarizes the macros for use in allocating and releasing storage. Following the table is a description of the meaning of each column.

Table 10. Storage Management Macros

Allocate macro	Free macro	Storage location	Register saving	Host environment	ABDPL address
BLSQAANS	BLSQFANS	ANY	STM	SNAP 0r IPCS	Parm 1
BLSQALLR	BLSQFRER	BELOW	STM	SNAP 0r IPCS	Reg 1
BLSQALLS	BLSQFRES	BELOW	STM	SNAP 0r IPCS	Parm 1
BLSUA24R	BLSUF24R	BELOW	BAKR	IPCS	Reg 1
BLSUA24S	BLSUF24S	BELOW	BAKR	IPCS	Parm 1
BLSUA31R	BLSUF31R	ANY	BAKR	IPCS	Reg 1
BLSUA31S	BLSUF31S	ANY	BAKR	IPCS	Parm 1
BLSUAANR	BLSUFANR	ANY	STM	IPCS	Reg 1
BLSUAANS	BLSUFANS	ANY	STM	IPCS	Parm 1
BLSUALGR	BLSUFRGR	ANY	STMG	IPCS	Reg 1
BLSUALGS	BLSUFRGS	ANY	STMG	IPCS	Parm 1
BLSUALL2	BLSUFRE2	BELOW	STM	IPCS	Parm 2
BLSUALLR	BLSUFRER	BELOW	STM	IPCS	Reg 1
BLSUALLS	BLSUFRES	BELOW	STM	IPCS	Parm 1

Storage Location

‘BELOW’ storage indicates that the macro allocates storage below the 16 MB line. Modules executing in either 24-bit or 31-bit addressing mode can request BELOW storage.

‘ANY’ storage indicates that the macro may allocate storage above or below the line. Only modules executing in 31-bit addressing mode can request ANY storage.

The storage allocation macros execute in the addressing mode of the caller and initialize the first word of the storage allocated to conform with IBM standards:

- Bits 0-7 of word 0 are set to zero to indicate that an assembler language program owns the storage rather than a program generated by a HLL compiler.
- Bits 8-31 of word 0 are set to indicate the length of the storage.
- Modules that are designed to execute on zSeries hardware and are passed save areas long enough to employ F4SA or F7SA linkage may use STMG to efficiently save general purpose registers. See Starting in AMODE 64 in the *z/OS MVS Programming: Assembler Services Guide* for more information regarding such linkages.

Register Saving

The IPCS automatic storage allocation macros require the saving of all registers.

- Traditional module linkage involves the saving of registers passed in a 72-byte save area via STM. Modules using this means to save registers should select macros from rows in Table 10 on page 55 where the “Register saving” column says “STM”.
- A newer, hardware-assisted linkage convention employs the BAKR instruction to save registers. Modules using this means to save registers should select macros from rows where the “Register saving” column says “BAKR”.

Host Environment

The following recommendations apply:

- Modules that must execute as part of SNAP/ABDUMP processing should select macros where the “Host environment” column says “SNAP or IPCS”. Because SNAP/ABDUMP processing may need to be performed in environments where virtual storage is constrained, a very small amount of automatic storage data is recommended. It is also recommended that the location be ANY, if possible.
- Modules that only need to execute as part of an IPCS session should select macros from a row where the “Host environment” column says “IPCS”. The path length for these macros is slightly shorter than equivalent “SNAP or IPCS” macros, and the modules that acquire system storage to support this purpose ask for larger increments, reducing the frequency with which system services need to be explicitly requested.

ABDPL Address

The BLSABDPL mapping macro is the IPCS exit routine parameter list. There are three ways in which the ABDPL address can be passed to the storage management macros:

1. Routines that require no formal parameter may accept control in primary mode with the ABDPL address loaded into general purpose register 1. Such modules must use macros from rows where the “ABDPL address” column reads “Reg 1”.
2. Routines that require formal parameters should be designed to accept control in primary mode and receive the ABDPL address as the first formal parameter. Modules that do so should use macros from rows where the “ABDPL address” column reads “Parm 1”.
3. IKJPARS validity-check routines, many of which have been written for use by IPCS, receive control in primary mode with IKJPARS forwarding an address supplied to it as a second formal parameter. IKJPARS validity-check routines written for use by modules that supply IKJPARS with the ABDPL address should use macros from rows where the “ABDPL address” column reads “Parm 2”.

Programming Considerations

Register Usage

The allocation macros and the modules that they invoke presume the right to alter a large number of general purpose registers. Assembler language programs using the macros must exercise caution regarding registers loaded prior to using the macros and expected to remain unaltered by them.

The storage allocation macros intended for use in both SNAP and IPCS host environments load registers 1, 9, and 10 with the addresses of the storage assigned for automatic storage purposes, the ABDPL, and internal use. Registers 0, 2-5, 11, 14, and 15 may be altered unpredictably. Register 13 is expected to contain the address of the 72-byte save area supplied by the caller of the module. The storage allocation macros intended for use in only the IPCS host environment load registers 1 and 9 with the addresses of the storage assigned for automatic storage purposes and of the ABDPL respectively. Registers 0, 2-7, 11, 14, and 15 may be altered unpredictably. Register 13 is expected to contain the address of the 72-byte save area supplied by the caller of the module.

Error Return

If storage cannot be allocated, all IPCS automatic storage allocation return to the caller of the module requesting automatic storage with return code 16.

ANALYZE Exit Routine

An ANALYZE exit routine is a way for IPCS to detect resource contention that existed in the system at the time the system wrote the dump. The ANALYZE exit routine uses the CQE create service to describe the contention to IPCS so that IPCS can analyze contention on a system-wide basis and generate appropriate reports.

Define the ANALYZE exit routine in the BLSCUSER parmlib member with the following statement:

```
EXIT EP(name) ANALYZE
```

name is the name of the ANALYZE exit routine.

See the following:

- "Contention Queue Element (CQE) Create Service" on page 108 for information about using the CQE create service to describe contention information
- *z/OS MVS IPCS Commands* for examples of the reports produced by the ANALYZE subcommand
- *z/OS MVS Initialization and Tuning Reference* for information about the EXIT statement in a BLSCUSER parmlib member

Possible Uses

ANALYZE exit routines can enhance the analysis provided by IBM-supplied ANALYZE exit routines by processing either:

- Installation resources
- IBM component or subsystem resources that do not supply ANALYZE exit routines

ANALYZE exit routines increase the diagnostic value of the ANALYZE subcommand reports.

Programming Considerations

Be aware of the following information when writing an ANALYZE exit routine.

Performance Implications

When the IPCS user enters the ANALYZE subcommand against a dump, IPCS gives each ANALYZE exit routine control once. An ANALYZE exit routine uses the CQE create service to make contention information available. IPCS records this contention information in the dump directory; this information can be reused as often as needed.

Restrictions and Limitations

None.

Data Areas

- **ADPLEXTN:** (Mapped by mapping macro BLSABDPL) IPCS sets field ADPLEFCD to ADPLEFAN (X'0001') upon entry to an ANALYZE exit routine to permit a common entry point to serve as an ANALYZE exit routine, as well as one of the following types of exit routines:
  - ASCB exit routine
  - Post-formatting exit routine
  - TCB exit routine

– Verb exit routine

The registers and other conditions on entry to each of these types of exit routines are the same. Use field ADPLEXTN to determine the purpose of a given call to a multi-function entry point.

See the following:

- *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for information about the BLSABDPL macro
- *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a mapping of BLSABDPL

Passing Control

An ANALYZE exit routine receives control from IPCS when the IPCS user enters the ANALYZE subcommand against a dump. IPCS invokes the ANALYZE exit routines in the order they are listed in the installation-supplied BLSCUSER parmlib member.

See Chapter 9, “Installing IPCS Exit Routines,” on page 93 for information about installing exit routines by listing them in the installation-supplied BLSCUSER parmlib member.

Input

In addition to the input considerations described in “Conditions on Entry to an IPCS Exit Routine” on page 50, the following special input considerations apply to ANALYZE exit routines.

On entry to an ANALYZE exit routine, register 1 points directly to the IPCS exit parameter list (ABDPL data area).

Field ADPLEFCD of ABDPL is an exit function code. IPCS sets ADPLEFCD to ADPLEFAN (X'0001'). The ADPLEFCD field allows a user to write an exit routine that acts as both a verb exit routine, which formats data, and an ANALYZE exit routine, which simply reports on contention. The exit routine can follow a data area chain and perform formatting or call the CQE create service depending on the value in ADPLEFCD.

Output

In addition to the output considerations described in “Conventions for Return to Caller for an IPCS Exit Routine” on page 54, the following special output considerations apply to ANALYZE exit routines:

Registers at Exit

Register
Contents

0 through 14
Same as on entry

15
Ignored

Example

The following notes describe how a user might go about writing an ANALYZE exit routine for a mythical resource called WIDGET:

1. Choose a name for the WIDGET ANALYZE exit routine, such as ANLZWIDG, and install it.

   See Chapter 9, “Installing IPCS Exit Routines,” on page 93 for information about installing IPCS exit routines.
2. In module ANLZWIDG, establish addressability to the ABDPL data area. The simplest way to establish addressability to ABDPL is to use the following instruction:

```
USING ABDPL,R1
```

This takes advantage of the fact that register 1 directly addresses ABDPL upon entry. A more practical example would be:

```
LR     ABDPLPTR,R1
USING  ABDPL,ABDPLPTR
```

Since register 1 is used to invoke services, most ANALYZE exits will need to copy register 1 to another register before using it.

3. Use the storage access service to locate the major WIDGET data area (WIDGMAJ) that contains the anchor to a chain of WIDGET control elements (WIDGCE). Each WIDGCE points to a list of WIDGET data areas (WIDGCB) that indicate the owner or waiter for a particular WIDGET. Examine the WIDGMAJ to make sure the data area is valid and has not be overlaid with bad data. If the data area is bad, issue an error message and stop processing.

   See "Storage Access Service" on page 135 for information about using the storage access service.

4. Code a loop that accesses each WIDGCE in the chain. Possibly add checks or controls to prevent infinite looping if the chain has become broken or circular. The following steps apply for each WIDGCE:

   a. If there are no units of work waiting for this particular WIDGET, go on to the next WIDGCE.

      Note: If the act of owning a WIDGET is a significant piece of debugging information, it might be desirable to report the owner of the WIDGET using the CQE create service. This information could then be viewed in the ASID or RESOURCE report generated on the ANALYZE subcommand.

   b. If the WIDGET has both units of work owning it and waiting for it, then invoke the CQE create service for each of these units of work. The type of information needed is:

      Resource name
      Start all resource names for a particular class of resources with the same characters (for example ENQs start with **MAJOR=**, locks start with **Local lock ...**). This will cause the ANALYZE RESOURCE report to group similar resources together. If WIDGETs were identified by their position on the queue, the resource name might be **WIDGET 00001**.

      Additional Data
      Additional data is usually unique to each resource type. For ENQs, the GRS ANALYZE exit routine indicates if the ENQ is a result of a RESERVE macro. For local locks, the additional data indicates that the owner is actively running on a certain CPU. For WIDGETs, there might be some special characteristics that are useful to know. This information is placed in the additional data and is formatted in the ANALYZE reports.

      Unit of Work
      There are fields in the CQE create parameter list that allow the user to define the unit of work that owns or is waiting for the resource. The exit routine specifies any information that might be needed to identify the unit of work, such as the system name, ASID, data area name (for example TCB), data area address, job name, and processor address. This information is used to tell the user exactly which units of work are the bottlenecks in a contention situation. The information allows the ANALYZE process to correlate resource information from multiple classes for a single unit of work. For example, a task (TCB) might own WIDGET 00005 and ENQ, and be waiting for an I/O device. Be sure to save any information needed by the exit routine in control blocks where the information will be available to the exit routine.

   See "Contention Queue Element (CQE) Create Service" on page 108 for information about the CQE create service.

Address Space Control Block (ASCB) Exit Routine

Write an ASCB exit routine to either:

- Generate a unique diagnostic report about a specific address space data area (ASCB)
- Enhance the output generated by the IPCS SUMMARY subcommand for each ASCB processed

Define the ASCB exit routine in the BLSCUSER parmlib member with the following statement:

```
EXIT EP(name) ASCB
```

name is the name of the ASCB exit routine.

In lieu of writing an ASCB exit routine, a post-formatting exit routine can be written to generate a unique diagnostic report about any data area.

See the following:

- "Post-Formatting Exit Routine" on page 83 for information about post-formatting exit routines
- *z/OS MVS Initialization and Tuning Reference* for information about the EXIT statement in a BLSCUSER parmlib member

Note: IBM does not provide any ASCB exit routines.

Possible Uses

The ASCB exit routine generates information related to an ASCB in a dump currently being processed by IPCS, which could be either:

- Installation address space storage
- IBM component address space storage

Programming Considerations

Be aware of the following information when writing an ASCB exit routine.

Performance Implications

When the IPCS user enters either the ASCBEXIT or the SUMMARY subcommand with the FORMAT parameter, IPCS gives each ASCB exit routine control once.

See *z/OS MVS IPCS Commands* for information about the ASCBEXIT and SUMMARY subcommands.

Restrictions and Limitations

None.

Data Areas

- **ADPLEXTN:** (Mapped by mapping macro BLSABDPL) IPCS sets field ADPLEFCD to ADPLEFAC (X'0002') upon entry to an ASCB exit routine to permit a common entry point to serve as an ASCB exit routine as well as one of the following types of exit routines:
  - ANALYZE exit routine
  - Post-formatting exit routine
  - TCB exit routine
  - Verb exit routine

  The registers and other conditions on entry to each of these types of exit routines are the same. Use field ADPLEXTN to determine the purpose of a given call to a multi-function entry point.

Passing Control

An ASCB exit routine receives control in one of the following ways:

- The ASCBEXIT subcommand allows the IPCS user to invoke one or all ASCB exit routines. To have the ASCBEXIT subcommand invoke an installation-provided ASCB exit routine, either:
  - Define the ASCB exit routine in the BLSCUSER parmlib member
  - Invoke the ASCB exit routine by module name

- The CBFORMAT subcommand with the EXIT parameter will invoke all ASCB exit routines defined in the totality of IPCS parmlib members, if an ASCB is selected for formatting. IPCS invokes the ASCB exit routines immediately after IPCS formats the ASCB.
- The SUMMARY subcommand with the FORMAT parameter invokes all ASCB exit routines defined in the totality of IPCS parmlib members. IPCS invokes the ASCB exit routines immediately after IPCS formats the ASCB and related data areas.

See the following:

- Chapter 9, "Installing IPCS Exit Routines," on page 93 for information about installing exit routines
- *z/OS MVS IPCS Commands* for the ASCBEXIT, CBFORMAT, and SUMMARY subcommands

Input

In addition to the input considerations described in "Conditions on Entry to an IPCS Exit Routine" on page 50, the following special input considerations apply to ASCB exit routines.

Note: The following section describes what IPCS sets on entry to an ASCB exit routine. However, if you are trying to invoke an ASCB exit routine, you must set these values.

On entry to an ASCB exit routine, register 1 points directly to the IPCS exit parameter list (ABDPL data area).

- IPCS sets field ADPLASID to identify the address space for which ASCB formatting is requested, whenever the ASCB resides in virtual storage.

  This field can be set by ASCB exit routines to specify an ASID before the exit routines invoke the storage access service to retrieve virtual storage, or when the exit routines call formatter or model processor services to retrieve virtual storage data before formatting it.
- IPCS sets field ADPLCBP (also known as ADPLTCB) to contain the virtual storage dump address of the block being processed. If a block does not reside in virtual storage, this field is zeroed, and the exit routine must use ADPLESYP to find the address of the block in the storage mapped by BLSRESSY. This can only happen in the IPCS environment.
- When the exit routine is written to run in an IPCS environment, IPCS sets field ADPLESYP to address a block of storage described by mapping macro BLSRESSY. That block of storage in turn describes the address space, address, and the type of data for which a formatting exit routine is being invoked. This allows the exit routine to pass the data unaltered, to the storage access function of the IPCS symbol service and retrieve an image of the block from the dump.

  Note: The block of storage described by mapping macro BLSRESSY can reside in storage whose address is greater than X'FFFFFF'. Formatting exit routines that wish to utilize this support must run in AMODE(31) during that portion of their processing that accesses this parameter.

  When the exit routine is written to run in a SNAP environment, this pointer contains zero upon entry to the post-formatting exit routine.
- IPCS sets field ADPLEFCD of the IPCS task variable to ADPLEFAN (X'0001'). The ADPLEFCD field allows a user to write an exit routine which acts as both a verb exit routine which formats data and as an ASCB exit routine which produces information about a specific ASCB.

See *z/OS MVS Programming: Assembler Services Reference ABE-HSP* and *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for information about the BLSABDPL and BLSRESSY macros.

Output

In addition to the output considerations described in "Conventions for Return to Caller for an IPCS Exit Routine" on page 54, the following special output considerations apply to ASCB exit routines.

Registers at Exit

Register
Contents

0 through 14
Same as on entry

15
Ignored

Component Trace Exit Routines

You can use the IPCS CTRACE subcommand to format your installation-supplied application trace data. To use IPCS to format this data, you provide the following:

CTRACE format table
Informs IPCS of the component trace entries (CTEs) and how they are to be formatted. Through the FMTTAB parameter on the CTRACE macro with the DEFINE parameter, you specify the name of the load module containing the CTRACE format table. With Release 10, you can define components through the CTRACE statement of the BLSCUSER parmlib member or other parmlib members starting with BLSCECT.

See "CTRACE Format Table" on page 64.

IPCS model
Specifies a template for the format of a particular CTE. A model is optional.

See "IPCS Models" on page 65.

CTRACE formatter
Consists of executable code that formats a CTE. The code can use an IPCS model. A formatter is optional.

See "CTRACE Formatter" on page 66.

CTRACE buffer find exit routine
Locates trace buffers, if the buffers are externalized in a dump. The routine places the locations of the buffers into the equate symbol (ES) record. You place the name of the routine in the CTRACE format table. The CTRACE buffer find exit routine is required.

See "CTRACE Buffer Find Exit Routine" on page 67.

CTRACE filter/analysis (CTRF) exit routine
Filters trace entries for formatting purposes or for analysis of the data. The routine applies filter options to each CTE. You place the name of the routine in the CTRACE format table. The CTRF routine is optional.

See "CTRACE Filter/Analysis (CTRF) Exit Routine" on page 69.

After you set up the table, model, and routines, IPCS does the following in response to a CTRACE subcommand:

1. Locates the CTRACE format table.
2. Loads the CTRACE buffer find exit routine. Calls the routine repeatedly to locate the trace buffers in the dump until all trace buffers are located.
3. Loads the CTRF exit routine, if you specified one. Calls the routine for each entry in the trace buffers.

   If you did not specify a CTRF exit routine, IPCS formats all CTEs in the buffers that pass the component trace global filters, such as timestamp range selection.
4. Formats all CTEs that pass the filters. To format the CTEs, IPCS uses your CTRACE formatter, your models, or both, if provided. If not provided, IPCS displays the information in hexadecimal.

Figure 11 on page 64 illustrates the interactions among the following. In the figure, the format table is located in SYS1.MIGLIB. However, the format table can be in any library available to IPCS.

ABC
An application, running in ASID(01), that generates component trace data. ABC issues the CTRACE macro to define the ABC application to the component trace service.

IPCS
IPCS running in the user's TSO/E address space. From IPCS, a user enters the CTRACE subcommand CTRACE COMP(ABC) to request that trace data from ABC be processed.

FORMAT
The name of the load module containing the CTRACE format table. FORMAT contains non-executable code that defines a CTRACE format table (FORMTAB) through multiple invocations of the ITTFMTB macro.

FORMTAB
The name of the CTRACE format table.

FINDRTN
The name of the CTRACE buffer find exit routine.

FILTERTN
The name of the CTRF exit routine.

FORMAT1
The name of the formatter associated with event identifier 1, as specified by the ITTFMTB macro.

MODEL1
The name of the model associated with event identifier 2, as specified by the ITTFMTB macro.

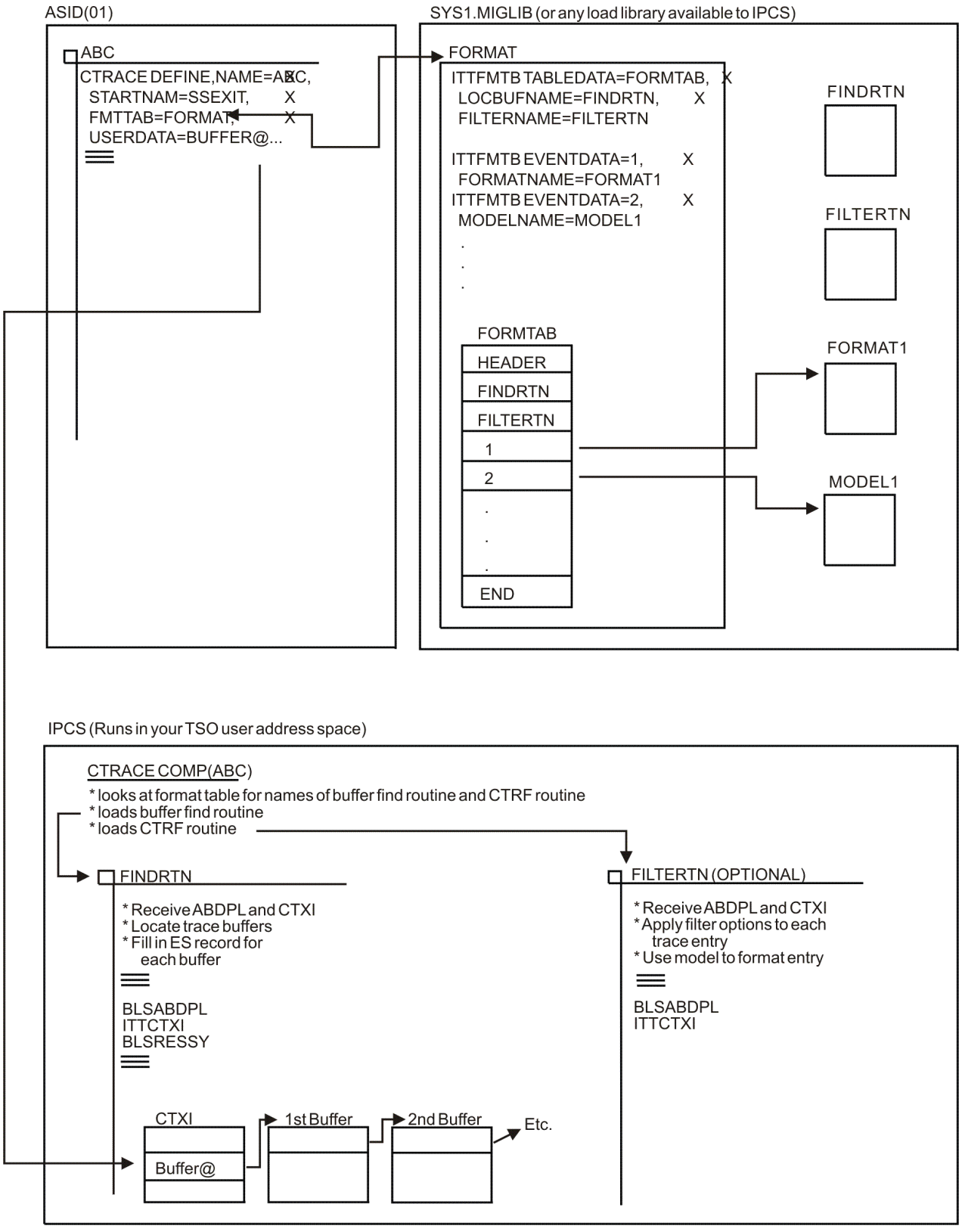

Figure 11. Formatting Installation-Supplied Application Trace Data with IPCS

CTRACE Format Table

The CTRACE format table provides information to IPCS for formatting CTEs. To set up a format table, do the following:

1. Specify the name of the format table on the FMTTAB parameter of the CTRACE macro with the DEFINE parameter.
2. Use the ITTFMTB macro to create and map the format table. Create the format table as follows:
   a. In a separate load module, issue the ITTFMTB macro to define the beginning of the table (TABLEDATA parameter). At this time, you can also identify the name of your CTRACE buffer find exit routine (LOCBUFNAME parameter) and the name of your CTRF exit routine (FILTERNAME), if being provided.
   b. Issue an ITTFMTB macro for each unique CTE type that your component or application uses. In each ITTFMTB, the EVENTDATA parameter specifies the event identifier (CTEFMTID) for the CTE type. For example, if you have 50 types of CTEs, issue the ITTFMTB macro 50 times, once for each CTEFMTID.

      When you create each CTE using a ITTCTE mapping macro, the CTEFMTID field indicates the format table entry.
   c. Issue the ITTFMTB macro to define the end of the format table (TABLEEND parameter).
3. Place the load module in a load library available to IPCS. See "Making Load Libraries Available to IPCS" on page 54 for information about making a load library available to IPCS.
4. The system creates the format table at assembly time.

Contents of the CTRACE Format Table

The table contains:

- Optional entries that point to CTRACE formatting support
- The name of your CTRACE buffer find exit routine
- The name of your CTRF exit routine
- Using the ITTFMTB macro, unique trace identifiers

See the following:

- *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* for information about the CTRACE macro
- *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for information about the ITTFMTB macro

IPCS Models

To display a CTE, you can use a model, which is a template. For most CTEs, usually only a model is needed, rather than a formatter or formatter and model.

In the CTRACE format table, specify a model by either a name or address on the MODELNAME or MODELADDR parameter of the ITTFMTB macro. Use MODELADDR when the model resides in the same load module as the CTRACE format table. Use MODELNAME when the model does not reside in the same load module as the CTRACE format table. The model must reside in a load library available to IPCS. See "Making Load Libraries Available to IPCS" on page 54 for how to make a load library available to IPCS.

When a model is specified by name in the format table, CTRACE loads it and retains its entry point for subsequent use. All such loaded support is deleted when CTRACE completes.

If the model for a particular format ID key (CTEFMTID in the ITTCTE mapping macro) is a model, IPCS calls the model processor service specifying that model and the appropriate view control value from the table.

See the following:

- "Model Processor Formatting (MPF) Exit Routine" on page 81 for information about models
- *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for information about the ITTFMTB macro

CTRACE Formatter

To display a CTE, you can use a formatter, which is executable code. The formatter may use an IPCS model.

In the CTRACE format table, specify a formatter by either a name or address on the FORMATNAME or FORMATADDR parameter of the ITTFMTB macro. Specifying it by address is more efficient. Use FORMATADDR when the formatter code resides in the same load module as the CTRACE format table. Use FORMATNAME when the formatter code does not reside in the same load module as the CTRACE format table. The formatter must reside in a load library available to IPCS. See "Making Load Libraries Available to IPCS" on page 54 for how to make a load library available to IPCS.

See *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for information about the ITTFMTB macro.

Programming Considerations

IPCS calls the CTRACE formatter for each CTE.

A CTRACE formatter may use a 4-kilobyte work area. The address of the work area is in the CTXIUSWA field of the CTXI data area. Each time a formatter is used, it can use the same work area, providing continuity across multiple invocations of the formatters. However, this work area is not preserved for use by other CTRACE exit routines.

When a formatter is specified by name in the format table, CTRACE loads it and retains its entry point for subsequent use. All such loaded support is deleted when CTRACE completes.

Performance Implications

Some components might produce a very large number of CTEs, and because a CTRACE formatter might be called for each one, consider exploiting the 4-kilobyte work area, or the CTXIUSER and CTXIUSRL fields to anchor storage obtained by the formatters. On the first call, IPCS initializes CTXIUSER and CTXIUSRL to a value of zero. If, on some subsequent call (call *n*), a formatter updates either of these fields, the updated information appears in the field as input to a formatter on the next call (call *n+1*). CTRACE frees any storage anchored in CTXIUSER.

Restrictions and Limitations

The 4-kilobyte work area, pointed to by CTXIUSWA, provided by CTRACE for use by all exit routines and formatters can be used only by component-supplied routines or formatters. Do not attempt other uses.

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a mapping of the CTXI data area.

Data Areas

- ABDPL (mapped by mapping macro BLSABDPL)
- ADPLPFMT (mapped by mapping macro BLSABDPL)
- ADPLPFXT (mapped by mapping macro BLSABDPL)
- CTXI (mapped by mapping macro ITTCTXI)

Input

On entry to a CTRACE formatter, register 1 points to a parameter list containing the addresses of the following data areas:

1. ABDPL
2. CTXI

The CTXIFMP field points to an initialized format parameter. If the format table specifies a model by address for the current CTE, IPCS loads that model; if the format table specifies the model by name, IPCS sets the format parameter field, ADPLPPTR, to point to the model.

Output

The CTRACE formatter must call the formal model processor or the print service to produce printed output.

See "Format Model Processor Service" on page 120 and "Standard Print Service" on page 133 for information about these services.

CTRACE Buffer Find Exit Routine

In order for the CTRACE subcommand to process the CTEs produced by an installation-supplied application, IPCS requires an installation-supplied CTRACE buffer find exit routine.

To define a CTRACE buffer find exit routine to IPCS, the routine must reside in a load library available to IPCS and the name of the exit routine must be provided in the CTRACE format table.

See the following:

- "Making Load Libraries Available to IPCS" on page 54 for information about making a load library available to IPCS
- "Component Trace Exit Routines" on page 62 for information about using a CTRACE buffer find exit routine in formatting installation-supplied application trace data
- "CTRACE Format Table" on page 64 for information about creating the CTRACE format table
- *z/OS MVS Programming: Authorized Assembler Services Guide* for producing CTEs for an installation-supplied application

Possible Uses

CTRACE buffer find routines are used exclusively by the CTRACE subcommand to locate the CTRACE buffers in a dump associated with a particular component.

Programming Considerations

IPCS calls the CTRACE buffer find routine until all buffers are located.

The CTRACE buffer find routine may use a 4-kilobyte work area. The address of the work area is in the CTXIUSWA field of the CTXI data area. Each time the routine is used, it can use the same work area, providing continuity across multiple invocations of the routine. However, this area is not preserved for use by other CTRACE exit routines or CTRACE formatters.

Performance Implications

Consider exploiting the 4-kilobyte work area, or the CTXIUSER and CTXIUSRL fields to anchor storage obtained by the CTRACE buffer find exit routine. On the first call, IPCS initializes CTXIUSER and CTXIUSRL to a value of zero. If, on some subsequent call (call *n*), the exit routine updates either of these fields, the updated information appears in the field as input to the exit routine on the next call (call *n+1*). CTRACE frees any storage anchored in CTXIUSER.

Restrictions and Limitations

The 4-kilobyte work area, pointed to by CTXIUSWA, provided by CTRACE for use by all exit routines and formatters can be used only by component-supplied routines or formatters. Do not attempt other uses.

The CTRACE buffer find exit routine must reside in a library available to IPCS.

Data Areas

- ABDPL (mapped by mapping macro BLSABDPL)
- CTXI (mapped by mapping macro ITTCTXI)
- Equate symbol (ES) record

See “Equate Symbol Service” on page 111 for more information on equate symbol records.

Passing Control

When the IPCS user enters the CTRACE subcommand, IPCS calls the installation-supplied CTRACE buffer find exit routine to locate the buffers in a dump. IPCS saves this information in the symbol table for the dump. As long as the symbols remain intact, the CTRACE buffer find exit routine is not called again for the same dump.

Returning to Caller

The following list describes the return codes and the actions they cause.

Code
Action

00
The ES record defines a buffer, and there may be more buffers, so call again.

04
The ES record defines a buffer, and there are no more buffers, so do not call again.

08
The ES record does not define a buffer, and there are no more buffers, so do not call again.

16
Difficulties encountered, like missing storage in the path to the buffers. ES record does not describe a buffer, do not call again.

Design Example

The following is a high-level example of a CTRACE buffer find exit routine:

1. Locate the work area (field CTXIUSWA in the CTXI data area) and determine if this routine has been called before.
2. For the first call to this routine, do the following:
   a. Initialize CTXIUSWA.
   b. Use IPCS exit services or CTXI USERDATA, or both, to locate control information in the dump needed to identify the trace buffers. Save the anchor address in the work area.
3. Locate the next trace buffer defined by the CTRACE control structure. Save the buffer identified in the work area.
4. Fill in the ES record to identify the trace buffer.

   The first time the exit is called, BLSRESSY is in 31-bit format.

   When the CTRACE buffer is in 31 bit storage, continue to use the 31 bit ES record. The following fields are initialized:

   DOF is set to 00000000
   DLE is set to 00000004
   DOB is set to 00
   DLB is set to 00
   DTY is set to E4 (type = U)
   DTD is set to 40 ... (blanks)

   When the CTRACE buffer is in 64 bit storage, the ES record will need to be converted to a 64 bit format. The storage provided for the ES record is large enough to hold the 64 bit format. To convert the ES record to this format, remap the ES record using the BLSRESSY macro with ABITS=64 parameter. The exit routine must then set the RST field at offset +x'3B' to '2' (x'F2') and set the LAD field at +x'4C' to a 64 bit address. The exit routine must also set appropriate values to the following fields as their offsets have changed and are no longer initialized: DOF, DLE, DOB, DLB, DTY, and DTD.
5. Set the return code.

6. Return to the caller.

Input

On entry to a CTRACE buffer find exit routine, register 1 points to a parameter list containing the addresses of the following data areas:

1. ABDPL
2. CTXI

Field CTXIESR (in CTXI) points to an initialized BLSRESSY ABITS(31) structure.

Output

The CTRACE buffer find exit routine describes a CTRACE buffer by filling in the BLSRESSY structure. For IPCS z/OS R8 and later, the buffer find routine might return BLSRESSY ABITS(31) or BLSRESSY ABITS(64) formats to CTRACE. The buffer passed to the CTRACE buffer find exit routine is large enough to hold the largest BLSRESSY ABITS(64) structure that IPCS supports.

BLSRESSY ABITS(64) structures can describe buffers as long as X'FFFFFFFF_FFFFFFFF' bytes in length. CTRACE imposes an artificial limit of X'00000000_80000000' bytes and recommends the use of much smaller buffers. Trace entries within any buffer form a list structure. Loss of one page when a dump is recorded or damage to a single byte can render all remaining trace entries illegible.

CTRACE Filter/Analysis (CTRF) Exit Routine

If an application supports specific tracing options, the application can provide a CTRF exit routine. This routine receives control for each installation-supplied CTE.

Possible Uses

A CTRACE filter/analysis (CTRF) exit routine can:

- Perform statistical analysis of the CTEs
- Provide additional filtering
- Limit the number of entries processed

To define a CTRF exit routine to IPCS, the routine must reside in a load library available to IPCS. There are two ways to define a CTRF exit routine to IPCS:

- Through the FILTERNAME parameter on the ITTFMTB macro.

  Use this method when you are designing CTRACE support and you want to include a CTRF exit routine as part of your design. See “Component Trace Exit Routines” on page 62 for further information.
- Through the USEREXIT parameter on the CTRACE subcommand in IPCS.

  This method is useful for someone doing problem determination who decides that additional filtering (beyond that provided in the original design) is necessary. See *z/OS MVS IPCS Commands* for information about the CTRACE subcommand.

You can use the PANDEF parameter in a BLSCUSER parmlib member to define an input panel to accompany your CTRF exit routine. The input panel will be available when the CTRACE processing option in the IPCS dialog is used. The user can specify CTRACE options through the input panel. If you want to provide an accompanying help panel for the input panel, specify the help panel within the contents of the input panel.

The naming convention for input panels is to begin all names with COMPOPT. This ISPF variable should be set with the string of options to be collected on the input panel.

See the following:

- The section on the ANALYSIS option of the IPCS dialog in *z/OS MVS IPCS User's Guide* for information on the CTRACE processing option.
- *z/OS ISPF Dialog Tag Language Guide and Reference* for information about creating panels.
- *z/OS MVS Initialization and Tuning Reference* for information about using the PANDEF parameter in a BLSCUSER parmlib member.

Programming Considerations

IPCS calls the CTRF exit routine for each CTE.

The CTRF exit routine may use a 4-kilobyte work area. The address of the work area is in the CTXIUSWA field of the CTXI data area. Each time the routine is used, it can use the same work area, providing continuity across multiple invocations of the routine. However, this work area is not preserved for use by either the CTRACE buffer find exit routine or a CTRACE formatter.

Performance Implications

Some components might produce a very large number of CTEs, and because the CTRF exit routine might be called for each one, consider exploiting the 4-kilobyte work area, or the CTXIUSER and CTXIUSRL fields to anchor storage obtained by the exit routine. On the first call, IPCS initializes CTXIUSER and CTXIUSRL to a value of zero. If, on some subsequent call (call *n*), the exit routine updates either of these fields, the updated information appears in the field as input to the exit routine on the next call (call *n+1*). CTRACE frees any storage anchored in CTXIUSER.

Restrictions and Limitations

The 4-kilobyte work area, pointed to by CTXIUSWA, provided by CTRACE for use by all exit routines and formatters can be used only by component-supplied routines or formatters. Do not attempt other uses.

Data Areas

- **ABDPL (mapped by mapping macro BLSABDPL)**
- **CTXI (mapped by mapping macro ITTCTXI)**

Passing Control

When the IPCS user enters a CTRACE subcommand with the USEREXIT parameter, CTRACE calls, in this order:

- Any component-supplied CTRF exit routines, as defined through the FILTERNAME parameter on the ITTFMTB macro
- Any CTRF exit routines, as specified through the USEREXIT parameter on the CTRACE subcommand.

CTRACE calls the exit routines for each CTE that passes the standard filtering options in effect.

IPCS calls the CTRF exit routine one additional time, with the bit CTXIDONE in the CTXI turned on, after all CTEs are processed. This last call will not occur if either exit routine returns code 16, or if the exit routine returns code 12.

Returning to Caller

The following list describes the return codes and the actions they cause.

Code
Action

00
Normal processing of the entry

04
Reread CTEs from the first

08
The current entry is bypassed

12
No further calls to the CTRF exit routine

16
Ending of the subcommand

Design Example

The following is a high-level example of a CTRF exit routine:

1. Locate the work area (field CTXIUSWA in the CTXI data area) and determine if this routine has been called before.
2. For the first call to this routine, do the following:
   a. Initialize CTXIUSWA.
   b. On the first call, parse the application-specific option string to determine which CTEs must pass through filtering and be formatted. If any errors occur, the CTRF exit routine issues an error message using the IPCS expanded print service and sets a failing return code. The routine saves the results in the work area.
3. Apply application-unique filter options, or defaults, to the CTE.
4. Set the return code.
5. Return to the caller.

Input

On entry to a CTRF exit routine, register 1 points to a parameter list containing the addresses of the following data areas:

1. ABDPL
2. CTXI

Output

The CTRF exit routine can use the exit environment to produce any desired output.

Control Block Formatter Exit Routine

The control block formatter exit routine is called by the control block formatter service to assist in control block formatting.

Define a control block formatter exit routine in the BLSCUSER parmlib member with the following statement:

```
DATA FORMAT(name[,level])
```

name is the name of the control block formatter exit routine. ***level*** is a function systems mode ID (FMID), which indicates a version and release of the MVS system. IPCS gives the formatting routine control when the control block to be formatted is supported by that MVS version and release. The level qualifier associates the formatter with an application programming interfaces supported by MVS releases with one of the following FMIDs:

- JBB2125 supported data areas in virtual and real storage and had none of the services involving the BLSRESSY structure. A formatter written at this level uses the information in the ABDPL and ADPLEXT only. This level does not support data blocks residing in a data space.
- HBB3310 introduced all of the services associated with IPCS and accomodated description of data spaces. This level supports 31-bit storage formatting only.

- HBB7703 introduced a set of IPCS structures capable of describing 64-bit storage. Formatter written to accept descriptions of their data anywhere in 64-bit storage will be given control by IPCS in Release 10.

The default for API support is FMID HBB3310.

See the following:

- "Control Block Formatter Service" on page 101 for information about the control block formatter service
- *z/OS MVS Initialization and Tuning Reference* for information about the DATA statement and its parameters in a BLSCUSER parmlib member

Possible Uses

Control block formatter exit routines can be written to provide functions that cannot be accomplished using the format model processor service. A control block formatter exit routine can:

- Select the correct model for a block that has several variations
- Adjust the virtual address to account for a prefix
- Determine array or list dimensions and set the format parameter accordingly
- Establish the actual block length to limit formatting

See "Format Model Processor Service" on page 120 for information about the format model processor service.

Programming Considerations

Be aware of the following information when writing a control block formatter exit routine.

Performance Implications

None.

Restrictions and Limitations

A control block formatter exit routine is considered to be an extension of the data area formatter service, and as such is exposed to both the basic and the extended format interfaces.

Any control block formatter exit routine should be able to interpret the format parameters ADPLPFMT and ADPLPFXT. For those that can only interpret the basic format interface (ADPLPFMT only), the control block formatter service will attempt to convert to the basic interface. Specify **JBB2125** as the second parameter of the FORMAT parameter of the DATA statement in BLSCUSER parmlib member that defines your control block formatter exit routine:

```
DATA FORMAT(name,JBB2125)
```

When writing a control block formatter exit routine for systems that support 64-bit storage, specify HBB7703 as the second parameter of the FORMAT parameter of the DATA statement in BLSCUSER. Exits written to accept the HBB7703 API that also need to run against dumps of earlier releases should check the format of the BLSRESSY structure(s) passed. Ones solely designed to run on HBB7703 or higher levels may assume that the 64-bit format will be passed.

The interface cannot be converted if the block does not reside in:

- Virtual storage
- Real storage
- The dump header record
- The CPU status record

See *z/OS MVS Initialization and Tuning Reference* for information about creating a BLSCUSER parmlib member.

Data Areas

- **ABDPL**
- **ADPLPFMT, ADPLPFXT:** The format parameters can be referenced for decision making, and altered to influence the formatting process.

Passing Control

The control block formatter service calls control block formatter exit routines when the control block formatter service has been called to format a data area with the acronym that is associated with the control block formatter exit routine.

Returning to Caller

The following list describes the return codes and the actions they cause.

Return Code
Action

00
Normal operation

Nonzero
Sets ADPLPRNF and a return code of 4 is sent to the initiating exit program.

Input

On entry to a control block formatter exit routine, register 1 points to a parameter list containing the addresses of the following data areas:

1. ABDPL
2. ADPLPFMT

Output

A control block formatter exit routine is expected to call the format model processor service after it has adjusted the format parameters.

See "Format Model Processor Service" on page 120 for information about the format model processor service.

Control Block Status (CBSTAT) Exit Routine

Write a control block status (CBSTAT) exit routine to do the following:

- Perform analysis and generate a unique diagnostic report that is not currently available in IPCS
- Perform analysis and enhance the report produced by IBM-supplied CBSTAT exit routines

CBSTAT exit routines do not format data areas or report on normal conditions.

This exit routine might process a dump for either:

- Installation application storage
- IBM component data areas and storage

Before writing a CBSTAT exit routine, become familiar with the existing CBSTAT exit routines to avoid duplicating functions that are already available. IBM supplies CBSTAT exit routines for ASCBs and TCBs. CBSTAT exit routines for ASCBs generate status for the ASCB or address space. CBSTAT exit routines for TCBs generate status for the TCB or the task in general.

Define the CBSTAT exit routine in the BLSCUSER parmlib member with the following statement:

```
EXIT EP(epname[HBB7703]) CBSTAT(cbname)
```

epname is the name of the CBSTAT exit routine.

cbname is the name of the control block for which status is being obtained.

EP(*epname*,HBB7703) indicates that the CBSTAT routine will accept a BLSACBSP in 64-bit format. Exits written to accept the HBB7703 API that also need to run against dumps of earlier releases should check the format of the BLSACBSP structure passed. Ones solely designed to run on HBB7703 or higher levels may assume that the 64-bit format will be passed.

See the following:

- *z/OS MVS IPCS Commands* for information about the CBSTAT exit routines supplied by IBM
- *z/OS MVS Initialization and Tuning Reference* for information about the DATA statement in a BLSCUSER parmlib member

Possible Uses

Write a CBSTAT exit routine to generate the status for other data areas in the system besides the ASCB and TCB. For example, a CBSTAT exit routine could be written to describe the status of a data set data area (DCB) or a unit data area (UCB).

Programming Considerations

Be aware of the following information when writing a CBSTAT exit routine.

Performance Implications

When the IPCS user enters the CBSTAT subcommand against a dump, IPCS gives each CBSTAT exit routine control once.

Restrictions and Limitations

To print data area status, the exit routines must use the expanded print service to print all messages. Set the PPR2OCOL option flag in BLSUPPR2 to indicate that IPCS is to use the predetermined indentation level set in ADPLSCOL.

Leave the overflow indentation level at the default of 2, unless a specific reason exists for formatting it differently. By using the expanded print service in this way, the output will merge consistently with IBM generated output to produce easily readable reports.

See “Expanded Print Service” on page 116 for information about the expanded print service.

Data Areas

- **ABDPL:** IPCS task variable (mapped by mapping macro BLSABDPL)
- **CBSP:** The CBSTAT parameter list (mapped by mapping macro BLSACBSP)

Passing Control

A CBSTAT exit routine receives control from IPCS when the IPCS user enters the CBSTAT subcommand against a dump. IPCS invokes CBSTAT exit routines in the order in which they are listed in the BLSCUSER parmlib member.

See the following:

- Chapter 8, “Writing IPCS Exit Routines,” on page 49 for information about installing IPCS exit routines by listing them in the BLSCUSER parmlib member.
- *z/OS MVS Initialization and Tuning Reference* for information about creating BLSCUSER parmlib members.

Input

In addition to the input considerations described in "Conditions on Entry to an IPCS Exit Routine" on page 50, the following special input considerations apply to CBSTAT exit routines.

On entry to a CBSTAT exit routine, register 1 points to a parameter list that contains the addresses of the following:

- The exit parameter list (data area ABDPL)
- The CBSTAT parameter list CBSP (mapped by mapping macro BLSACBSP)

The CBSTAT parameter list contains the following information that is useful for a CBSTAT exit routine:

Field
Description

CBSPAS
Address space description (see mapping macro BLSRDATS within BLSACBSP).

CBSPIDL
Whether the ABITS=31 or ABITS=64 format pf BLSACBSP has been passed to the CBSTAT exit. The ABITS=64 format will always be used by IPCS in Release 10 when HBB7703 exits are given control. The ABITS=31 format is always used otherwise.

CBSPAS2
Fullword containing the ASID.

CBSPLAD
Virtual address of the data area being processed.

CBSPD
Data description (see mapping macro BLSRDATC within BLSACBSP).

CBSPDLE
Length of the data area.

If the data area length was not provided as input to the CB status service and IPCS does not recognize the structure type, the length of the data area will take the default length specified on the SETDEF subcommand. This will generate a situation where the entire data area might not be contained in the buffer being passed to the CB status exit routine. CB status exit routines for data areas that are not known to IPCS must check the CBSPDLE field to determine if the entire data area is contained in the passed buffer. If the passed length is too short, the CB status exit routine must do a dump access for the correct length, in order to view the entire data area.

CBSPDTD
The name of the data area being processed (for example, ASCB).

CBSPBFAD
Address of a buffer containing the data area being processed. For prefixed data areas, the passed buffer will contain the prefix and the main body of the data area. The address placed in CBSPBFAD will point to the start of the main body of the data area.

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for information about the BLSABDPL and BLSACBSP mapping macros.

Output

In addition to the output considerations described in "Conventions for Return to Caller for an IPCS Exit Routine" on page 54 the following special output considerations apply to CBSTAT exit routines.

Registers at Exit

Register
Contents

0 through 14
Same as on entry

15
Ignored

Find Exit Routine

A find exit routine associates a symbol with an AREA or STRUCTURE in a dump. A find routine is given:

- A symbol, such as ASCB00001, CVT, or PRIVATEX
- An IPCS data type, such as STRUCTURE(ASCB), STRUCTURE(CVT), or AREA(PRIVATEX)
- Access to a dump or the active system (ACTIVE | MAIN | STORAGE)

It returns a return code, indicating whether the symbol could be defined, and if it could, returns a definition of the symbol (see BLSRESSY). Filling in all of the fields in BLSRESSY constitutes defining the symbol. Note that in Release 10, IPCS passes BLSRESSY in ABIT=31 format.

Define the find exit routine in the BLSCUSER parmlib member with the following statement:

```
DATA STRUCTURE(structure) FIND(name)
```

structure is the name of the data area to be defined.

name is the name of the find exit routine.

See *z/OS MVS Initialization and Tuning Reference* for information about the DATA statement in a BLSCUSER parmlib member.

Possible Uses

Write a find exit routine to process installation application storage in a dump or IBM component data areas and storage.

Programming Considerations

Be aware of this information when writing a find exit routine.

Performance Implications

IPCS records the information collected by the find exit routine in the IPCS symbol table; this information can be reused as often as needed.

Restrictions and Limitations

All IPCS services can be used to resolve the definition.

The only restriction is that the definition of the symbol being resolved by the find exit routine cannot be requested directly or indirectly while the find exit routine is active. If it is, the IPCS get symbol service will detect the incorrect recursion and return with return code 12.

Data Areas

- **ABDPL (mapped by mapping macro BLSABDPL)**
- **BLSRESSY:** BLSRESSY contains the symbol in the SYM field and the encoded representation of AREA(*name*) or STRUCTURE(*name*) in the DATA statement in the BLSCUSER parmlib member.

Passing Control

A find exit routine receives control when:

- IPCS receives a request for the definition of a symbol that is not currently defined in the symbol table for the dump being processed.
- Attribute AREA(*name*) or STRUCTURE(*name*) is associated with the request in the BLSCUSER parmlib member.
- A find routine has been associated with the type of AREA or STRUCTURE named in the BLSCUSER parmlib member.

See *z/OS MVS Initialization and Tuning Reference* for information about creating the BLSCUSER parmlib member.

Input

In addition to the input considerations described in "Conditions on Entry to an IPCS Exit Routine" on page 50, the following special input considerations apply to find exit routines.

Two parameters are passed to the find exit routine:

1. BLSABDPL
2. BLSRESSY (Note that ABITS=31 format is always used.)

BLSRESSY contains the symbol in the SYM field and the encoded representation of AREA(*name*) or STRUCTURE(*name*) in the DATA statement in the BLSCUSER parmlib member.

- If multiple instances of AREA(*name*) or STRUCTURE(*name*) exist in a dump, the symbol should be used to determine which instance should be located and associated with the symbol.

  IPCS might edit symbols before passing control to a find exit routine. For example, for the ASCB, the following statement appears in an IBM-supplied parmlib member embedded in BLSCECT:

  ```
  SYMBOL PREFIX(ASCB) SUFFIX(COUNT1) STRUCTURE(ASCB)
  ```

  If symbol ASCB00001 is not defined in the symbol table and the subcommand **CBFORMAT ASCB1** is entered, the previous SYMBOL statement causes IPCS to edit the symbol to ASCB00001 prior to passing control to the find routine for STRUCTURE(ASCB).
- If only one instance of AREA(*name*) or STRUCTURE(*name*) should exist in a dump, the symbol in BLSRESSY should be ignored.

Output

The most important output from a find routine is a description of the named AREA or STRUCTURE. The description is provided by filling in fields in the BLSRESSY data area. The definition may be returned in either the ABITS=31 format passed to the FIND exit or in ABITS=64 format. The buffer passed is sized to accommodate the largest BLSRESSY structure supported by IPCS.

The find routine fills in the fields of BLSRESSY as follows:

- The address space in which the block was found in the AS field within BLSRESSY. The AS field is, itself, a structure that is described by BLSRDATS (contained in BLSRESSY mapping macro).
- The address of the block in the LAD field within BLSRESSY.
- Other attributes of the block in the D field within BLSRESSY. The D field is, itself, a structure that is described by BLSRDATC (contained in BLSRESSY mapping macro).
- A remark that pertains to the block in the R field within BLSRESSY. The R field is, itself, a structure that consists of a halfword binary field and 512 bytes of space for remark text. The halfword should be set to indicate the number of bytes of EBCDIC text in the remark supplied.

If the routine receives a device number as the final characters of a symbol, for example, UCBdddd, the device number consists of 4 digits, with leading zeros.

In addition, the following return codes are supplied:

Code
Meaning

00
Successful completion - a usable definition of the AREA or STRUCTURE has been returned in BLSRESSY.

04
Attention conditions detected - but a usable definition of the AREA or STRUCTURE has been returned in BLSRESSY.

08
Error conditions detected - but a usable definition of the AREA or STRUCTURE has been returned in BLSRESSY.

12
Symbol not resolved, possibly due to path to block not available, block failed validation, or IPCS user requested early termination of processing - **NO** usable definition of the AREA or STRUCTURE has been returned in BLSRESSY.

16
Find exit unable to function, for example, GETMAIN failure - **NO** usable definition of the AREA or STRUCTURE has been returned in BLSRESSY.

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a mapping of the BLSRESSY data area.

GTFTRACE Filter/Analysis Exit Routine

The GTFTRACE subcommand formats and prints generalized trace facility (GTF) trace output records.

The interface to filter exit routines and format appendages is upward compatible, and has been extended to provide more facilities to exit routines and appendages. This means that if you have any exit routines or appendages that worked with PRINT DUMP EDIT, you can use them with GTFTRACE.

Possible Uses

GTFTRACE filter/analysis exit routines can:

- Do statistical analysis of the GTFTRACE records
- Provide additional filtering
- Limit the number of records processed

See *z/OS MVS Diagnosis: Tools and Service Aids* for the formats of the GTF records.

Programming Considerations

Be aware of this information when writing a GTFTRACE filter/analysis exit routine.

Performance Implications

None.

Restrictions and Limitations

Name the GTFTRACE filter/analysis exit routine any name, but the routine must reside in a load library currently available to IPCS. See “Making Load Libraries Available to IPCS” on page 54 for how to make a library available to IPCS.

Data Areas

- **ABDPL (mapped by mapping macro BLSABDPL)**

- **FFAP (mapped by mapping macro AHLFFAP)**
- **GTO (mapped by mapping macro AHLZGTO)**
- **GTS (mapped by mapping macro AHLZGTS)**

Passing Control

IPCS calls an installation-provided GTFTRACE filter/analysis exit routine when the EXIT parameter is specified on the GTFTRACE subcommand.

The GTFTRACE filter/analysis exit routine is called for each GTFTRACE record before the standard filtering options are processed.

The GTFTRACE filter/analysis exit is called one additional time when there are no more trace records to be processed, if the EOF parameter was specified on the GTFTRACE subcommand. On the last call, register 2 contains the value 4. The last call does not occur if the GTFTRACE filter/analysis exit routine returns code 16 or 20.

Returning to Caller

The following list describes the return codes provided and the actions they cause.

Code
Action

00
Print the print buffer and call the exit routine again for the same trace record

04
Print the print buffer, then locate the next record

08
Normal filtering of the record

12
Bypass the record, then locate the next record

16
Print the print buffer, then no further calls to the exit routine

20
Normal filtering, then no further calls to the exit routine

24
End the GTF subcommand

28
Unconditional formatting of the record

Input

On entry to a GTFTRACE filter/analysis exit routine, register 1 points to the FFAP. Register 2 will contain zeros for all calls except the last call.

The FFAP contains the address of the ABDPL, therefore, all exit services are available to your GTFTRACE filter/analysis exit routine.

Output

The GTFTRACE filter/analysis exit routine can use the exit environment to produce any desired output.

GTFTRACE Formatting Appendage

The GTFTRACE subcommand formats and prints GTF trace output records. The interface to filter exits and format appendages is upward compatible, and has been extended to provide more facilities to exits and

appendages. This means that if you have any exits or appendages that worked with PRINT DUMP EDIT, you can use them with GTFTRACE without modification.

Possible Uses

An installation-provided GTFTRACE formatting appendage can obtain formatted output of GTF records containing a particular FID, and with an EID in the user range.

See *z/OS MVS Diagnosis: Tools and Service Aids* for the formats of the GTF records and a list of EIDs and FIDs for user trace records.

Programming Considerations

Be aware of this information when writing a GTFTRACE formatting appendage.

Performance Implications

IPCS calls the installation-provided GTFTRACE formatting appendage for each trace record. For some components, the number of trace records can be very large. Use the field in the WKAL to anchor dynamically acquired storage and improve performance. Use the FID value of the record you are going to process as an index into the WKAL.

Restrictions and Limitations

Name the GTFTRACE formatting appendage one of the following:

- AMDUSRxx
- HMDUSRxx
- IMDUSRxx

xx is the character representation of the FID value in the records that the GTFTRACE formatting appendage will format. The GTFTRACE formatting appendage should reside in a load library currently available to IPCS. See “Making Load Libraries Available to IPCS” on page 54 for how to make a library available to IPCS.

Data Areas

- **ABDPL (mapped by mapping macro BLSABDPL)**
- **ADPLPFMT (mapped by mapping macro BLSABDPL)**
- **ADPLPFXT (mapped by mapping macro BLSABDPL)**
- **FFAP (mapped by mapping macro AHLFFAP)**
- **GTO (mapped by mapping macro AHLZGTO)**
- **GTS (mapped by mapping macro AHLZGTS)**
- **WKAL (mapped by mapping macro AHLWKAL)**

Passing Control

IPCS calls an installation-provided GTFTRACE formatting appendage when it encounters a GTF record that:

- Contains an element identifier (EID) with the first four bits equal to X‘E’
- Contains an FID equal to the name suffix
- Passes the standard filtering options in effect

When IPCS encounters an installation record that satisfies current filtering options, and an eligible formatting appendage cannot be loaded, the record is displayed in hexadecimal dump format. This also happens for a particular record if the appendage returns code 12, and for all associated records after the appendage returns code 16.

Returning to Caller

The following list describes the return codes provided and the actions they cause.

Code
Action

00
Print the print buffer and call the appendage again for the same trace record

04
Print the print buffer, then locate the next record

08
Locate the next record. When your appendage uses the model processor to display the record, use this value.

12
Display the record in hexadecimal dump format, then locating the next record

16
Disable the appendage, with a message describing the situation.

Input

On entry to a GTFTRACE formatting appendage, register 1 points to the FFAP.

The FFAP contains the address of the ABDPL; therefore, all exit services are available to your GTFTRACE formatting appendage.

Output

The GTFTRACE formatting appendage can use exit services to produce any desired output.

IPCS initializes a format parameter and passes it to your GTFTRACE formatting appendage, such that the appendage might have little more to do than specify a model and a view control value, then call the model processor exit service.

Model Processor Formatting (MPF) Exit Routine

A model processor formatting exit (MPF) routine is a way to extend the capabilities of the format model processor service.

See “Format Model Processor Service” on page 120 for information about the format model processor service.

Possible Uses

An installation-provided MPF exit routine can enhance the output produced by the model processor or augment its functions. Some functions a model processor can provide include:

- Display a value in decimal rather than hexadecimal
- Add an eye-catcher to an exceptional value
- Interpret a field of data and formatting date by establishing an array, for example
- Capture formatted output and saving it for later processing
- Reject uninteresting data with zero or blank suppression
- Perform alternate character code translation
- Initiate another formatting operation

Programming Considerations

Be aware of this information when writing an MPF exit routine.

Performance Implications

If the MPF exit routine is called many times, consider providing its storage needs from the initiating exit routine, and point to it using ADPLPFXC contained in the ADPLPFMT.

Restrictions and Limitations

None.

Data Areas

- **ABDPL:** The ABDPL contains the address of the print buffer, and other fields of possible use to the MPF exit routine. ABDPL is mapped by mapping macro BLSABDPL.
  - **ADPLPFMT, ADPLPFXT:** The format parameters can be referenced for decision making, and altered to influence the formatting process. Of particular interest is ADPLFXC, an address field that can be used to communicate between the MPF exit and the initiating exit program.
- **FXL:** The format exit routine list (FXL) is a data area for MPF exit routines. It is mapped by macro BLSQFXL, and contains the addresses of data of potential interest to the MPF exit routine, as well as a description of the formatted line.

  Figure 12 on page 82 is a display of the FXL by the debug tool. The display shows data that corresponds to the contents of the print buffer shown in the first line.

```
    +0004  FWDP..... 5C5C5C5C  ASID..... 5C5C      TRQP..... 5C5C5C5C

BLSQFXL: 02A249D8
    +0000  MHDR..... 80068428  MENT..... 0006845C  LPOS..... 0C
    +0009  DPOS..... 16        DLEV..... 0001      DPTR..... 02A65004
    +0010  LNO...... 0002      NLPOS.... 48        NDPOS.... 48
    +0014  ENDX..... 0002      ITMC..... 04        CTF...... 00
    +0018  COLD..... 0000      ENTNM.... 0000

           LINE

               ILBP  IDTP  IDTL  IFLG
               ----  ----  ----  ----
           001 00    06    00    00
           002 0C    16    04    40
           003 20    2A    02    40
           004 34    3E    04    40
```

Figure 12. FXL Data Area as Displayed by the IPCS Debug Tool

Note:

1. When the call to the format model processor exit service is in response to ADPLPOLM, the FXLMENT (model entry pointer) and FXLDPTR (data pointer) refer to the first item on the line. If the call is made in response to a CALLRTN flag in the model, those pointers refer to that entry and the corresponding data. If more than one model entry contained the CALLRTN flag, the pointers refer to the first. In the array of item descriptors, the first always refers to the formatted offset or address, even if suppressed.
2. In Figure 12 on page 82, entry number 2 in the list indicates that the label is in column X‘0C’, the data is in column X‘16’, and the flags indicate that it is a hexadecimal field.

- **address list:** The address list is a list of buffer addresses that describe the data areas currently being formatted. Entries in the list contain three addresses and correspond to the SRCNDX values in the model.
  1. The address of the byte of data in the buffer considered to be at offset zero in the block
  2. The address of the first byte of the buffer
  3. The address of the last byte of the buffer

Passing Control

IPCS gives an MPF exit routine control from the format model processor service after a line of output has been formatted, but before it is printed. IPCS passes control to the MPF exit once for every line of output when the initiating exit program sets the option bit ADPLPOLM on, and places the address of the MPF exit routine in ADPLPLME.

IPCS also passes control to the MPF exit routine when a model entry with the CALLRTN flag set on is processed. An additional call is made to the MPF exit after the model processor has finished. The MPF exit sets a bit in the FXL, FXLLAST, to indicate this special call took place.

Returning to Caller

The following list describes the return codes provided and the actions they cause.

Code
Action

00
The model processor continues and prints the line

04
The model processor continues but does NOT print the line

If the bit FXLQUIT is set on, the model processor ends processing.

Input

On entry to an MPF exit routine, register 1 points to a parameter list containing the addresses of the following:

1. ABDPL data area (mapped by mapping macro BLSABDPL)
2. ADPLPFMT data area (mapped by mapping macro BLSABDPL)
3. FXL data area (mapped by mapping macro BLSQFXL)
4. Buffer address list

Note: Normal output of the format model processor is in the buffer pointed to by ADPLBUF. However, when a message is being constructed in response to the MSGID flag in the model, a different buffer is used, and it is not addressable by the MPF exit routine.

Output

The MPF exit routine can use exit services to produce any desired output.

Post-Formatting Exit Routine

IPCS supports post-formatting exit routines for any type of structure that can be described in a parmlib member DATA statement. ASCB and TCB exit routines are two examples of post-formatting exit routines supported by IPCS.

Define the post-formatting exit routine in the BLSCUSER parmlib member with the following statement:

```
EXIT EP(name) FORMAT(data_structure)
```

name is the name of the post-formatting exit routine.

data_structure is the name of the data area to be formatted.

EP(*epname*,HBB7703) indicates that the post-formatting exit routine will expect a 64-bit BLSRESSY. In the curcumstance that the block of interest resides above the bar, ADPLCBP cannot hold the address of the referenced control block. Exits written to accept the HBB7703 API that also need to run against dumps of earlier releases should check the format of the BLSRESSY structure passed. Ones solely designed to run on HBB7703 or higher levels may assume that the 64-bit format will be passed.

See *z/OS MVS Initialization and Tuning Reference* for the EXIT statement in the BLSCUSER parmlib member.

Possible Uses

A post-formatting exit routine can process installation application storage in a dump or it can process IBM component data areas and storage. IBM supplies specific exit routines for formatting ASCBs and TCBs, and with the post-formatting exit routine you can supply your own formatting routine for any type of structure that can be described in a parmlib member data statement. For example, you could write your own service request block (SRB) formatting routine.

The installation-provided post-formatting exit routine generates information related to the block currently being processed by IPCS.

Programming Considerations

Be aware of this information when writing an post-formatting exit routine.

Performance Implications

IPCS gives each post-formatting exit routine control once during the processing of the SUMMARY subcommand with the FORMAT parameter.

See *z/OS MVS IPCS Commands* for information about the SUMMARY subcommand.

Restrictions and Limitations

None.

Data Areas

- **ADPLEXTN:** (Mapped by the mapping macro BLSABDPL) IPCS sets field ADPLEFCD to ADPLEFSR (X'0004') upon entry to a post-formatting exit routine to permit a common entry point to serve as a post-formatting exit routine as well as one of the following types of exit routines:
  - ANALYZE exit routine
  - ASCB exit routine
  - TCB exit routine
  - Verb exit routine

  The registers and other conditions on entry to each of these types of exit routines are the same. ADPLEXTN can be used to determine the purpose of a given call to a multi-function entry point.

Passing Control

IPCS gives a post-formatting exit routine control when one of the following IPCS subcommands are entered:

- The CBFORMAT subcommand with the EXIT parameter. IPCS invokes all post-formatting exit routines defined in the parmlib members associated with the structure name of the block being processed. IPCS invokes the post-formatting exit routines immediately after the data area is formatted.
- The SUMMARY subcommand with the FORMAT parameter. IPCS invokes all post-formatting exit routines defined in the parmlib members associated with the structure name of the block being processed. IPCS invokes the post-formatting exit routines immediately after the data areas are formatted.

See *z/OS MVS IPCS Commands* for information about the CBFORMAT and SUMMARY subcommands.

Input

In addition to the input considerations described in "Conditions on Entry to an IPCS Exit Routine" on page 50, the following special input considerations apply to post-formatting exit routines.

On entry to an post-formatting exit routine, register 1 points directly to the IPCS task variable (ABDPL data area).

- Field ADPLASID identifies the address space in which the block resides whenever the block resides in virtual storage. When the block resides in an address space other than virtual storage, field ADPLASID is set to zero.

  This field can be set by post-formatting exit routines to specify an ASID before the exit routines invoke the storage access service to retrieve virtual storage, or when the exit routines call formatter or model processor services to retrieve virtual storage data before formatting it.
- Field ADPLCBP, (also known as ADPLTCB) contains the virtual storage dump address of the block being processed. If a block does not reside in virtual storage, this field is zeroed, and the exit routine must use ADPLESYP to find the address of the block in the storage mapped by BLSRESSY. This can only happen in the IPCS environment.
- When IPCS is the host, field ADPLESYP is set to address a block of storage described by macro BLSRESSY. That block of storage, in turn, describes the address space, address, and the type of data for which a formatting exit routine is being invoked. This allows the exit routine to pass the data unaltered, to the storage access function of the IPCS symbol service and retrieve an image of the block from the dump.

  Note: The block of storage described by macro BLSRESSY can reside in storage whose address is greater than X'FFFFFF'. Formatting exit routines that wish to utilize this support must run in AMODE(31) during that portion of their processing that accesses this parameter.

  When SNAP is active, this pointer contains zero upon entry to the post-formatting exit routine.

Output

In addition to the output considerations described in "Conventions for Return to Caller for an IPCS Exit Routine" on page 54 the following special output considerations apply to post-formatting exit routines.

Registers at Exit

Register
Contents

0 through 14
Same as on entry

15
Ignored

Scan Exit Routine

Scan exit routines check the validity of an area or structure in a dump.

- The type of area or structure and its location in the dump is identified by the caller.
- IPCS uses the DATA statements in BLSCUSER and other parmlib members embedded during BLSCECT processing to associate the type of area or structure with the name of the scan exit routine.

If you have coded a scan exit routine called MYSCAN for STRUCTURE(MYDATAAREA) and you want to make it available to IPCS, do the following:

1. Link-edit MYSCAN into a load module library that will be available during your IPCS sessions.
2. Define the scan exit routine in the BLSCUSER parmlib member with the following statement:

```
DATA STRUCTURE(MYDATAAREA) SCAN(MYSCAN)
```

Note: If the system is OS/390® Release 10 or higher, define the scan exit routine in the BLSCUSER parmlib member with the following statement that includes the *level* of the system.

```
DATA STRUCTURE(MYDATAAREA) SCAN(MYSCAN,HBB7703)
```

Exits written to accept the HBB7703 API that also need to run against dumps of earlier releases should check the format of the BLSRSASY structure passed. Ones solely designed to run on HBB7703 or higher levels may assume that the 64-bit format will be passed.

3. Start a new IPCS session to cause your modified BLSCUSER to be processed.

If you have an instance of STRUCTURE(MYDATAAREA) at location 12345 in ASID(45) DSPNAME(MYDATASP), you may request validation of that structure during your IPCS session by entering the subcommand:

```
LIST 12345 ASID(45) DSPNAME(MYDATASP) STRUCTURE(MYDATAAREA)
```

MYSCAN will be invoked to verify that the structure is valid.

See *z/OS MVS Initialization and Tuning Reference* for information about the EXIT statement in a BLSCUSER parmlib member.

Possible Uses

Write a scan exit routine to check the validity of either:

- Installation application storage
- IBM component data areas and storage

Programming Considerations

Be aware of this information when writing a scan exit routine.

Performance Implications

IPCS records in the IPCS symbol table the information collected by the scan exit routine; this information can be reused as often as needed.

Restrictions and Limitations

None - Use any IPCS service to check validity.

Data Areas

- **BLSABDPL**
- **BLSRSASY**

The ABITS=64 format will be passed to any exit identified to IPCS with level HBB7703. The ABITS=31 format will be used otherwise.

Passing Control

IPCS gives a scan exit routine control when:

- IPCS receives a request for the validation of AREA(*name*) or STRUCTURE(*name*) at a designated location in some address space.
- The storage map for the dump being processed indicates one of the following situations:
  - No validation has been performed for the block.
  - Validation has been started but remains incomplete.

  - Validation determined that errors of interest to the user were discovered, and the user wants to see the messages describing the error again.
- A scan exit routine has been associated with the type of AREA or STRUCTURE named. Requests for validation of STRUCTUREs and AREAs may be made through the use of the IPCS subcommands such as LIST. Requests may also be made by analysis routines. Request code XMSPVAL for the storage map service and request codes XSSPACCV and XSSPVAL for the symbol service are used to make such requests in an analysis routine.

See the following:

- "Storage Map Service" on page 137 for information about the XMSPVAL request code
- "Symbol Service" on page 140 for information about the XSSPACCV and XSSPVAL request codes

Input

In addition to the input considerations described in "Conditions on Entry to an IPCS Exit Routine" on page 50, the following special input considerations apply to scan exit routines.

Three parameters are passed to the scan exit routine:

1. BLSABDPL
2. BLSRSASY
3. A parameter that should be forwarded to the service that describes the block being scanned.

BLSRSASY is completely filled in upon entry.

- If the AREA(*name*) or STRUCTURE(*name*) is described externally, the description in field D should be left unaltered, and validation should proceed based on the description of the block passed.
- If the AREA(*name*) or STRUCTURE(*name*) is self-describing, the description in field D should be updated to accurately describe the block.

Output

After a scan exit routine has completed, it supplies a return code that indicates how successful the routine was in producing a usable scan result:

Code
Meaning

00
Successful completion - a usable scan result of the AREA or STRUCTURE has been returned in BLSRSASY.

04
Attention conditions detected - but a usable scan result of the AREA or STRUCTURE has been returned in BLSRSASY.

08
Error conditions detected - but a usable scan result of the AREA or STRUCTURE has been returned in BLSRSASY.

12
Symbol not resolved, possibly due to path to block not available, block failed validation, or IPCS user requested early termination of processing - **NO** usable scan result of the AREA or STRUCTURE has been returned in BLSRSASY.

16
Find exit unable to function, for example, GETMAIN failure - **NO** usable scan result of the AREA or STRUCTURE has been returned in BLSRSASY.

The most important output from a scan exit routine is a description of the results of scan processing. The description is provided by filling in fields in the BLSRSASY data area.

The scan exit routine fills in the fields of BLSRSASY as follows:

- Attributes of the AREA or STRUCTURE are placed in the D field of BLSRSASY. The D field is, itself, a structure that is described by the IPCS mapping macro BLSRDATC.
- Scan flags are placed in the SF field within BLSRSASY.
- The GMT field within BLSRSASY is altered or set to zero (even if it was already zero) to indicate that new scan results were produced.
- One of the following codes is placed in the SASYSRC field within BLSRSASY. The code in the SASYSRC field summarizes the results of the scan processing and indicates the validity of the scanned area or structure.

  Code
  Meaning

  00
  Normal block

  04
  Attention condition(s) detected

  Note: Scan routines provided by IBM use this code when no problem with the block can be identified on the basis of its address, and the image of the block cannot be retrieved from the dump.

  08
  Error condition(s) detected

  Note: Scan routines provided by IBM use this code when pointers in the block scanned address a block in which serious conditions are detected.

  12
  Serious condition(s) detected

  Note: A scan exit routine must produce this code the first time that it is called to scan a block. The process of following pointers to add addressed blocks to the IPCS storage map and determine whether they, in turn, are usable may require multiple calls to the scan exit routine.

  Code 00, 04, or 08 indicate that the block is usable. Code 12 indicates that it is not.
- The C field within BLSRSASY may be used to save data between invocations of a scan exit routine for one AREA or STRUCTURE. The data can include information from the scanning of an unusable block or information that is incomplete. Saving data in the C field can cause subsequent calls to the scan exit routine to bypass the processing related to the saved data.

  The C field is, itself, a structure that consists of a halfword binary field and 2816 bytes of space for data. The halfword should be set to indicate the number of bytes of data supplied.

Task Control Block (TCB) Exit Routine

A TCB exit routine can:

- Generate a unique diagnostic report about a specific TCB
- Enhance the output generated by the IPCS SUMMARY subcommand for each TCB processed

A TCB exit routine can process either:

- Installation application storage
- IBM component data areas and storage

Define the TCB exit routine in the BLSCUSER parmlib member with the following statement:

```
EXIT EP(name) FORMAT(TCB)
```

name is the name of the TCB exit routine.

See *z/OS MVS Initialization and Tuning Reference* for information about the EXIT statement in a BLSCUSER parmlib member.

Possible Uses

A TCB exit routine generates information related to the TCB currently being processed by IPCS.

To avoid duplicating functions that are already available, you should know about existing TCB exit routines before you decide to write new ones. IBM supplies TCB exit routines for the following items:

- Addressing registers
- Data management data areas
- Execute channel program (EXCP) data areas
- Input/output supervisor (IOS) data areas
- Linkage stacks
- Recovery termination manager (RTM) data areas
- Vector Facility data

See *z/OS MVS IPCS Commands* for information about the TCBEXIT subcommand.

Programming Considerations

Be aware of this information when writing a TCB exit routine.

Performance Implications

IPCS gives each TCB exit routine control once when processing a TCBEXIT subcommand or the SUMMARY subcommand with the FORMAT parameter.

See *z/OS MVS IPCS Commands* for information about the TCBEXIT and SUMMARY subcommands.

Restrictions and Limitations

None.

Data Areas

- **ADPLEXTN:** (Mapped by mapping macro BLSABDPL) IPCS sets field ADPLEFCD to ADPLEFTC (X'0003') upon entry to a TCB exit routine to permit a common entry point to serve as a TCB exit routine as well as one of the following types of exit routines:
  - ANALYZE exit routine
  - ASCB exit routine
  - Post-formatting exit routine
  - Verb exit routine

  The registers and other conditions on entry to each of these types of exit routines are the same. Use field ADPLEXTN to determine the purpose of a given call to a multi-function entry point.

Passing Control

IPCS gives a TCB exit routine control when the following subcommands are entered:

- The CBFORMAT subcommand with the EXIT parameter. IPCS invokes all TCB exit routines defined in parmlib members if a TCB is selected for formatting. IPCS invokes the TCB exit routines immediately after the TCB is formatted.
- The TCBEXIT subcommand. This subcommand allows the IPCS user to request one or all TCB exit routines to be invoked. TCB exit routines can be defined in parmlib member BLSCUSER, or they can be invoked by module name.

- The SUMMARY subcommand with the FORMAT parameter. IPCS invokes all TCB exit routines defined in the BLSCUSER parmlib member. IPCS invokes the TCB exit routines immediately after the TCB and related data areas are formatted.

See the following:

- Chapter 8, "Writing IPCS Exit Routines," on page 49 for information about installing IPCS exit routines
- *z/OS MVS IPCS Commands* for information about the CBFORMAT, TCBEXIT, or SUMMARY subcommands
- *z/OS MVS Initialization and Tuning Reference* for information about the BLSCUSER parmlib member

Input

In addition to the input considerations described in "Conditions on Entry to an IPCS Exit Routine" on page 50, the following special input considerations apply to TCB exit routines.

On entry to a TCB exit routine, register 1 points directly to the IPCS task variable (ABDPL data area).

- Field ADPLASID identifies the address space in which the TCB resides whenever the address space resides in virtual storage. When the address space resides in an address space other than virtual storage, field ADPLASID is set to zero.

  This field can be set by TCB exit routines to specify an ASID before t he exit routines invoke the storage access service to retrieve virtual storage, or when the exit routines call formatter or model processor services to retrieve virtual storage data before formatting it.
- Field ADPLTCB (also called ADPLCBP) contains the virtual storage dump address of the TCB being processed. If a block does not reside in virtual storage, this field is zeroed, and the exit routine must use ADPLESYP to find the address of the block in the storage mapped by BLSRESSY. This can only happen in the IPCS environment.
- When IPCS is the host, field ADPLESYP is set to address a block of storage described by mapping macro BLSRESSY. That block of storage, in turn, describes the address space, address, and the type of data for which a formatting exit routine is being invoked. This allows the exit routine to pass the data unaltered, to the storage access function of the IPCS symbol service and retrieve an image of the block from the dump.

  Note: The block of storage described by mapping macro BLSRESSY can reside in storage whose address is greater than X'FFFFFF'. Formatting exit routines that wish to utilize this support must run in AMODE(31) during that portion of their processing that accesses this parameter.

  When SNAP is active, this pointer contains zero upon entry to the post-formatting exit routine.

Output

In addition to the output considerations described in "Conventions for Return to Caller for an IPCS Exit Routine" on page 54 the following special output considerations apply to TCB exit routines.

Registers at Exit

Register
Contents

0 through 14
Same as on entry

15
Ignored

Verb Exit Routine

A verb exit routine can generate a unique diagnostic report that is not currently available in IPCS. A verb exit routine can process either:

- Installation application storage

- IBM component data areas and storage

Verb exit routines can be defined in BLSCUSER, in the IPCSPARM concatenation data set, or invoked by name. Define the verb exit routine in the BLSCUSER parmlib member with the following statement:

```
EXIT EP(name) VERB(verb_name) AMASK(X'aaFFFFFF')
ABSTRACT('text') HELP(helppanel)
```

The variables are the following:

name
The exit routine name.

verb_name
The exit routine verb name.

aa
Can be either:

00
Indicates 24-bit storage accessing.

7F
Indicates 31-bit storage accessing.

text
The abstract shown on the component data analysis panel entry associated with this verb exit.

helppanel
The help panel to accompany this exit routine.

The AMASK, ABSTRACT, and HELP parameters are optional. If the AMASK parameter is not used in the exit entry, the storage accessing mask will default to a 31-bit mask. If the ***verb_name*** of the exit routine is not defined, IPCS will use this ***verb_name*** as the module name to locate it. If the module name with the matching ***verb_name*** is found, but is not a valid verb exit routine, the user should modify the verb exit entry in BLSCUSER by replacing the ***name*** with the user-written module name. The VERBEXIT subcommand allows the IPCS user to request one or all verb exit routines to be invoked.

See *z/OS MVS Initialization and Tuning Reference* for information about the EXIT statement in a BLSCUSER parmlib member.

Possible Uses

Besides generating a unique diagnostic report, verb exit routines also print a title for each major report, and generate table of contents entries.

Before writing a verb exit routine, become familiar with the existing verb exit routines to avoid duplicating functions that are already available.

See the following:

- "Table of Contents Service" on page 146 for information about the table of contents service
- *z/OS MVS IPCS Commands* for a list of some of the verb exit routines provided by IBM

Programming Considerations

Be aware of this information when writing a verb exit routine.

Performance Implications

None.

Restrictions and Limitations

None.

Data Areas

- **ADPLEXTN:** (Mapped by mapping macro BLSABDPL) IPCS sets field ADPLEFCD to X'0000' upon entry to a verb exit routine to permit a common entry point to serve as a verb exit routine as well as one of the following types of exits:
  - ANALYZE exit routine
  - ASCB exit routine
  - Post-formatting exit routine
  - TCB exit routine

  The registers and other conditions on entry to each of these types of exit routines are the same. Use field ADPLEXTN to determine the purpose of a given call to a multi-function entry point.

Passing Control

A verb exit routine receives control when the VERBEXIT subcommand is issued with the module name or verb name of this user-written verb exit routine.

See the following:

- *z/OS MVS IPCS Commands* for information about the VERBEXIT subcommand
- *z/OS MVS Initialization and Tuning Reference* for information about the EXIT statement in the BLSCUSER parmlib member

Input

In addition to the input considerations described in “Conditions on Entry to an IPCS Exit Routine” on page 50, the following special input considerations apply to verb exit routines.

On entry to a verb exit routine, register 1 points directly to the IPCS task variable (ABDPL data area). Field ADPLEXT in ABDPL points to the ABDPL extension, ADPLEXTN. Field ADPLCPPL in ADPLEXTN points to the valid CPPL. If parameters were specified on the VERBEXIT subcommand, field ADPLOPTR (in the ABDPL extension, ADPLEXTN) contains the address of a buffer that contains the parameters. If parameters were specified, field ADPLOPLN (in the ABDPL) contains the length of the parameters. The exit routine must then parse the input parameters and perform the requested functions. In order to be consistent with IPCS, it is suggested that the user take advantage of the TSO/E parser, IKJPARS.

Output

In addition to the output considerations described in “Conventions for Return to Caller for an IPCS Exit Routine” on page 54 the following special output considerations apply to verb exit routines.

A verb exit routine can specify an additional return code in the ABDPL parameter list extension. If a parameter on a user control statement contains an error, the parameter list pointer, ADPLOPTR, in the extension should be changed by the exit to a code of 04, 08, or 12. This code causes IPCS to print an error message on the SYSPRINT data set.

The following shows the user control statement error codes and their corresponding messages:

Code
Message

04
Delimiter error in operand field, which contains the parameters

08
Incorrect parameter

12
Syntax error in the parameters

Chapter 9. Installing IPCS Exit Routines

IPCS exit routines can be installed to format either ABEND and SNAP dumps or dumps formatted by IPCS.

Installing Routine For ABEND/SNAP Formatting

To install an IPCS exit routine for use in formatting ABEND or SNAP dumps, place the exit routine in:

- The SYS1.LPALIB library
- The SYS1.MIGLIB library

Create or customize a BLSCUSER parmlib member to indicate the name and function of the exit routine.

Installing Routine for IPCS Formatting

To install an IPCS exit routine for use by IPCS, place the exit routine in:

- A library in the LNKLST
- A library that is part of the JOBLIB or STEPLIB,
- A library accessed through the TASKLIB parameter of the TSO/E IPCS command
- A data set specified by the ISPLLIB DDNAME, if the exit routine is only accessed during IPCS dialog processing
- The SYS1.MIGLIB library

Then do the following:

- For most exit routines, create or customize a BLSCUSER parmlib member to indicate the name and function of the exit routine.
- For all CTRACE-related exit routines, place the exit routines in a load library available to IPCS.

  For a CTRACE buffer find exit routine, the name of the routine must be placed in the CTRACE format table. Create the CTRACE format table with the ITTFMTB macro.

  For a CTRACE filter/analysis exit routine, invoke the routine explicitly by name on the USEREXIT parameter of the CTRACE subcommand.
- For GTFTRACE-related exit routines, do the following:
  - Give a GTFTRACE filter/analysis exit routine any name. Place the routine in a load library available to IPCS.
  - Give a GTFTRACE formatting appendage one of the following names:
    - AMDUSER***xx***
    - IMDUSER***xx***
    - HMDUSER***xx***

    xx is the character value of the FID records to be processed by the appendage.

See the following:

- “Making Load Libraries Available to IPCS” on page 54
- *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for information about the ITTFMTB macro used to create the CTRACE format table
- *z/OS MVS IPCS Commands* for information about the IPCS command used to initiate an IPCS session
- *z/OS MVS Initialization and Tuning Reference* for information about creating the BLSCUSER parmlib member

The following table shows examples of the lines needed in the BLSCUSER parmlib member to install any of these exit routines:

Table 11. Installing IPCS Exit Routines in the BLSCUSER Parmlib Member

Exit Routine	Line Added to BLSCUSER	Purpose of Line
ANALYZE	EXIT EP(ANALYZED) ANALYZE	Identifies entry point ANALYZED to IPCS as an ANALYZE exit routine.
Address space control block (ASCB)	EXIT EP(ASCBEXIT) ASCB	Identifies entry point ASCBEXIT to IPCS as a ASCB exit routine.
Control block formatter	DATA STRUCTURE(THAT) FORMAT(FORMATTER)	Identifies control block formatter exit routine FORMATTER with the structure THAT.
Control block status (CBSTAT)	EXIT EP(CBEXIT) CBSTAT(ASCB)	Identifies entry point CBEXIT to IPCS as a CBSTAT exit routine for the ASCB control block.
Find	DATA STRUCTURE(IT) FIND(FINDIT)	Associates find exit routine FINDIT with the structure IT.
Post-formatting	EXIT EP(POSTEXIT) FORMAT(***data_structure***)	Identifies entry point POSTEXIT to IPCS as a post-formatting exit routine which will format the data area ***data_structure***.
Scan	DATA SCAN(SCANIT)	Identifies scan exit routine SCANIT.
Task control block (TCB)	EXIT EP(TCBEXIT) FORMAT(TCB)	Identifies entry point TCBEXIT to IPCS as a TCB exit routine. TCB exit routines are defined as post-formatting exit routines to IPCS.
Verb	EXIT EP(VERBAL) VERB(MYDATA) AMASK(X'***aa***FFFFFF') ABSTRACT('***text***') HELP(***helppanel***)	Identifies entry point VERBAL to IPCS as a verb exit routine. MYDATA is a verb name used with the exit identified with the IPCS VERBEXIT subcommand. For example, by entering VERBEXIT MYDATA, you give verb exit routine VERBAL control. ***aa*** is either 00, to indicate 24-bit storage accessing, or 7F to indicate 31-bit storage accessing. '***text***' is the text of the abstract shown on the component data analysis panel entry associated with this verb exit. ***helppanel*** is the name of the help panel that accompanies this exit routine. The AMASK, ABSTRACT, and HELP parameters are optional.

Chapter 10. IPCS Exit Services

This chapter describes the exit services that give your IPCS exit routine access to the basic services necessary for accessing, formatting, analyzing, printing, or displaying information contained in a dump data set.

Exit Services

The following services are recommended.

Service	Use When You Want to:	Invoked Through:
"Add Symptom Service" on page 99	Generate symptoms from stand-alone dumps, SVC dumps, or ABEND dumps written to SYSMDUMP data sets	Exit services router
"Control Block Formatter Service" on page 101	Format and print a complete data area in one invocation	Exit services router
"Control Block Status (CBSTAT) Service" on page 107	Invoke all CBSTAT exit routines for a requested data area	Exit services router
"Contention Queue Element (CQE) Create Service" on page 108	Create contention queue elements (CQE) that are called by ANALYZE exit routines	Exit services router
"Equate Symbol Service" on page 111	Create a symbol entry in the symbol table	Exit services router
"Exit Control Table (ECT) Service" on page 113	Invoke an exit routine within an exit routine, or invoke a group of exit routines	Exit services router
"Expanded Print Service" on page 116	Display data at a terminal and write data to the IPCS print data set	Exit services router
"Format Model Processor Service" on page 120	Format and print an entire data area using a control block model	Exit services router
"Get Symbol Service" on page 123	• Retrieve symbols from the symbol table • Initialize the BLSRESSY macro for your exit routine	Exit services router
"Locate-Mode SWA Manager" on page 149	Convert 3-byte SWA virtual addresses (SVAs) to addresses and related data	Two ways: • Issue a LINK macro. • Through CALL after the use of a LOAD macro.
"Name Service" on page 124	Describe the data space and/or address space associated with a specified STOKEN	Exit services router
"Name/Token Lookup Service" on page 127	Retrieve the token from a name/token pair	Exit services router
"Select Address Space Identifier (ASID) Service" on page 130	Scan ASCBs in a dump and generate a list of entries for selected address spaces	Exit services router

Service	Use When You Want to:	Invoked Through:
"Standard Print Service" on page 133	Print a line of output. **Note:** Use expanded print service, especially for exit routines that issue messages with prefixes.	Exit services router
"Storage Access Service" on page 135	Access data in a dump and read dump data into storage	Exit services router
"Storage Map Service" on page 137	Process storage map entries and obtain the data they represent	Exit services router
"Symbol Service" on page 140	Process symbols and obtain the data they represent	Exit services router
"Table of Contents Service" on page 146	Add entries to the table of contents	Exit services router
"WHERE Service" on page 147	Determine the system area in which an address resides	Exit services router
"Obtaining Information About Coupling Facility Structures" on page 150	Search for information about coupling facility structures	Issue an IXLZSTR macro
"Obtaining Information About Loaded Modules" on page 150	Search for information about loaded modules	Issue a CSVINFO macro
"Quiesce IPCS Transaction" on page 151	Quiesce a transaction that is currently being processed by IPCS.	Three ways: • From an authorized program, different task: Schedule an IRB. • From an authorized program, same task: Issue a LOAD macro, then a SYNCH macro. • From an unauthorized program, same task: Issue a LINK macro or issue a LOAD macro followed by a call.
"TOD Clock Service" on page 152	Obtain time-of-day (TOD) clock image	Two ways: • Issue a LINK macro. • Issue a LOAD macro followed by a call.
"17-Character Time Stamp Service" on page 154	Obtain 17-character EBCDIC time stamp	Two ways: • Issue a LINK macro. • Issue a LOAD macro followed by a call.
"26-Character Time Stamp Service" on page 157	Obtain 26-character EBCDIC time stamp	Two ways: • Issue a LINK macro. • Issue a LOAD macro followed by a call.

Services Supporting 64-Bit Addresses and Lengths

With OS/390 Release 10 and higher, the following IPCS exit services support the processing of 64-bit addresses and lengths in dump data set information:

- Control block formatter
- Control block status (CBSTAT)
- Contention queue element create (CQE)
- Format model processor
- Storage map
- Symbol
- WHERE

Services Not Recommended

These exit services were provided in systems before MVS/SP Version 3 and are maintained in later systems for compatibility. These exit services are not recommended.

Note: Chapter 12, "IPCS Exit Services Supported for Compatibility," on page 167 describes how to invoke these services directly through pointers in the BLSABDPL exit parameter list. These services cannot be invoked through the exit services router.

The format service is not the same as the control block formatter or the format model processor, which replace it.

Service Maintained	Use When You Want to:	Invoked Through:
"Dump Index Service" on page 167	Print a table of contents for each of the significant parts. Use instead the table of contents service.	Load register 15 with contents of ADPLNDX. Use BALR instruction to branch to address in register 15.
"Format Service" on page 168	Convert data to printable hexadecimal, if necessary, and format data in the output buffer. Use instead the control block formatter or format model processor service.	Obtain the address of the format service from field ADPLFRMT in the BLSABDPL parameter list. Use standard linkage conventions to invoke the service.
"Old Storage Access Service" on page 171	Obtain data from a dump data set. Use instead: • The storage access service, for exit routines that do SNAP or both SNAP and IPCS formatting • The storage map or symbol service, for exit routines that do IPCS formatting	Two ways: • Exit service router. • Obtain the address of the storage access routine from the ADPLMEMA field of the BLSABDPL exit parameter list. Use standard linkage conventions to invoke the service.
"Print Service" on page 173	Print an output line of a dump. Use instead the standard print or expanded print.	Obtain the address of the print service from the ADPLPRNT field of the BLSABDPL exit parameter list. Use standard linkage conventions to invoke the service.
"Summary Dump Data Access Service" on page 174	Access the summary dump data contained in an SVC dump. Use instead the symbol service, with an ACC code identifying the summary dump as desired storage.	Use a CALL macro to pass control to IEAVTFRD, using standard linkage contentions. Register 1 must contain the address of the exit parameter list.

Invoking with the Exit Services Router

Use the exit services router to invoke most of the exit services. The address of the exit services router is in field ADPLSERV, mapped by mapping macro BLSABDPL.

Your exit routine must pass the save area address in register 13.

Your exit routine invokes the requested exit service by calling the exit services router through the CALL macro and by putting the address of a parameter list into register 1. The parameter list must contain the following:

- **The address of the ABDPL:** This parameter establishes addressability to all the fields that your exit routine might want to reference in BLSABDPL.
- **The address of a word containing the exit service code:** This parameter identifies the requested service.
- **The address of the parameter list for the requested service, if applicable:** This parameter establishes addressability to the fields within the requested service parameter list. Some fields must be initialized, other fields are optional. See the description of the individual exit service for the required and optional information.

Table 12 on page 98 lists the service codes with the corresponding exit services.

Table 12. Exit Services, Service Codes, Parameter Lists, and Mapping Macros

Exit Service	Service Code	Exit Service Parameter List	Parameter List Mapping Macro	Valid for ABEND/SNAP Formatting?
Add symptom	ADPLSADS	ADSY	BLSADSY	No
Control block formatter	ADPLSCBF	ADPLPFMT	BLSABDPL	Yes
Control block status (CBSTAT)	ADPLSCBS	CBSP	BLSACBSP	No
Contention queue element (CQE) create	ADPLSCQE	PCQE	BLSAPCQE	No
Exit control table (ECT) exit	ADPLSECT	ADPLPECT	BLSABDPL	No
Equate symbol	ADPLSEQS	BLSRESSY	BLSRESSY	No
Expanded print	ADPLSPR2	PPR2	BLSUPPR2	Yes
Format model processor	ADPLSFMT	ADPLPFMT	BLSABDPL	Yes
Get symbol	ADPLSGTS	BLSRESSY	BLSRESSY	No
Name	ADPLSNAM	NAMP	BLSRNAMP	No
Name/token lookup	ADPLSNTK	NTKP	BLSQNTKP	No
Select address space identifier (ASID)	ADPLSSEL	ADPLPSEL	BLSABDPL	No
Standard print	ADPLSPRT	BLSUPPR2	BLSUPPR2	Yes
Storage access	ADPLSACC	ADPLPACC	BLSABDPL	Yes
Storage map	ADPLSMAP	XMSP	BLSRXMSP	No
Symbol	ADPLSSYM	XSSP	BLSRXSSP	No
Table of contents	ADPLSNDX	None	None	No
WHERE	ADPLSWHS	PWHS	BLSRPWHS	No

The exit services router uses the exit service code to obtain the address of the requested exit service. If ABEND/SNAP formatting does not support the exit service, register 15 contains a return code of zero and field ADPLCODE contains a value of X'04' Otherwise, the exit services router calls the service for your exit routine. After the service has performed its processing, register 15 contains a return code from the requested exit service.

See the following:

- "Format Service" on page 168 for information about the format service that uses patterns
- *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for information about coding the mapping macros listed in Table 12 on page 98
- For macro mappings, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Add Symptom Service

The add symptom service permits exit routines to generate symptoms to stand-alone dumps, SVC dumps, and SYSMDUMP ABEND dumps. The service does the following:

- Provides symptoms
- Makes symptoms available in machine readable format

An IPCS exit routine can provide one symptom or a pair of symptoms on each invocation of the service.

Prior to Release 10, if you are authorized to update the dump data set, you can use the IPCS add symptom service to add secondary symptom strings up to 2048 bytes; the actual length may be fewer bytes, depending on the space available in the dump header record. IPCS creates a literal definition of the symbol SECONDARYSYMPTOMS from the first 256 bytes of the new symptom string. With Release 10, IPCS only records symptom strings in the dump directory. Dump data sets are treated as read-only.

Restrictions

The add symptom service is subject to the following restriction:

- The service is only supported in an IPCS environment. This service is not supported in a SNAP environment.

Requirements

Before invoking this service, your exit routine must place the following information in the BLSADSY mapping macro:

Field
Description

ADSYMP
First symptom passed

ADSYML
Length of first symptom

ADSYMP2
Second symptom passed (if symptoms are paired)

ADSYML2
Length of second symptom (if symptoms are paired)

ADSYNOSV
Bit field in byte ADSYFL1. This bit is turned on by the caller when the symptom is inappropriate for an SVC dump. This bit causes the add symptom service to ignore the symptom when processing an SVC dump. This bit is set to 0 in BLSADSY.

In addition to these requirements, the following considerations should be given to the symptoms provided:

- The symptom must be in the form of KEYWORD/DATA
- Total length cannot exceed 15 characters
- Parameter cannot exceed 8 characters
- Paired symptoms are not the same

The symptoms passed should contain valid RETAIN parameters. See *z/OS Problem Management* for a description of the valid RETAIN parameters. The add symptom service does not check for the validity of the parameters.

Invoking the Service

After setting the required field, your exit routine can invoke the add symptom service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL
- The address of the add symptom service code (ADPLSADS)
- The address of the area mapped by the BLSADSY mapping macro, which is set to describe the symptom(s) being passed

Output

When the add symptom service returns control to your exit routine, register 15 contains one of the following return codes:

Code
Meaning

00
The add symptom service successfully added the symptom(s)

04
The symptoms already exist

12
A user requested attention exit

Example

Figure 13 on page 101 illustrates subroutine ADSYS using the add symptom service to add paired symptoms.

```
*
*The ADDSYMP subroutine calls the add symptom service to
*put paired symptoms
*in section 4, symptom area, of the dump header.
*
*======] Set up for call to add symptom service
ADDSYMP  LA    R15,SYMP1        Load address of 1st symptom into R9
         ST    R15,ADSYMP       Address if 1st symptom
         LA    R15,L'SYMP1      Load length of 1st symptom into R9
         ST    R15,ADSYML       Length of 1st symptom
         LA    R15,SYMP2        Load address of 2nd symptom into R10
         ST    R15,ADSYMP2      Address of 2nd symptom
         LA    R15,L'SYMP2      Load length of 2nd symptom into R10
         ST    R15,ADSYML2      Length of 2nd symptom
         OI    ADSYFL1,ADSYNOSV Do not add symptom on an SVC dump
*======] Call the add symptom service
         L     R15,ADPLSERV     -]Exit services router
         CALL  (15),((R11),REQCODE,ADSY) Call the add symptom ser
         L     R13,SAVEAREA+4   Resume use of input save area
         RETURN (14,12),RC=0
R01      EQU   1                Register 1  - Parameter list address
R11      EQU   11               Register 11 - ABDPL address
R12      EQU   12               Register 12 - TADSYASM base register
R13      EQU   13               Register 13 - Save area address
R15      EQU   15               Register 15 - Entry point address
SYMP1    DC    C'FLDS/CHRID'    First symptom
SYMP2    DC    C'VALU/CBADID'   Second symptom
REQCODE  DC    A(ADPLSADS)      Request is for add symptom service
SAVEAREA DS    18F              Register save area
ADSY     BLSADSY DSECT=NO       Add symptom service parameter list
         BLSABDPL ,             Common parameter list
```

Figure 13. Example - Invoking the Add Symptom Service

Control Block Formatter Service

The control block formatter service formats and prints a complete control block in one invocation. To invoke the service, place required information into the parameter list for the control block formatter service (ADPLPFMT) mapped by the BLSABDPL mapping macro.

The maximum size of the control block is 64 kilobytes.

See *z/OS MVS IPCS Commands* for a list of the control blocks this service supports.

You can use the control block formatter service to format literal data as if it were a valid instance of a control block. IBM does not normally recommend this use. For example, you could ask the service to format a symbolic literal as a task control block (TCB); however, it would be inappropriate to use the formatted “TCB” for diagnosis.

Note that in OS/390 Release 10 and higher, the control block formatter service supports 64-bit addresses and lengths.

Requirements

Prior to invoking this service, your exit routine must place the following information into the ADPLPFMT parameter list:

- Set field ADPLPCHA to contain the requested control block acronym.
- Set field ADPLPVCL to contain the view control to select the individual fields for display from the control block. If you do not specify view control, your exit routine does not receive any output.

  See “View Control” on page 105 for information about specifying the view control.
- If the data is not in a buffer and the data is in an address space, call the storage access service. Set field ADPLPBAV to contain the address of the control block in the dump. Set field ADPLASID to contain the address space identifier. Set fields ADPLPBAS and ADPLPBLS to 0.
- If the data is in a buffer, set field ADPLPBAS to contain the buffer address. Set field ADPLPBLS to contain the length of the dump data if your exit routine has already accessed the requested control

block or if the control block is of variable length. Set field ADPLPBAV to contain the address of the control block in the dump.

Invoking the Service

After setting the required fields, your exit routine can invoke the control block formatter service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL.
- The address of the control block formatter service code (ADPLSCBF).
- The address of the control block formatter service parameter list (ADPLPFMT) mapped by the BLSABDPL mapping macro, which is set to describe the control block being formatted.

  You must obtain storage for or establish addressability to BLSABDPL. Before initializing the fields, remember to set the fields to 0.

  By setting the individual bit strings within field ADPLPFMT of the BLSABDPL mapping macro, you can request formatting of a control block by specifying its acronym in field ADPLPCHA.

  Note: To format a literal value, the third parameter must be the address of the area mapped by the BLSRESSY mapping macro. The BLSRDATS area, which is part of the BLSRESSY area, must refer to the literal value.

For the BLSABDPL mapping macro, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Output

When the control block formatter service returns control to your exit routine, IPCS sets register 15 to contain one of the following return codes:

Code
Meaning

00
The control block formatter service completed normally

04
Attention: Check the following bit string flags in the ADPLPRET field (within BLSABDPL) for additional diagnostic information:

ADPLPRAC
Control block acronym check failed

ADPLPRNL
Unable to load the control block formatting model

ADPLPRNB
Unable to access the control block

ADPLPRNF
Unable to format the control block

ADPLPRTB
Truncated control block

ADPLPRNC
The CBFORMAT service was unable to associate the data type specified with any data type defined to IPCS. This may be the result of the caller spelling the name incorrectly or may reflect the absence of a definition. Definitions are supplied by DATA statements in BLSCECT, BLSCUSER, or related parmlib members read at the start of the IPCS session.

ADPLPRNE
The CBFORMAT service was able to associate the data type specified with a defined data type, but neither formatter nor model is, in turn associated with that data type.

ADPLPRNG
Storage not available for the CBAT

ADPLPRUU
Control block formatter previously marked unusable

ADPLPRIM
Incorrect control block formatting model

ADPLPRCM
Control block formatting model error

ADPLPNVM
No view match and therefore, no output

ADPLPBXI
Identifier on parameter list extension is bad

ADPLPFEF
Formatting exit failure

ADPLPNXD
No exit data

16
The system stopped processing this service because of a lack of sufficient storage for the CBAT

If the control block formatter service completed successfully, your exit routine has a formatted control block.

Note:

1. This service changes the contents of fields ADPLPBAS (buffer address in central storage) and ADPLPBLS (length of the block in central storage). These two fields must be set by your exit routine, otherwise the control block formatting service interprets the nonzero contents of these fields to indicate that the control block is in a buffer.
2. Some formatting programs must perform another dump access to get extensions to the first control block, and in some environments, the same buffer location is reused. The buffer might not contain an image of the first control block. Therefore, your exit routine should do its own dump access if it needs information out of a control block.

Customization

The bits in flag byte ADPLPOPT can be set to customize the control block formatter service:

To Check the Validity of the Requested Acronym

Your exit routine can set bit ADPLPOAC. If the acronym in the model is blank, the bit is ignored. The acronym in the model is compared with the contents of the dump at the offset and length specified in the model. If the comparison fails, a message is issued (unless error messages are suppressed by setting bit ADPLNMSG), no formatting is done, and bit ADPLPRAC is set to indicate that the control block failed the acronym check.

To Suppress the Dump Header

Your exit routine can set bit ADPLPSDH. The header consists of the acronym or control block label from the model and the dump address.

To Suppress the Dump Data Offsets

Your exit routine can set bit ADPLPSOF. The offset of the first data item on each line is usually printed at the left of the formatted output. Your exit routine should suppress the dump data offsets if the fields are not in order or if you specified a view control resulting in a subset of the fields being formatted.

To Print the Dump Address

Your exit routine can set bit ADPLPPDA. This causes a display of the dump address of the data (instead of an offset) with the first line of formatted output. The ADPLPPDA bit and suppressing the header can be used together to obtain a compact display of small control blocks.

To Request Line Mode

Your exit routine can set bit ADPLPOLM. The format model processing service gives control to the routine whose address is specified in field ADPLPLME after each line is formatted and before the line is printed.

Any field entry in the model can be marked to cause the model processor to call a routine specified by ADPLPLME when that field is processed. A parameter list is passed that specifies the address of the ABDPL and the address of a data area containing information pertinent to the formatting process.

Through the use of this facility, your exit routine can inspect the print buffer contents for information or modification, before printing the buffer contents.

If ADPLPOLM (line mode bit) is set, and ADPLPLME contains a nonzero value, control is given to that program after each line is formatted into the print buffer, regardless of the control bits in the model.

To Suppress Messages that Indicate Truncation Has Occurred

Your exit routine can set bit ADPLPSTM. The length of the control block, as indicated by ADPLPBLS, is usually added to the location in field ADPLPBAS. This service uses the resulting address to limit the scope of storage referenced by the model processor. If the model directs the processor to exceed that address, processing of that field is inhibited and a message is issued. Setting the ADPLPSTM bit suppresses that message. This bit setting is useful for formatting a data area of arbitrary length in hexadecimal dump format.

Use fields ADPLPBLC, ADPLPDAC, ADPLPOSI, ADPLPDL1, ADPLPDL2, ADPLPDU1, and ADPLPDU2 to tailor the output from a general model.

To Print Blank Lines

Your exit routine can set field ADPLPBLC to a requested number (including 0).

When this field is nonzero, the service prints the specified number of blank lines before the control block is formatted. If no formatted lines are generated, the service does not print any blank lines.

To Format a Dynamic Array

Your exit routine must set field ADPLPDAC to the number of entries in the dynamic array, unless the array count is in the model.

To Change the Starting Offset of Dump Data

Your exit routine can set field ADPLPOSI to a requested offset.

The service adds the signed value in this field to the offset normally generated.

Setting ADPLPOSI is useful for mapped data areas imbedded in control blocks at an offset other than zero.

To Control the Number of Formatted Entries for an Array

Your exit routine can set these fields: ADPLPDL1 (the lower limit of the first dimension), ADPLPDU1 (the upper limit of the first dimension), ADPLPDL2 (the lower limit of the second dimension), and ADPLPDU2 (the upper limit of the second dimension). If one of these limits is coded with an * in the BLSQMFLD macro, your exit routine must provide the dimension. Setting these fields permits the processing of a large array to be displayed in pieces.

To Communicate Additional Information to the Formatting Service

The ADPLPFXT data area maps the format parameter extension. This data area is used by exit routines to communicate additional information to the formatting service. It is pointed to from the ADPLPEXP field in the format parameter. It can be used together with current models if only one data area is described. The advantage of its use is the ability to describe a data area in terms of an equate symbol record, and the availability of additional formatting specifications.

It must be used together with a multiple source model, and the areas described must match with the source index values in the model.

Example

Figure 14 on page 105 illustrates the subroutine FORMAT using the IPCS-supplied control block formatter service to format the PSA using the IBM-supplied format model.

```
*--------------------------------------------------------------------
*The FORMAT subroutine calls the control block formatter service
*to format the PSA using the IBM-supplied formatter model.
*ABDPLPTR is a pointer to the DSECT created by the second invocation
*of BLSABDPL.  ABDPLPTR is stored in register 11 to preserve the pointer
*across the call to the control block formatter service.
*ADPLPFMT refers to the DS set up by the first invocation of BLSABDPL.
*--------------------------------------------------------------------
FORMAT   MVI   ADPLPOPT,ADPLPOAC  Check acronym
         MVC   ADPLPCHA,=CL8'PSA' Name of control block to format
         L     R15,ADPLSERV       Load address of exit services router
         CALL  (15),((ABDPLPTR),CODECBF,ADPLPFMT)  Invoke service
         LTR   R15,R15            Was the PSA displayed?
         BNZ   EXIT               No.  End processing
CODECBF  DC    A(ADPLSCBF)        Control block formatter code
ABDPLPTR EQU   11                 General register 11
*--------------------------------------------------------------------
*        Reserve space for an initialized control block formatter
*        service parameter list.
*        The view control field is set to X'0300'.
*--------------------------------------------------------------------
         BLSABDPL DSECT=NO,AMDEXIT=NO,AMDOSEL=NO,                   *
               AMDPACC=NO,AMDPFMT=YES,AMDPECT=NO,AMDPSEL=NO
*--------------------------------------------------------------------
*        Define the format of the ABDPL addressed by R1 on input
*--------------------------------------------------------------------
         BLSABDPL DSECT=YES,AMDEXIT=YES,AMDOSEL=NO,                  *
               AMDPACC=NO,AMDPFMT=NO,AMDPECT=NO,AMDPSEL=NO
```

Figure 14. Example - Invoking the Control Block Formatter Service

View Control

Use the view control bits in field ADPLPVCL to enable your exit routine to provide multiple levels of detail in the formatting of a control block with only one control block model. Set various view control bits in your exit routine to control the formatting of a requested control block.

The IPCS-supplied control block model contains a 16 bit view control field for each field in the control block. The view control field is divided into two sections, a general view of 12 bits and a component view of 4 bits. The format model processor compares the views specified by the model and your exit routine to determine if a field should be formatted or not. At least one bit in the general section must match, and if the model view component section is not all zero, at least one component section bit must also match.

When VIEWMATCH=VALUE is coded on the BLSQMDEF macro, a bit is set in the model header that causes the model processor to perform view matching in a different manner. In this mode, the first byte of the view control fields in the model and in the format parameter list must match exactly in order to display the corresponding field of data. The rules for a match in the component portion of the view are the same as for bit matching.

Table 13 on page 106 lists the general view control bits, their hexadecimal settings, and their usage conventions.

Table 13. View Control Bits and Their Recommended Meaning

Setting	Name	Recommended Meaning
X'8000'	ADPLPKEY	Exhibits/inhibits key fields of a defined control block (as defined by the KEYFIELD parameter)
X'4000'	ADPLPSUM	Exhibits/inhibits the summary fields
X'2000'	ADPLPREG	Exhibits/inhibits the register save area (The 16 general purpose registers are formatted as four lines of four words with a leading caption)
X'1000'	ADPLPLIN	Exhibits/inhibits the linkage fields
X'0800'	ADPLPEFD	Exhibits/inhibits the error indicating fields
X'0400'	ADPLPHEX*	Exhibits/inhibits the data in a hexadecimal dump format
X'0200'	ADPLPNOR	Exhibits/inhibits the non-reserved (defined) fields
X'0100'	ADPLPRES	Exhibits/inhibits the reserved fields
X'0080'	ADPLPSTA	Exhibits/inhibits arrays in the control block with a static dimension
X'0080'	ADPLPDCD	Exhibits/inhibits decoding of flag fields
X'0040'	ADPLPDYN	Exhibits/inhibits arrays in the control block with a dynamic dimension
X'0020'	ADPLPINP	Exhibits/inhibits input fields in a two-way communication area
X'0010'	ADPLPOUT	Exhibits/inhibits output fields in a two-way communication area
X'0008'	Component use	
X'0004'	Component use	
X'0002'	Component use	
X'0001'	Component use	

**Note: ***ADPLPHEX is static. You will always get a hexadecimal dump format if you specify this view control bit.

If you want your exit routine to format a hexadecimal dump of the requested control block, set field ADPLPVCL to X'0400' by setting bit ADPLPHEX. If you want your exit routine to format a summary of the requested control block and the register save area, set field ADPLPVCL to X'6000' by setting bits ADPLPSUM and ADPLPREG respectively. To completely format a control block, your exit routine should set both ADPLPNOR and ADPLPRES bits, thereby setting field ADPLPVCL to X'0300'.

The 4 bits in the component section are named ADPLPCV1 through ADPLPCV4 and have no assigned significance. These are used by formatting modules associated with a model and can be used as needed.

The component view bits work in a slightly different manner from the general view bits. If a field in the model has any component view bit on, then there must be a match of at least one general view bit and at least one component view bit in order for the field to be displayed.

An example of the output that the view control can produce follows. These examples do not reflect dump data. Only the fields for the specified view control are shown. If your exit routine wanted to format the key fields of the ASCB, you would call the control block formatter service and provide:

- The control block acronym, ASCB

- The view control X'8000'
- The address of the ASCB in field ADPLBAV (0CBADD00 is a fictitious address)

Figure 15 on page 107 illustrates the report that would be produced:

```
ASCB: 0CBADD00
   +0004  FWDP..... 00000000  ASID..... 0000      CSCB..... 00000000
   +003C  TSB...... 00000000  AFFN..... 0000      ASXB..... 00000000
   +0072  DSP1..... 00        FLG2..... 00        SRBS..... 0000
   +0080  LOCK..... 00000000  ASSB..... 00000000
```

Figure 15. Example - View Control Displaying Key Fields

Or if your exit routine wanted to format the reserved fields of the ASCB, you would call the control block formatter service and provide the same information as previously stated except the view control would be X'0100'.

Figure 16 on page 107 illustrates the report that would be produced:

```
ASCB: 0CBADD00
   +0029  R029..... 00        R02C..... 00000000  R035..... 000000
   +0074  RSV...... 0000      R09B..... 00        R121..... 000000
   +0158  R158..... 00000000  00000000  00000000  00000000
   +0170  R170..... 00000000
```

Figure 16. Example - View Control Displaying Reserved Fields

Control Block Status (CBSTAT) Service

The control block status (CBSTAT) service invokes all CBSTAT exit routines for a requested control block. New IPCS exit routines can be defined in IPCS parmlib member BLSCUSER. The exit routines must generate output by using the expanded print service.

The callers of the CB status service must fill in parameter list (BLSACBSP) with information describing the request for control block status. If any errors are detected in the CBSTAT parameter list, IPCS issues message BLS01042I.

Note that in OS/390 Release 10 and higher, the CBSTAT service supports 64-bit addresses and lengths.

Requirements

Prior to invoking this service, your exit routine must set field ESSYSYM in the BLSRESSY mapping macro to contain the equated symbol. If the get symbol service processed successfully, then your exit routine has a completely initialized BLSRESSY macro that defines the requested symbol.

Invoking the Service

After setting the required field, your exit routine can invoke the CBSTAT service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL
- The address of the CBSTAT service code (ADPLSCBS)
- The address of the area mapped by the BLSACBSP mapping macro, which is set to describe the control block being analyzed

Output

When the CBSTAT service returns control to your exit routine, register 15 contains one of the following return codes:

Code
Meaning

00
The CBSTAT service completed normally.

04
The symbol was created but an attention condition was detected. Preceding this return code, there are informational messages that provide additional diagnostic aid.

08
The symbol was created but an error condition was detected. Preceding this return code, there are informational messages that provide additional diagnostic aid.

12
No symbol was created. The attention interrupt key might have been pressed.

16
An error in the IPCS environment was encountered.

Example

Figure 17 on page 108 illustrates the subroutine CBSTATUS setting up the CBSTAT parameter list (CBSP) and calling the CBSTAT service to invoke all CBSTAT exit routines.

```
*
*The CBSTATUS subroutine sets up the CBSTAT parameter
*list (CBSP) and calls the CBSTAT service to invoke
*all CBSTAT exit routines.
*
*======  Set up for call to CBSTAT service
CBSTATUS MVC   CBSPD,ESSYD      Data description
         MVC   CBSPAS,ESSYAS    Address space description
         MVC   CBSPLAD,ESSYLAD  Address of control block
*======  Call the CBSTAT service
         L     R15,ADPLSERV     - Exit services router
         CALL  (15),((R11),REQCODE,CBSP) Call the CBSTAT se
R01      EQU   1                Register 1  - Parameter list address
R11      EQU   11               Register 11 - ABDPL address
R15      EQU   15               Register 15 - Entry point address
REQCODE  DC    A(ADPLSCBS)
CBSP     BLSACBSP DSECT=NO      CBSTAT service parameter list
         BLSABDPL ,             Common parameter list
```

Figure 17. Example - Invoking the CBSTAT Service

Contention Queue Element (CQE) Create Service

ANALYZE exit routines call the contention queue element (CQE) create service to create CQEs. Each time an ANALYZE exit routine calls the CQE create service, IPCS creates one CQE. The ANALYZE exit routine fills in the CQE create service parameter list (PCQE mapped by the BLSAPCQE mapping macro), with information relating to a unit of work that holds or is waiting for a resource. The exit routine then passes the PCQE to the CQE create service by calling the exit services router with a router value of ADPLSCQE.

See “ANALYZE Exit Routine” on page 57 for a description of ANALYZE exit routines.

Requirements

Prior to invoking this service, your exit routine must place information into BLSAPCQE. There are two methods that you can use:

- If you are writing non-reentrant code, you can issue the BLSAPCQE mapping macro using the DSECT=NO parameter. This method is used in Figure 18 on page 111, where PCQESYNM and PCQEJOBN are not explicitly provided, but are filled in because DSECT=NO is specified on BLSAPCQE at label PCQE.
- If you are writing reentrant code, you can issue BLSAPCQE two times, once specifying DSECT=YES to describe a dynamic parameter list, and once specifying DSECT=NO to create an initialized image. Before invoking the service, copy required information from the initialized image to the dynamic parameter list.

Field
Description

RSA
A pointer to a buffer containing the resource name.

ADA
A pointer to additional data for owners of a resource. The additional data is optional. If additional data is provided, it should contain data relevant to debugging.

RSL
The length of the resource name.

ADL
The length of any additional data.

SYNM
The system name (SYSNAME) where the unit of work is running. This must be provided when the resource has cross system contention. The name of the dumped system is in CVTSNAME. If a SYSNAME is different from CVTSNAME, no attempt is made to access the current dump on behalf of this request; for example, the ASCB and TCB are not accessed to produce further status. The SYSNAME should be left justified and padded with blanks.

EAS
An instance of the BLSRDATS macro within BLSAPCQE. Specify in BLSRDATS the following:

- **Field AS1:** The central processor address, which is used to uniquely identify active system request blocks (SRBs).
  - CPU(n)
  - NOCPU

  For CPU(n), specify X'00000000' for processor 0, X'00000001', for processor 1, and so forth.

  If the unit of work is not associated with a particular processor, specify the value ZZZAS1NO to indicate NOCPU.
- **Field AS2:** The address space identifier (ASID) of the unit of work: ASID(n).

DTD
DTD in PCQED indicates the type of work being performed (such as TCB, SRB, or any name up to 31 characters). This control block name is used in the analysis process to call the CBSTAT service to extract more information about a unit of work. In order to conform to the IPCS standard for structure names, the name should start with an alphabetic character and contain only alphanumeric characters.

LAD
The address of the control block.

OW
Indicates whether the unit of work owns (C‘O ’) or is waiting (C‘W ’) for this resource.

JOBN
An optional 8-character JOBNAME. This should be filled in by exit routines that are producing contention entries that describe contention caused by another system. This must be done because the ASCB for another system is not available in the dump. The job name should also be provided when the work being performed has a special relationship to the address space, as in the case of SRBs that are performing a service and always run in an address space such as MASTER.

If the job name has not been saved in an easily accessible control block, it is recommended that you let the ANALYZE exit routine find it in the ASCB. The ANALYZE exit routine can determine the job name from the ASID.

Initially, the exit routine sets PEQEJOBN to zero. When the exit routine fills in the field, the job name should be left-justified and padded with blanks.

MODN
An optional 8-byte field that contains the CSECT name of the module calling the CQE create service. This field is used in error messages to identify the caller that passed a bad parameter list.

If an incorrect PCQE is passed to the CQE create service, IPCS issues message BLS01001I to help diagnose the problem.

Invoking the Service

After setting the required field, your exit routine can invoke the CQE create service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL
- The address of the CQE create service code (ADPLSCQE)
- The address of the area mapped by the BLSAPCQE mapping macro, which is set with the contention information

Output

The contention queue service returns these standard IPCS return codes to the caller:

Code
Meaning

00
The CQE create service completed normally.

04
The CQE was not created after an attention condition was detected. Preceding this return code, there are informational messages that provide additional diagnostic aid.

08
The symbol was not created after an error condition was detected. Preceding this return code, there are informational messages that provide additional diagnostic aid.

12
No CQE was created. The attention interrupt key might have been pressed.

16
An error in the IPCS environment was encountered. If the exit routine passed an incorrect PCQE, IPCS issues message BLS01001I to help you diagnose the problem with the parameter list.

A return code greater than 00 from the CQE create service should cause the ANALYZE exit routine to end processing.

When control is returned to the exit routine, IPCS will create a CQE, add it to the contention queue, and save it in the IPCS dump directory data set.

An exit routine can call the contention queue service as many times as is necessary for a given contention situation.

Example

Figure 18 on page 111 illustrates the subroutine ANLZ using the get symbol service to obtain a definition of master scheduler's ASCB. ANLZ builds the CQE create service parameter list to define master scheduler's ASCB as waiting for **RESOURCE NUMBER 1**.

```
*======] Invoke Get Symbol Service for ASCB
ANLZ     MVC   ESSYSYM(31),=CL31'ASCB00001' Symbol
         MVI   ESSYDTY,ZZZDTYM    Structure
         MVC   ESSYDTD(31),=CL31'ASCB' Data name
         L     R15,ADPLSERV       -]Exit services router
         CALL  (15),((R11),CODEGTS,ESSY) Invoke Get Symbol service
         CH    R15,=H'12'         Was get symbol service successful?
         BNL   EXIT               No. End processing
*======] Set up for call to CQE Create service
         SPACE 1                  Begin dump formatting
         MVC   PCQEMODN,MODNAME   Module name for diagnostics
         LA    R02,RESNAME        Address of resource name
         ST    R02,PCQERSA        Is placed in the parameter list
         LA    R02,RESLEN(0)      Length of resource name
         STH   R02,PCQERSL        Is placed in the parameter list
         LA    R02,ADDDATA        Address of additional data
         ST    R02,PCQEADA        Is placed in the parameter list
         LA    R02,ADDLEN(0)      Length of additional data
         STH   R02,PCQEADL        Is placed in the parameter list
*======] Define the control block which represents the unit of work
         MVC   PCQEAS,ESSYAS      Address space description from ESSY
         MVC   PCQELAD,ESSYLAD    Address of control block
         MVC   PCQED,ESSYD        Data characteristics from ESSY
         MVC   PCQEOW,WAITER      Indicate unit of work is waiting
*======] Call the CQE Create service
         SPACE 1                  Begin standard module epilogue
         L     R15,ADPLSERV       -]Exit services router
         CALL  (15),((R11),REQCODE,PCQE) Call the CQE Create service
R11      EQU   11                 Register 11 - ABDPL address
REQCODE  DC    A(ADPLSCQE)
CODEGTS  DC    A(ADPLSGTS)        Get symbol service code
RESNAME  DC    CL20'RESOURCE NUMBER 1'  Resource name
RESLEN   EQU   *-RESNAME          Resource name length
ADDDATA  DC    CL20'ADDITIONAL DATA  '  Additional data
ADDLEN   EQU   *-ADDDATA          Additional data length
WAITER   DC    CL2'W '            Waiting for resource indicator
MODNAME  DC    CL8'TCQECASM'      Module name for diagnostics
SAVEAREA DS    18F'0'             Register save area
PCQE     BLSAPCQE DSECT=NO        CQE Create service parameter list
ESSY     BLSRESSY DSECT=NO        IPCS ES record buffer
         BLSABDPL ,               Common parameter list
```

Figure 18. Example - Invoking the CQE Create Service

Note: In order to run this code properly, it must be placed in a module which is defined as an ANALYZE exit routine in the BLSCUSER parmlib member, or in a parmlib member imbedded in the BLSCUSER member.

See Chapter 9, “Installing IPCS Exit Routines,” on page 93 for information about installing IPCS exit routines.

Equate Symbol Service

The equate symbol service stores a symbol entry in the symbol table.

By maintaining this symbol table, you can reduce the access time for subsequent references to the same control block or data area. Each symbol entry can consist of associated attributes, such as the address and the length. If a symbol for the processing entry does not exist, the symbol entry is added to the symbol table. However, if a symbol already exists in the symbol table for the processing entry, the new definition overlays the existing one.

If the symbol is a literal, the equate symbol service derives the symbol from the definition of another literal symbol. For example, your exit routine issued the following subcommand to retrieve the definition of the literal:

```
literal astring c'ABC'
```

Use the get symbol service to retrieve the symbol ABC from the symbol table and place it in the BLSRESSY area. The definition in the BLSRESSY area is:

```
ESSYSYM   CL31'ASTRING'
ESSYAST   C'LI'          (symbol ZZZASTLI)
```

```
ESSYAS1   F'n1'     (an arbitrary number assigned by IPCS)
ESSYDLE   F'3'
```

Your exit routine changes the definition to the following:

```
ESSYSYM   CL31'BSTRING'
ESSYDLE   F'2'
```

Your exit routine calls the equate symbol service and receives a return code of 0. Symbol BSTRING is now defined in the symbol table in your dump directory as though IPCS had processed the following subcommand:

```
literal bstring c'AB'
```

In addition, the equate symbol service customized BLSRESSY to match the definition in the symbol table by changing the following:

```
ESSYAS1   F'n1'     (an arbitrary number assigned by IPCS)
```

Symbol BSTRING describes the first 2 bytes of a LITERAL(n_2) address space. The first 2 bytes are occupied by C'AB'. None of the remaining bytes of the address space are available.

Requirements

Prior to invoking this service, your exit routine must:

1. Prepare a definition of the requested symbol in a block of storage whose format is described by mapping macro BLSRESSY. Set all fields, including reserved fields, except the ESSYRDX field.

   Use one of these methods to prepare a definition:

   - Place blanks in field ESSYSYM within the buffer in which you compose your definition. Pass that buffer to the get symbol service to initialize the buffer.
   - Invoke macro BLSRESSY in a CSECT to generate an initialized block of storage. Copy that block into the buffer in which you compose your definition.

     For a literal, use the BLSRDATS area in the BLSRESSY area to identify LITERAL(n), which is an address space that contains the literal from which the new definition is to be derived. The address field LAD and the fields in the BLSRDATS area indicate the part of the literal to be included in the derived definition. Using the values in the areas, the caller of the service can specify all or part of the actual literal value.
   - Use the get symbol service to retrieve a definition of another block into the buffer in which you compose your definition. Check that the fields you expect to use are unchanged as part of your definition.
2. Change only those fields that are needed to define the symbol, address space, address, and attributes that you wish to associate with a block of storage.

Invoking the Service

After setting the required fields, your exit routine can invoke the equate symbol service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL
- The address of the equate symbol service code (ADPLSEQU)
- The address of the area mapped by the BLSRESSY mapping macro, which is set to the symbol information

Output

When the equate symbol service returns control to your exit routine, register 15 contains one of the following return codes:

Code
Meaning

00
The equate symbol service completed normally.

04
The symbol was created but an attention condition was detected. Preceding this return code, there are informational messages that provide additional diagnostic aid.

08
The symbol was created but an error condition was detected. Preceding this return code, there are informational messages that provide additional diagnostic aid.

12
No symbol was created. The attention interrupt key might have been pressed.

16
An error in the IPCS environment was encountered.

If the equate symbol service was storing a literal, the service updates the area mapped by the BLSRESSY mapping macro with a description of the derived symbol.

Example

Figure 19 on page 113 illustrates the subroutine EQUATE using the equate symbol service to store symbol MYASCB in the IPCS symbol table. The subroutine uses the same definition of ASCB00001 that IPCS returns after processing the get symbol service.

```
*
*After processing the get symbol service
*(as described in
Figure 23 on page 124),
*the EQUATE subroutine calls the equate symbol service to
*store symbol MYASCB in the IPCS symbol table,
*using the same definition of ASCB00001.
*
EQUATE   MVC   ESSYSYM(31),=CL31'MYASCB' Symbol
         L     R15,ADPLSERV     Load address of exit services router
         CALL  (15),((ABDPLPTR),CODEEQS,ESSY) Invoke service
         CH    R15,=H'12'       ASCB located?
         BNL   EXIT             No.  Quit
CODEEQS  DC    A(ADPLSEQS)      Equate symbol service code
ABDPLPTR EQU   11               General register 11
ESSY     BLSRESSY DSECT=NO      IPCS ES record buffer
*-----------------------------------------------------------------
*        Define the format of the ABDPL addressed by R1 on input
*-----------------------------------------------------------------
         BLSABDPL
```

Figure 19. Example - Invoking the Equate Symbol Service

Exit Control Table (ECT) Service

The exit control table (ECT) service allows you to invoke an exit routine within an exit routine or to invoke a group of exit routines. You might use this service to process a verb exit routine by supplying either the exit verb name (such as CVTMAP) or the exit verb module name.

The ECT service uses the local defaults of your exit routine when giving control to an exit routine, subcommand, or command. For example, your exit routine, MYPGMA, received control as follows:

```
setdef terminal noprint
verbexit mypgma noterminal print
```

When MYPGMA invokes the ECT service, the local defaults are NOTERMINAL and PRINT. When the ECT service invokes exit routines or issues subcommands or commands, the local defaults are also NOTERMINAL and PRINT. Note that, in the IPCS of systems prior to MVS/ESA SP 5.2, the subcommands or commands instead use the TERMINAL and NOPRINT defaults established by the SETDEF subcommand.

Requirements

Prior to invoking this service, your exit routine must place the following information into BLSABDPL:

- If your exit routine wants to invoke an exit verb name, set field ADPLPEFG to contain zeros, and set field ADPLPEVB to contain the requested verb name, such as ASMDATA.
- If your exit routine wants to invoke a particular type of exit, set the appropriate bits in field ADPLPEFG. For example, set bit ADPLPEPN to invoke the print nucleus exit.
- If your exit routine wants to execute an IPCS subcommand or an IPCS command procedure, set the ADPLPESC bit in the ADPLPECT parameter list and set ADPLPEPL to the address of a standard TSO command buffer that contains the text to be processed. These fields are described in the BLSABDPL mapping macro. (See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Invoking the Service

After setting the required fields, your exit routine can invoke the ECT service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL.
- The address of the ECT service code (ADPLSECT).
- The address of the ECT service parameter list (ADPLPECT) mapped by the BLSABDPL mapping macro. You must obtain storage for or establish addressability to the ECT service parameter list. Before initializing the fields, remember to set the fields to 0.

Field ADPLPECT in data area BLSABDPL maps the ECT parameter list. If you want your exit routine to request a type of exit, set the individual bit strings within field ADPLPEFG. For example, to process the TCB exit routines, set the ADPLPETB bit to 1, and set the ABPLCBP field to contain the address of the block. To process the ASCB exit routines, set the ADPLPEAS bit to 1, and set the ADPLCBP field to contain the address of the block. Set field ADPLPEFG to contain zeros. Set field ADPLPEVB to contain one of the IBM-supplied verb names or its corresponding module name.

See the following:

- For a mapping of the BLSABDPL data area, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).
- *z/OS MVS IPCS Commands* for the VERBEXIT subcommand and the list of verb names and module names that you can specify

Output

When the ECT service returns control to your exit routine, register 15 contains one of the following return codes:

Code
Meaning

00
The ECT service completed normally.

04
Attention: Check the following bit string flags in the ADPLPERR field (within mapping macro BLSABDPL) for additional diagnostic information.

ADPLPEST
Insufficient storage available

ADPLPENV
Verb name was not found in the ECT

ADPLPELI
LINK macro failed with an X'806' ABEND

ADPLPENE
No ESTAE recovery environment established

16
There was no ESTAE recovery routine.

If the ECT service completed successfully, your exit routine has access to the requested exit.

Example

Figure 20 on page 116 illustrates the subroutine ECT using the IPCS-supplied ECT service to invoke the MTRACE verb exit routine. When your exit routine invokes another exit for which parameters are defined, your exit routine can specify these parameters as follows:

- Set ADPLOPTR to the address of the character string containing the parameter.
- Set ADPLOPLEN to the length of the character string.

```
*
*The ECT subroutine calls the ECT service to
*display master trace data using the IBM-supplied MTRACE verb.
*
TSTSECT  TITLE 'TSTSECT--ECT Service Usage Example'
TSTSECT  START 0                  Sample dump processing exit
         SPACE 2                  Begin standard module prolog
*******  Standard module prolog
         SPACE 1                  Begin standard module prolog
         SAVE  (14,12),T,*        Save registers
         LR    R12,R15            Base register for TSTSECT
         USING TSTSECT,R12        Base register for TSTSECT
         LA    R15,SAVEAREA       Address of local save area
         ST    R13,SAVEAREA+4     Chain input save area to local
         ST    R15,8(,R13)        Chain local save area to input
         LR    R13,R15            Establish use of local save area
         LR    ABDPLPTR,R01       Base register for ABDPL
         USING ABDPL,ABDPLPTR     Base register for ABDPL
         SPACE 2                  Begin dump formatting
*******  Use the ECT service to invoke the MTRACE verb exit
         SPACE 1                  Begin dump formatting
         MVC   ADPLPEVB,=CL8'MTRACE' Verb name
         L     R15,ADPLSERV       - Services router
         CALL  (15),((ABDPLPTR),ECT#SERVICE,ADPLPECT) Invoke the exit
         SPACE 2                  Begin standard module epilogue
*******  Standard module epilogue
         SPACE 1                  Begin standard module epilogue
         L     R13,SAVEAREA+4     Resume use of input save area
         RETURN (14,12),T,RC=(15) Restore registers and return
         SPACE 2                  Begin data definitions
*******  Define data
         SPACE 1                  Begin data definitions
R00      EQU   0                  Register 0
R01      EQU   1                  Register 1
ABDPLPTR EQU   11                 ABDPL base register
R12      EQU   12                 Register 12
R13      EQU   13                 Register 13
R14      EQU   14                 Register 14
R15      EQU   15                 Register 15
SAVEAREA DC    18F'0'             Register save area
ECT#SERVICE DC A(ADPLSECT)        ECT service request code
         LTORG ,                  Literal pool
         BLSABDPL DSECT=NO,AMDPECT=YES,AMDEXIT=NO,AMDPACC=NO,      *
               AMDPFMT=NO,AMDPSEL=NO,AMDOSEL=NO
         BLSABDPL DSECT=YES,AMDPECT=NO,AMDEXIT=YES,AMDPACC=NO,     *
               AMDPFMT=NO,AMDPSEL=NO,AMDOSEL=NO
         END   TSTSECT            Test dump formatting exit
```

Figure 20. Example - Invoking the ECT Service

Expanded Print Service

The expanded print service provides a means for exit routines to write data to both the terminal and the IPCS print data set, IPCSPRNT. The expanded print service differs from the standard print service, in that it requires a parameter list PPR2 (mapped by BLSUPPR2) to be passed that describes which new print functions are to be used. The expanded print service provides the following functions:

Conditional headings

Expanded print service saves a predetermined heading and writes it only if some future action occurs that calls the service to print data.

If the expanded print service is called to define a conditional heading when a conditional heading is already in place, the previous conditional heading is written out. 250 characters is the maximum size of a conditional header.

A flag in the PPR2 allows the user to request the cancelation of a conditional header.

For this call, the expanded print service sets a return code of 0 to indicate that the header was canceled and a return code of 4 to indicate that the conditional header was already written out. This allows the program establishing the conditional header to know whether any data was written following the conditional header.

A token field (PPR2TOKN) is provided to allow the user to identify a specific conditional header for cancelation.

If the PPR2TOKN field does not match the token saved at the time the conditional header was saved, the conditional header is not canceled. If the PPR2TOKN field is all blanks, any conditional header is canceled.

Indentation
The expanded print service uses the ADPLSCOL field to determine the number of spaces to indent the output. This function allows a service to generate output that appears in multiple reports at varying indentation levels without requiring any extra coding.

Print buffer
The user must specify the address of a print buffer that contains the data to be printed. The expanded print service requires the length of the print buffer. Specifying the address of a print buffer removes the burden of formatting and requesting multiple prints for data that is considered one logical line. The user of the expanded print service can specify the address of the print buffer in the print service parameter list.

Truncation avoidance
The expanded print service uses the recommended line width to break up the data, in the print buffer, into sections that fit within the recommended line width. As part of this function, the caller can specify the indentation level to be applied to all overflow lines. This allows generation of reports with consistent indentation.

Controlled truncation
The user can request truncating the line being printed at the recommended line width. This truncation occurs after all requested indentation has been applied to the output buffer.

Message support
The user can specify that the print buffer contains a message.

When this occurs, the message identifier is assumed to occupy all positions in the print buffer, up to the first blank. The expanded print service examines the user profile table (UPT) and either removes the message identifier (PROFILE NOMSGID) or leaves the identifier in the message (PROFILE MSGID). All truncation and indentation rules apply to messages.

New line support
The expanded print service recognizes the EBCDIC new line character (X'15') when requested by setting the new line flag on in the PPR2. The new line character causes the preceding data to be printed and the following data to be started on a new line.

Terminal only support
When OPTIONS(TERM) is specified on the BLSUPPR2 expansion, the print request is only sent to the terminal.

Print Line Width
When print line width is specified, the print request uses the current print data set line width as the criteria for where to split a line of text. This action allows old formatter exit routines to preserve their old output format while using the new expanded print service.

Requirements

Before invoking this service, your exit routine must set the desired fields in BLSUPPR2.

Field
Description

PPR2PFL1
Contains the following flags:

Flag
Description

PPR2CNH
A conditional heading has been placed in the output buffer.

PPR2OCOL
The data is to be indented by the number of spaces specified by ADPLSCOL. If this flag is on when the conditional heading is requested, the conditional heading is saved with the current indentation taken from ADPLSCOL. An ADPLSCOL value of zero means the data starts in column one.

PPR2CCNH
The conditional heading request should be canceled.

PPR2TRUN
The data in the print buffer should be truncated, so as to fit on 1 output line.

PPR2MSG
The data in the print buffer is a message with a message identifier that should be treated according to the rules of PROFILE MSGID/NOMSGID.

PPR2EJEC
A page eject is performed when the first line of output is written out.

PPR2NL
New line characters (X'15') are to cause the printing of any prior text and the start of a new line.

PPR2TERM
The output is only written to the terminal for IPCS.

PPR2PLW
The output width is taken from the print data set LRECL.

PPR2BUF
This field contains the address of a print buffer to be used by the expanded print service. The user can specify the address from ADPLBUF or user storage in this field.

PPR2BUFL
This field contains the length of the data to be printed.

PPR2OVIN
The overflow lines indentation level. This field is defaulted to 2.

PPR2TOKN
An 8-character user-specified token used to identify a conditional header. The token is used only to cancel a conditional header request.

PPR2MODN
An 8-character field that contains the name of the module that is calling the expanded print service. This name is used to issue any error messages about the print request.

Invoking the Service

After setting the required field, your exit routine can invoke the expanded print service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL
- The address of the expanded print service code (ADPLSPR2)
- The address of the PPR2 area mapped by the BLSUPPR2 mapping macro, which is set to describe the data to be printed

Output

When the expanded print service returns control to your exit routine, register 15 contains one of the following return codes:

Code
Meaning

00
The expanded print service completed normally and the requested function was performed.

04
The cancelation of a conditional header was requested, but no conditional header existed.

12
An attention interrupt occurred during print processing. This return code occurs only in the IPCS environment.

16
A bad input parameter list (PPR2) was passed to the expanded print service. Message BLS21100I explains the error that was identified.

Information on the Expanded Print Service

The following list describes how the expanded print service functions for various parameter specifications:

- The overflow indentation level is added to ADPLSCOL (when ADPLSCOL has been activated) to determine the actual starting point of overflow data.
- If the requested indentation exceeds half the size of the recommended line width, indentation stops at the halfway point and overflow indentation stops at the halfway point plus 2.
- The expanded print service truncates all blanks from the end of the print buffer. For example, if the buffer contains 20 characters of data followed by blanks and the specified length is 132, only the 20 characters are written.
- In the processing of a long print buffer, if an entire line within the buffer contains blanks, that blank line is not written. An exception to this is when the new line character is used to force a new line. In this case a blank line might be printed.
- For message processing, the first blank delimits the message identifier. If there is no data following the message identifier, one blank line is written.
- For message processing, the overflow indentation used is the sum of the specified overflow indentation and the length of the message identifier. If PROFILE NOMSGID is in effect, the length of the message identifier is zero.
- If a conditional heading has been saved and the current exit routine fails, the exit service cleans up the conditional heading so as not to affect the next exit to get control.
- The expanded print service does not change any fields in the PPR2 parameter list. If the user provides their own print buffer, the contents of the print buffer are not changed. If the user points to the buffer supplied by IPCS and addressed by ADPLBUF as the print buffer, the buffer is blanked out on return from the expanded print service.
- When the new line character appears in the print buffer multiple times, a skip to a new line occurs for each occurrence. In the following example *NL* represents the EBCDIC new line character:

  ```
  NLITEM A NLNL ITEM B
  ```

  produces the following data:

  ```
  Line 1: blanks
  Line 2: ITEM A
  Line 3: blanks
  Line 4: ITEM B
  ```

- All leading new line characters are processed before doing any message identifier processing.
- An expanded print parameter list that specifies a zero length for the text causes one blank line to be written.
- Any new line characters that appear in the message identifier can cause unpredictable results depending on whether message identifiers are being suppressed.

Example

Figure 21 on page 120 illustrates the subroutine EXPRINT using the expanded print service to transmit a simple user message with a message identifier.

```
*
*The EXPRINT subroutine calls the expanded print service to
*transmit a simple user message with a message identifier.
*
*======  Set up for call to expanded print service
         L     R02,ADPLBUF        Address of print buffer
         ST    R02,PPR2BUF        Put address into parameter list
         MVC   PPR2BUFL,=A(132)   Length of print buffer
         MVC   0(132,R02),MESSAGE Fill print buffer
         MVI   PPR2PFL1,PPR2MSG   Indicate buffer contains a message
*======  Call the expanded print service
         SPACE 1                  Begin standard module epilogue
         L     R15,ADPLSERV       - Exit services router
         CALL  (15),((ABDPLPTR),REQCODE,PPR2) Call expanded print
ABDPLPTR EQU   11                 Register 11 - ABDPL address
REQCODE  DC    A(ADPLSPR2)        Request code for service router
MESSAGE  DC    CL132'USR12345I User message with an identifier'
PPR2     BLSUPPR2 DSECT=NO        expanded print Parameter list
*-------------------------------------------------------------------
*        Define the format of the ABDPL addressed by R1 on input
*-------------------------------------------------------------------
         BLSABDPL ,               Common parameter list
```

Figure 21. Example - Invoking the Expanded Print Service

Note:

1. This is a non-reentrant example that depends on the DC statements produced by macro BLSUPPR2 to initialize most of the storage occupied by the expanded print service parameter list.
2. The print buffer supplied by IPCS or SNAP is used as the buffer that the message is composed in. This is indicated to the expanded print service by placing the address of the buffer into PPR2BUF and the length of the message into PPR2BUFL. In this case the message is a simple literal message:

   ```
   User message with an identifier'
   ```

3. The message has the message identifier:

   ```
   'USR12345I
   ```

   that is removed when it is transmitted to a user who has NOMSGID in the TSO/E PROFILE command in effect. The message identifier is retained as part of the message when the message is transmitted to a user who has MSGID in effect. The presence of a message identifier as part of the message is indicated by turning on bit PPR2MSG in flag byte PPR2PFL1.
4. Once the message buffer has been prepared and described in the expanded print service parameter list, all that remains is to call the service router, indicating that the expanded print service is desired. The message is transmitted.

Format Model Processor Service

The format model processor service formats and prints an entire control block using a control block model. If a model name is specified, the service loads the model. If a dump address is specified instead of a buffer address, the service accesses the storage, using either the length indicated in the model or the value in ADPLPBLS, if it is not zero.

The maximum size of the control block is 64 kilobytes

You can create your own model by using the BLSQMDEF, BLSQMFLD and BLSQSHDR macros. These macros allow you to tailor your requested control block's output.

You can extend the capabilities of the format model processor service by writing a model processor formatting (MPF) exit routine.

See the following:

- "Model Processor Formatting (MPF) Exit Routine" on page 81 for information about writing model processor formatting (MPF) exit routines
- "Format Models" on page 122 for a discussion of the format models
- *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for information about the BLSQMDEF, BLSQMFLD, and BLSQSHDR macros

Note that in OS/390 Release 10 and higher, 64-bit addresses and dimensions may be designated.

Requirements

Before invoking this service, your exit routine must place the following information into ADPLPFMT:

- Set field ADPLPPTR to contain the model's address, or set field ADPLPCHA to contain the name of a model to be loaded.
- Set field ADPLPBAV to contain the dump data's address, or set field ADPLPBAS to contain the buffer address.
- Set field ADPLPBLS to contain the length of the buffer, if ADPLPBAS is specified.
- Set field ADPLPVCL to contain a view control specification.

Invoking the Service

After setting the required fields, your exit routine can invoke the format model processor service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL.
- The address of the format model service code (ADPLSFMT).
- The address of the format model service parameter list (ADPLPFMT) mapped by the BLSABDPL mapping macro. You must obtain storage for or establish addressability to the format model processor service parameter list. Before initializing the fields, remember to set the fields to 0.

  Note: To format a literal value, the third parameter must be the address of the area mapped by the BLSRESSY mapping macro. The BLSRDATS area, which is part of the BLSRESSY area, must refer to the literal value.

For the control block formatter and format model parameter list, ADPLPFMT, in the BLSABDPL data area, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Output

When the format model processor service returns control to your exit routine, register 15 contains one of the following return codes:

Code
Meaning

00
The format model service completed normally

04
The control block was truncated, or there was no view match

08
There is an error in the control block model

16
Failure in processing this service because of an incorrect model specification caused ending

If the format model processor service completed successfully, your exit routine has a formatted control block or data area.

Customization

The information that can be customized is identical to that described for the control block formatter, which is discussed in the previous topic.

Example

Figure 22 on page 122 illustrates the subroutine MODEL using the format model processor service to format a user defined control block.

```
  MODEL    USING ABDPL,R4          Base register for ABDPL
           USING ADPLPFMT,R5       Base register for ADPLPFMT
           L     R4,0(R1)          Address of ABDPL
           L     R5,4(R1)          Address of ADPLPFMT
           MVC   ADPLPPTR,=A(CBMODEL) Model s address
           MVC   ADPLPBLS,ADPLDLEN  Length accessed
           L     R15,ADPLSERV      Load address of exit services router
           CALL (15),((R4),CODEFMT,(R5)) Invoke service
           LTR   R15,R15           Was MYBLK displayed?
           BNZ   EXIT              No. End processing.
  . . .
CODEFMT  DC   A(ADPLSFMT)        Format service code
ABDPLPTR EQU  11                 General register 11
```

Figure 22. Example - Invoking the format model processor service

Format Models

A format model is a nonexecutable, read-only data structure that is used by the format model processor service to format a control block (or any data area). By assembling the set of macros, BLSQMDEF, BLSQSHDR, and BLSQMFLD, you can create format models. The format model contains:

- A header, created by the BLSQMDEF macro
- Subheaders, specified by the BLSQSHDR macro
- A list of entries that describe each field, created by the BLSQMFLD macro. Up to 40 BLSQMFLD macros can be specified for each line to be displayed.

See the following:

- Figure 22 on page 122 for an example of how to create your own model using the BLSQMDEF and BLSQMFLD macros
- *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for information about the BLSQMDEF, BLSQSHDR, and BLSQMFLD macros

Residence of Models

Models can reside in either a single or multi-CSECT load module or reside in the program referencing it.

Other Uses for Models

Format models and the format model processor can be used for purposes other than formatting control blocks. They can be used for decoding flag bytes where each bit has a unique significance. You can construct a model of subheaders, where each one of the subheaders has a unique general view definition that corresponds to the bit position in the flag byte. To perform the decoding, take the contents of the flag byte and use it for the view control in field ADPLPFMT. Put the address of the model in field ADPLPPTR and call the format model processor service. A nonzero buffer address should be provided to prevent any storage access attempt by the format model processor service.

Another use of models is to present summarized dump data. This is done by moving the data into a locally defined structure and creating a model with labels, offsets and lengths corresponding to the structure.

Still another use is to obtain a hexadecimal dump representation of storage that contains variable length fields. Construct a model with one field entry that specifies the hexadecimal dump view (X'0400'), data offset 0, and a field length of the maximum size expected. Then do the following:

- Set field ADPLPBLS to the actual length of the desired display
- Set field ADPLPSTM to suppress truncation messages
- Establish addressability to the model and dump data
- Call the format model processor

Get Symbol Service

The get symbol service retrieves symbols from the symbol table in the dump directory. This service can also be used to initialize the BLSRESSY macro for your exit routine, whether your symbol is in the symbol table or not. IBM recommends that you use this service prior to invoking the equate symbol service, which creates and adds entries to the symbol table. Use of these services can reduce the access time for subsequent references to the same control block or data area.

The get symbol service can retrieve literal values for symbols. The definition of a symbolic literal specifies the address space as LITERAL(0), which is an address space containing no storage.

For more information on the BLSRESSY mapping macro, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Requirements

Prior to invoking this service, your exit routine must place the following information into fields ESSYSYM, ESSYDTY, and ESSYDTD of the BLSRESSY mapping macro. The service assumes that the data is passed in ABITS=31 format and is expected in that format on return. Requests for valid definitions describing storage in ABITS=64 format will result in a “not found” return from this service.

Using these fields as search arguments, the get symbol service searches the symbol table for your requested symbol.

- Set field ESSYSYM to contain the equated symbol. When the symbol that you are retrieving is the UCB for device number 0003, code the symbol as ‘UCB0003’; when retrieving pointer seven from the stack, code the symbol as ‘Z00007’.

  For example, IPCS uses the symbol UCBdddd for a device number, with leading zeros before the number. For subcommand LISTUCB 12, IPCS creates the symbol UCB0012.

  Note that IPCS removes the leading zeros in the device number before displaying a UCB. For example, for subcommand LISTUCB 1D0, IPCS creates the symbol as UCB01D0, but displays the symbol as UCB1D0.
- Set field ESSYDTY to contain the data type code. The data type is a symbol's attribute. It might be a module, control block, character string, or any one of the attribute parameters that describe data. See BLSRDATT for the BLSRDATT macro.
- Field ESSYDTD must contain the data name. For this field either supply a meaningful name (such as MYAPPL for a module, or MYCB for a control block) or supply blanks.

Invoking the Service

After initializing the required fields, your exit routine can invoke the get symbol service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Register 1 must contain the address of the following three consecutive parameters:

- The address of the ABDPL
- The address of the get symbol service code (ADPLSGTS)
- The address of the area mapped by the BLSRESSY mapping macro, which is set to the symbol information

Output

When the get symbol service returns control to your exit routine, a copy of the symbol is in the area mapped by the BLSRESSY mapping macro.

Register 15 contains one of the following return codes:

Code
Meaning

00
The get symbol service completed normally.

04
The symbol was returned but an attention condition was detected. Preceding this return code, there are informational messages that provide additional diagnostic aid.

08
The symbol was returned but an error condition was detected. Preceding this return code, there are informational messages that provide additional diagnostic aid.

12
No symbol was returned.

16
An error in the IPCS environment was encountered.

Your exit routine receives an initialized BLSRESSY area whenever either:

- Field ESSYSYM is blanks.
- IPCS performs a search of the symbol table and cannot find the symbol defined in field ESSYSYM.

If the get symbol service is retrieving a literal, the system returns the literal in the area defined by the BLSRDATS mapping macro; this area is part of the BLSRESSY area. For more information, see BLSRESSY in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Example

Figure 23 on page 124 illustrates the subroutine GET using the IPCS-supplied get symbol service to locate the ASCB for address space 1. GET sets field ESSYSYM by using the naming conventions for IPCS symbols. GET sets field ESSYDTY by using the equate that corresponds to the data type for a control block. GET sets field ESSYDTD by using the actual control block's name.

```
*
* The GET subroutine calls the get symbol service to
* find the ASCB for ASID 1.
*
GET      MVC   ESSYSYM(31),=CL31'ASCB00001' Symbol
         MVI   ESSYDTY,ZZZDTYM  Structure
         MVC   ESSYDTD(31),=CL31'ASCB' Data name
         L     R15,ADPLSERV     Load address of exit services router
         CALL  (15),((ABDPLPTR),CODEGTS,ESSY) Invoke service
         CH    R15,=H'12'       Was get symbol service successful?
         BNL   EXIT             No.  End processing
ABDPLPTR EQU   11               General register 11
CODEGTS  DC    A(ADPLSGTS)      Get symbol service code
ESSY     BLSRESSY DSECT=NO      IPCS ES record buffer
*-------------------------------------------------------------------
*        Define the format of the ABDPL addressed by R1 on input
*-------------------------------------------------------------------
         BLSABDPL
```

Figure 23. Example - Invoking the Get Symbol Service

Name Service

The name service is used to convert an STOKEN or real address of a data space ASTE into:

- An ASID for an address space
- A data space name and owning ASID
- A hiperspace name and owning ASID
- A common addressable data space (CADS) name and owning ASID
- A subspace name and owning ASID

The name service can identify the data space for an STOKEN or a real ASTE address if the data space is accessible in the dumped environment; storage from the data space does not need to be dumped to enable the identification.

If requested, the service generates one line of output (see Figure 24 on page 125) describing the data space and/or address space associated with the specified STOKEN.

```
ADDRESS SPACE ASID(X'asid') STOKEN(X'stokenvalue')
DATA SPACE    ASID(X'asid') DSPNAME(name) STOKEN(X'stokenvalue')

HIPERSPACE    ASID(X'asid') DSPNAME(name) STOKEN(X'stokenvalue')
COMMON DSP    ASID(X'asid') DSPNAME(name) STOKEN(X'stokenvalue')
SUBSPACE      ASID(X'asid') SSPNAME(name) STOKEN(X'stokenvalue')
```

Figure 24. Name Service Output

Requirements

Prior to invoking this service, your exit routine must place the following information in the name parameter list (NAMP):

Field
Description

NAMPID
A standard control block identifier field.

NAMPPFLG
Contains the following input flag:

Flag
Description

NAMPFNOT
A flag that indicates that no output is requested listing the results of the translation. Messages are issued for errors detected in performing the translation.

NAMPASTE
The input is the real address of a data space ASTE.

NAMPSTKN
The input STOKEN to be translated.

When the NAMPASTE field is specified, NAMPSTKN is an output field that contains the STOKEN.

NAMPMODN
Contains the name of the module calling the name service. The name is used for diagnostic error messages only.

If the NAMP is incorrect, IPCS issues a message.

Invoking the Service

After setting the required field, your exit routine can invoke the name service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL
- The address of the name service code (ADPLSNAM)

- The address of the name service parameter list (NAMP) mapped by the BLSRNAMP mapping macro

The routine is called with the following attributes:

- AMODE(31)
- Task mode
- PRIMARY = SECONDARY = HOME
- Enabled
- Problem program state
- Key 8

See Figure 25 on page 127 for an example of calling the name service.

Output

If processing is successful, the return code is set to zero and the following fields are filled in:

Field
Description

NAMPPFLG
Contains the following output flags:

Flag
Description

NAMPFAS
A flag that indicates that the STOKEN was identified as an address space and that an ASID was returned.

NAMPFDS
A flag that indicates that the STOKEN was identified as a data space and that the data space name and owning ASID were returned.

NAMPFHS
A flag that indicates that the STOKEN was identified as a hiperspace and that the hiperspace name and owning ASID were returned.

NAMPFCAD
A flag indicating that the STOKEN was identified as a common data space and that the data space name and owning ASID were returned.

NAMPSTKN
When the NAMPASTE field is specified on input, NAMPSTKN is an output field that contains the STOKEN.

NAMPASID
The address space ASID or the owning ASID if the STOKEN represents a data space.

NAMPDSPN
The data space name.

NAMPOUT
A character field containing the space addressable by the STOKEN in standard IPCS display format.

NAMPOUTL
The length of the contents of the NAMPOUT field.

If an error in translation occurs, IPCS issues a message and sets the return code to 12.

Figure 25 on page 127, shows how a dump exit routine would invoke the name service to translate a STOKEN:

```
TSTSNAM  TITLE 'TSTSNAM--Name Service Usage Example'
TSTSNAM  START 0                  Sample dump processing exit
         SPACE 2                  Begin standard module prolog
*******  Standard module prolog
         SPACE 1                  Begin standard module prolog
         SAVE  (14,12),T,*        Save registers
         LR    R12,R15            Base register for TSTSNAM
         USING TSTSNAM,R12        Base register for TSTSNAM
         LA    R15,SAVEAREA       Address of local save area
         ST    R13,SAVEAREA+4     Chain input save area to local
         ST    R15,8(,R13)        Chain local save area to input
         LR    R13,R15            Establish use of local save area
         LR    ABDPLPTR,R01       Base register for ABDPL
         USING ABDPL,ABDPLPTR     Base register for ABDPL
         SPACE 2                  Begin dump formatting
*******  Use the name service to see whether XL8'1234567812345678'
*******  is a valid STOKEN in this dump.
         SPACE 1                  Begin dump formatting
         MVC   NAMPMODN,=CL8'TSTSNAM' Module requesting service
         MVC   NAMPSTKN,=XL8'1234567812345678' STOKEN
         L     R15,ADPLSERV       - Services router
         CALL  (15),((ABDPLPTR),name#SERVICE,NAMP) Check the STOKEN
         SPACE 2                  Begin standard module epilogue
*******  Standard module epilogue
         SPACE 1                  Begin standard module epilogue
         L     R13,SAVEAREA+4     Resume use of input save area
         RETURN (14,12),T,RC=(15) Restore registers and return
         SPACE 2                  Begin data definitions
*******  Define data
         SPACE 1                  Begin data definitions
R00      EQU   0                  Register 0
R01      EQU   1                  Register 1
ABDPLPTR EQU   11                 ABDPL base register
R12      EQU   12                 Register 12
R13      EQU   13                 Register 13
R14      EQU   14                 Register 14
R15      EQU   15                 Register 15
NAMP     BLSRNAMP DSECT=NO        Name service parameter list
SAVEAREA DC    18F'0'             Register save area
name#SERVICE DC A(ADPLSNAM)       Name service request code
         LTORG ,                  Literal pool
         BLSABDPL ,               Common parameter list
         END   TSTSNAM            Test dump formatting exit
```

Figure 25. Example - Invoking the Name Service

Name/Token Lookup Service

Use the name/token lookup service to obtain the token from a name/token pair. You specify the name and the level of the name/token pair. The name/token lookup service returns the following information:

- The token data
- Whether the name/token pair is persistent
- Whether an authorized program created the name/token pair
- The ASID for the address space associated with a name/token pair

Requirements

Prior to invoking this service, your exit routine must place information in the following fields of the name/token parameter list (NTKP). The NTKP is mapped by macro BLSQNTKP.

Field
Description

NTKPID
A standard control block identifier field. The identifier is NTKP1.

NTKPNAME
The name of the name/token pair.

NTKPMODN
The name of the module calling the name/token lookup service. IPCS uses this information for diagnostic error messages only.

NTKPASID
The ASID for the address space associated with the name/token pair.

For primary- and home-address-space-level name/token pairs, specify the ASID for the address space associated with the pair. For task-level name/token pairs, specify the ASID for the address space in which the TCB is located.

For a system-level name/token pair, set the field to zeros.

NTKPTCBP
The TCB address for a task-level name/token pair. For primary-, home-, and system-level name/token pairs, set this field to zeros.

NTKPPFL1
Contains the following flag:

Flag
Description

NTKPFNOT
A flag to suppress the printed output. Set this flag to 1 to suppress the output. Set this flag to 0 to display the output as shown in Figure 26 on page 129.

If you fill in NTKP incorrectly, IPCS indicates the error through a message.

Invoking the Service

After setting the required fields, your exit routine can invoke the name/token lookup service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL
- The address of the name/token lookup service code (ADPLSNTK)
- The address of the name/token lookup service parameter list (NTKP) mapped by the BLSQNTKP mapping macro

The routine is called with the following attributes:

- AMODE(31)
- Task mode
- PRIMARY = SECONDARY = HOME
- Enabled
- Problem program state
- Key 8

Output

When the name/token lookup service returns control to your exit routine, register 15 contains one of the following return codes:

Code
Meaning

00
Processing completed successfully

16
An error occurred in the lookup

If processing is successful, the following fields of NTKP are filled in:

Field
Description

NTKPPFL2
Contains the following output flags:

Flag
Description

NTKPNOTF
A flag that indicates that the name/token pair was not found.

NTKPAUTH
A flag that indicates that the name/token pair was created by an authorized program.

NTKPPRST
A flag that indicates that the name/token pair is persistent.

NTKPMSTG
A flag that indicates that dump storage needed for processing was missing.

NTKPFLERR
A flag that indicates that the service found an error in the dump data.

NTKPTOKN
The output token.

NTKPCASID
The ASID of the address space that created the name/token pair. This field is filled in for system-level name/token pairs only.

If field NTKPFNOT was set to 0, the name/token lookup service also lists the results (see Figure 26 on page 129) of the lookup.

```
System level
TOKEN.... SYSTLEV_TOKN_003
NAME..... SYSTLEV_NAME_003
ASID..... 000F
Persistent
Created by authorized program
```

Figure 26. Name/Token Lookup Service Output

If an error occurs in the lookup, IPCS issues a message.

Figure 27 on page 130 shows how a dump exit routine would invoke the name/token lookup service to retrieve a token:

```
TSTSNTK  TITLE 'TSTSNTK--Name/Token Service Usage Example'
TSTSNTK  START 0                  Sample dump processing exit
         SPACE 2                  Begin standard module prolog
*******] Standard module prolog
         SPACE 1                  Begin standard module prolog
         SAVE  (14,12),T,*        Save registers
         LR    R12,R15            Base register for TSTSNTK
         USING TSTSNTK,R12        Base register for TSTSNTK
         LA    R15,SAVEAREA       Address of local save area
         ST    R13,SAVEAREA+4     Chain input save area to local
         ST    R15,8(,R13)        Chain local save area to input
         LR    R13,R15            Establish use of local save area
         LR    ABDPLPTR,R01       Base register for ABDPL
         USING ABDPL,ABDPLPTR     Base register for ABDPL
         SPACE 2                  Begin dump formatting
*******] Use the name/token exit service to get the system-level
*******] token associated with name "NAMETESTVALUE"
         SPACE 1                  Begin dump formatting
         MVC   NTKPMODN,=CL8'TSTSNTK' Module requesting service
         MVC   NTKPNAME,=CL13'NAMETESTVALUE'  NAME
         L     R15,ADPLSERV       -]Services router
         CALL  (15),((ABDPLPTR),NTK#SERVICE,NTKP) Get the TOKEN
*******] Results will be displayed to user
         SPACE 2                  Begin standard module epilog
*******] Standard module epilog
         SPACE 1                  Begin standard module epilog
         L     R13,SAVEAREA+4     Resume use of input save area
         RETURN (14,12),T,RC=(15) Restore registers and return
         SPACE 2                  Begin data definitions
*******] Define data
         SPACE 1                  Begin data definitions
R00      EQU   0                  Register 0
R01      EQU   1                  Register 1
ABDPLPTR EQU   11                 ABDPL base register
R12      EQU   12                 Register 12
R13      EQU   13                 Register 13
R14      EQU   14                 Register 14
R15      EQU   15                 Register 15
NTKP     BLSQNTKP DSECT=NO        Name service parameter list
SAVEAREA DC    18F'0'             Register save area
NTK#SERVICE DC A(ADPLSNTK)        Name service request code
         LTORG ,                  Literal pool
         BLSABDPL ,               Common parameter list
         END   TSTSNTK            Test dump formatting exit
```

Figure 27. Example - Invoking the Name/Token Lookup Service

Select Address Space Identifier (ASID) Service

The select address space identifier (ASID) service scans the ASCBs in a dump by following the pointers in the ASVT and then generates a list of entries for selected address spaces within that dump. The select ASID service returns a list of ASCBs meeting selection criteria. The ASID service also creates storage maps entries for ASCBs, which indirectly improve performance.

The following address space selection criteria describe the types of address spaces that you might select:

ALL
Selects all address spaces in the dump.

CURRENT
Selects each address space that was active at the time the dump was generated.

When CURRENT is selected, a storage map entry is created for each address space selected. Each storage map entry describes the private area of one address space selected as AREA(CURRENT).

ERROR
Specifies processing of trace entries for address space identifiers (ASID) associated with tasks and/or address spaces in error. ERROR will process trace entries for any ASIDs associated with tasks or address spaces in error. To process just the trace entries for ASIDs associated with tasks in error, use TCBERROR.

TCBERROR
Specifies processing of trace entries for ASIDs associated with tasks in error. TCBERROR will process trace entries for any ASIDs associated with tasks in error; TCBERROR ignores errors that pertain to the whole address space.

Note: When you specify ERROR and TCBERROR in the same subcommand, IPCS processes all error address spaces, but will also identify those that are both ERROR and TCBERROR.

ASIDLIST
Selects a list of address spaces, a range of address spaces, or a single address space.

JOBLIST
Selects address spaces associated with a list of job names.

When JOBLIST is selected, a storage map entry is created for each address space selected. Each storage map entry describes the private area of one address space selected as AREA(JOB*aaaaaaaa*) where *aaaaaaaa* is one of two things:

1. **MASTER** for the master address space, ASID(1). The system carries the name of the master address space as ***MASTER***, but the asterisks cannot be used in an IPCS data name and are removed as a special case.
2. In all other cases *aaaaaaaa* is the job name or the name of the started task.

Restriction: Job names are checked to ensure that their names are valid in an IPCS data name, and, if they are not, no storage map entries are created. However, this is largely a theoretical concern. Use of normal system interfaces that create address spaces causes those address spaces to be given names that IPCS can use.

You can invoke this service by using the SELECT subcommand.

Requirements

Prior to invoking this service, your exit routine must place the following information into BLSABDPL:

- Set field ADPLPSEL to the requested address space selection criteria.

  Bit
  Meaning When Set to 1

  ADPLPSAL
  Obtains all address spaces that are marked assigned in the ASVT.

  ADPLPSCR
  Obtains the address space that was running on each processor and related address spaces.

  ADPLPSER
  Obtains any address space that satisfies the ERROR selection criterion.

  ADPLPSTE
  Obtains address spaces that satisfy the TCBERROR selection criterion.

  ADPLPSJL
  Obtains address spaces whose name matches a name in the list.

  ADPLPSAS
  Obtains address spaces corresponding to the ASIDs in the list.
- If job list or job name is requested, field ADPLPSJN must contain the address of the job name list. The job name list must be in the form of IKJIDENT PDEs for a list of 8-character identifiers. Field ADPLPSJN must point to the first PDE.
- If ASID list is requested, field ADPLPSAI must contain the address of the ASID list. The ASID list must be in the form of IKJIDENT PDEs for a list of ranges. Field ADPLPSAI must point to the first PDE. The ASID list can contain ranges of ASIDs as well as single ASIDs.

  For a range of ASIDs, the IKJIDENT PDEs provide for two instances of the ASID information. The first eight bytes describe the beginning of the ASID range and the second eight bytes describe the end of the ASID range. For example, specifying ASID(1:4) results in the first eight bytes describing a single EBCDIC

byte containing "1" and the second eight bytes describing a "4". If ASID(1) is entered instead, the first eight bytes will contain a "1" and the second eight bytes will contain zeros.

List PDEs such as those for job lists and ASID lists contain a pointer to a next PDE. It appears after the body of the PDE, at offset 8 in a job list PDE and offset 16 decimal in an ASID list PDE. A value of X'FF000000' appears in that pointer for the final PDE in the list.

Note: Both the job name list and ASID list are in the form of IKJIDENT PDEs. Your exit routine can either build the parameter list directly or link to module IKJPARS to detect parameters and build the chain of PDEs for ASIDLIST and JOBLIST. After processing, your exit routine should free the storage occupied by the PDEs.

Invoking the Service

After setting the required fields, your exit routine can invoke the select ASID service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Register 1 must contain the address of the following three consecutive parameters:

- The address of the ABDPL.
- The address of the select ASID service code (ADPLSSEL).
- The address of the select ASID service parameter list (ADPLPSEL) mapped by the BLSABDPL mapping macro, which has individual bits set to request the address spaces to be processed. You must obtain storage for or establish addressability to the select ASID service parameter list. Before initializing the fields, remember to set the fields to 0.

  You may use the ADPLPSXL bit in the ADPLPS31 field to avoid the 1023 entry limit. When this bit is set on, it tells the service to use the alternate length field.

For a mapping of the BLSABDPL data area, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Output

When the select ASID service returns control to your exit routine, register 15 contains one of the following return codes:

Code
Meaning

00
The select ASID service completed normally

04
The requested ASIDs were not all assigned or that a requested job name could not be found

08
A failure to access some data in the dump occurred

16
The service encountered an ending failure because of one of the following:

- The CVT pointer is 0
- Failure to access the CVT or ASVT in the dump
- Unable to obtain a work area

If the select ASID service completed successfully, your exit routine has, in field ADPLPSOL, the address of the list of address space descriptors.

The ASIDLIST is a data structure consisting of a header and an array of entries, one for each selected address space.

The header contains the following information:

- The number of entries in the array (ADPLOSCT)

- The size of the output list, in bytes (ADPLOSSZ)
- The subpool of the output list (ADPLOSSP)
- The selection flags (ADPLOSSF) indicating why some ASIDs or job names were not found
- The dump flags (ADPLOSDF) indicating the type of dump

Each array of entries contains the following information:

- A pointer to the ASCB for the requested address space (ADPLOSAP)
- Selection flags (ADPLOSF1) indicating why the ASID was selected
- Status flags (ADPLOSF2) indicating the status of the address space
- The CPUID (ADPLOSCP) if the address space was selected by specifying CURRENT
- The ASID (ADPLOSAI) of the address space
- The job name (ADPLOSJB) associated with the address space

Example

Figure 28 on page 133 illustrates the subroutine SELECT using the select ASID service to determine which ASIDs have ERROR and TCBERROR selection criteria conditions. SELECT sets field ADPLPSF1 to contain the indicators for the ERROR and TCBERROR address spaces (ADPLPSER and ADPLPSTE respectively).

Note: Issue the FREEMAIN macro to release the storage obtained for the output list if the return code from the select ASID service processing is not 16. Field ADPLOSSZ provides you with the size of the output list and field ADPLOSSP gives the subpool number. When the ADPLPSXL bit is set, use the value in ADPLPS31 (full word) for the FREEMAIN.

```
*
* The SELECT subroutine calls the select ASID service to
* process those ASIDs that have ERROR or TCBERROR conditions.
*
SELECT   MVI   ADPLPSF1,ADPLPSER+ADPLPSTE Set the parameter flags
         L     R15,ADPLSERV   Load address of exit services router
         CALL  (15),((ABDPLPTR),CODESEL,ADPLPSEL) Invoke service
         CH    R15,=H'8'      Was the select ASID service successful?
         BNL   EXIT           No. End processing
         L     R1,ADPLPSOL    Load address of Select ASID output list
         LH    R0,ADPLOSSZ-ADPLOSEL(,R1) Length of output list
         ICM   R0,B'1000',ADPLOSSP  Subpool number
         FREEMAIN R,LV=(0),A=(1) Free the output list, RC ¬=16

CODESEL  DC    A(ADPLSSEL)    Select ASID service code
ABDPLPTR EQU   11             General register 11
*-------------------------------------------------------------------
*        Reserve space for an initialized select ASID service
*        parameter list
*-------------------------------------------------------------------
         BLSABDPL DSECT=NO,AMDEXIT=NO,AMDOSEL=NO,                   *
               AMDPACC=NO,AMDPFMT=NO,AMDPECT=NO,AMDPSEL=YES
*-------------------------------------------------------------------
*        Define the format of the ABDPL addressed by R1 on input
*-------------------------------------------------------------------
         BLSABDPL DSECT=YES,AMDEXIT=YES,AMDOSEL=YES,                *
               AMDPACC=NO,AMDPFMT=NO,AMDPECT=NO,AMDPSEL=NO
```

Figure 28. Example - Invoking the Select ASID Service

Standard Print Service

The print service prints data from a dump.

See “Expanded Print Service” on page 116 for alternate ways of printing data.

Requirements

Prior to invoking this service, your exit routine must fill in the print line with formatted data. The print service causes the output buffer, which is a 133-character work area, to be printed, and returns a new

133-character buffer (set to blanks). IPCS supplies the carriage control character. When a print line is built, the print service must be given control to print the line.

Invoking the Service

After filling the print line, your exit routine can invoke the print service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following two consecutive parameters:

- The address of the ABDPL
- The address of the print service code (ADPLSPRT)

Output

Before the print service returns control to your exit routine, it clears the buffer. Register 15 contains one of the following return codes:

Code
Meaning

00
The line was printed

04
The line was not printed or that the attention interrupt key was pressed

Customization

To Override the Routing Parameter

Your exit routine can set bit ADPLPRT on and output will be sent to the terminal.

To Print a Blank Line

Your exit routine must pass control with a blank output buffer.

To Begin Printing at the Top of a New Page

Your exit routine must turn on the ADPLEJEC bit in field ADPLFLAG. This causes IPCS to skip to the top of a new page before printing the output buffer. IPCS turns the bit off when control is returned to your exit routine.

Example

Figure 29 on page 134 illustrates the subroutine PRINT using the print service to print data in a dump.

```
*
* The PRINT subroutine calls the print service to
* print MYBLK from the dump.
*
PRINT L     R15,ADPLBUF       Address of output buffer
         MVC   0(15,R15),=C'My Control Block' Fill output buffer
         L     R15,ADPLSERV     Load address of exit services router
         CALL  (15),((ABDPLPTR),CODEPRT) Invoke the print service
         LTR   R15,R15            Was the print service successful?
         BNZ   EXIT               No. End processing
CODEPRT  DC    A(ADPLSPRT)        Print service code
ABDPLPTR EQU   11                 General register 11
*-------------------------------------------------------------------
*        Define the format of the ABDPL addressed by R1 on input
*-------------------------------------------------------------------
         BLSABDPL
```

Figure 29. Example - Invoking the Print Service

Storage Access Service

The storage access service accesses data in a dump. You can specify a real or virtual address, the length of the data, and, optionally, an address space; in response, the storage access service reads the requested dump data into a buffer in storage.

Note: To access data in a data space, use the storage access function of the symbol service. See "Symbol Service" on page 140.

Requirements

Prior to invoking this service, your exit routine must place the following information into BLSABDPL or ADPLPACC, the access parameter.

- Field ADPLPAAD in ADPLPACC must contain the address of the requested dump data.
- Field ADPLDLEN must contain the length of the requested dump data. The maximum length that you can specify is 4096 bytes.
- Field ADPLPRDP must contain a bit indicator for the address qualification. If no bit is set, the default qualifier is virtual.
- Bit ADPLSAMK in field ADPLPRDP is the 24 bit address mask and ***must*** be set if you want to specify that the address in ADPLPAAD should be treated as a 24-bit address.
- Field ADPLASID must contain the ASID for accesses to information within an address space.

Invoking the Service

After setting the required fields, your exit routine can invoke the storage access service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL.
- The address of the storage access service code (ADPLSACC).
- The address of the storage access parameter list (ADPLPACC) mapped by the BLSABDPL mapping macro. You must obtain storage for or establish addressability to the storage access parameter list.

For a mapping of the BLSABDPL data area, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Output

When the storage access service returns control to your exit routine, field ADPLPART contains the address of the buffer containing the requested data, if the processing was successful. Register 15 contains one of the following return codes:

Code
Meaning

00
The storage was accessed successfully

04
The requested storage was not in the dump or the service was unable to access the storage.

Note:

1. Your exit routine must test the contents of register 15.
2. Your routine must use the data in the buffer to which ADPLPART points before the routine makes another call to an IPCS service. The reason is that the buffer contents are changed by a call to the storage access service and might be changed by a call to another IPCS service.

Customization

The following customization is available when using the storage access service.

Specifying a Different ASID Instead of the Current (Default) ASID

Your exit routine must store the requested ASID in field ADPLASID in BLSABDPL. If you do not specify an ASID to access storage IPCS uses the current ASID being processed, or in the case of exit routines called by the VERBEXIT subcommand, the ASID in the dump header record is passed.

Specifying Requests

To specify requests for any of the following:

- Data at a specified real storage address
- Processor status records
- The dump header record

Your exit routine can do the following:

- For real storage address requests, field ADPLPAAD must contain the real address of the data to be read.
- For processor status record requests, field ADPLPAAD must contain the processor address.

  In SADMP, the processor address is between X'0' and X'3F'. In an SVC dump the only valid processor address is X'0'.
- For header record requests, bit ADPLHDR must be set to 1.

IPCS determines the type of read operation requested by checking the bits in field ADPLPRDP. The following bit flags are used for address qualification:

Bit Flag
Meaning When Set to 1

ADPLVIRT
Request for virtual data

ADPLREAL
Request for real data

ADPLCPU
Request for processor data

ADPLHDR
Request for the dump header

When more than one bit in this field is on, IPCS considers the request incorrect and returns a nonzero value in register 15. When no bits are on, IPCS processes a request for data at a virtual storage address.

Note: The length field ADPLDLEN is not applicable for processor status and header record reads.

In the example in Figure 30 on page 137, the subroutine ACCESS uses the IPCS-supplied storage access service to access data within a dump. Fields ADPLDLEN and ADPLPAAD are set prior to invoking the storage access service.

```
*
* The ACCESS subroutine calls the storage access service to
* access MYBLK from the dump.
*
ACCESS   MVC   ADPLDLEN,=Y(MYBLKEND-MYBLK) Length of stor to access
         MVC   ADPLPAAD,=A(X'200') Dump address to access
         L     R15,ADPLSERV     Load address of exit services router
         CALL  (15),((ABDPLPTR),CODEACC,ADPLPACC) Invoke service
         LTR   R15,R15          Was storage retrieved?
         BNZ   EXIT             No.  End processing
         L     R10,ADPLPART     Yes.  Load address of buffer
*
* MYBLK--Define the format of a very simple control block
*        Note that this could be done by using a macro-invocation
*
CODEACC  DC    A(ADPLSACC)      Dump access service code
ABDPLPTR EQU   11               General register 11
*-------------------------------------------------------------------
*        Define the format of the ABDPL addressed by R1 on input and
*        reserve space for an initialized storage access service
*        parameter list
*-------------------------------------------------------------------
         BLSABDPL DSECT=NO,AMDEXIT=NO,AMDOSEL=NO,                  *
               AMDPACC=YES,AMDPFMT=NO,AMDPECT=NO,AMDPSEL=NO
MYBLK    DSECT ,                My simplest control block ever
MYBLKPSA DC    C'PSA'           Identifier
MYBLKDEF DC    X'00'            Flags
MYBLKD80 EQU   X'80'            1st flag bit
MYBLKD40 EQU   X'40'            2nd flag bit
MYBLKGHI DC    V(MYENTRY)       Address of my program
MYBLKEND EQU   *                End of my control block
*-------------------------------------------------------------------
*        Define the format of the ABDPL addressed by R1 on input and
*-------------------------------------------------------------------
         BLSABDPL AMDPACC=NO,AMDPFMT=NO,AMDPECT=NO,AMDPSEL=NO
```

Figure 30. Example - Invoking the Storage Access Service

Storage Map Service

The storage map service allows exit routines to process storage map entries and to obtain data represented by the storage map entries.

The functions provided by the storage map service are:

Storage Access (SA)
The storage map service accesses the storage requested in the storage map record that is passed as input. Because storage map records are being changed to describe data space storage, this function can be used to obtain data space storage from the dump. There are several ways to request the dump access by setting the function code flags.

The storage map service can retrieve literal values. The service can retrieve the entire literal value or an arbitrary, contiguous portion of it. The service does not add storage map entries to the storage map for symbolic literals; scan processing results are returned only to the caller.

Validity check
The storage map service allows the user to request that a control block be validity checked and an SA record (BLSRSASY macro) be created.

Describe block being scanned
The storage map service allows a user-supplied scan routine to introduce a control block in error, prior to describing the particular errors detected.

Note that in OS/390 Release 10 and higher, the storage map service supports 64-bit addresses and lengths.

Requirements

The storage map service parameter list is mapped by macro BLSRXMSP. The BLSRXMSP macro defines the following fields:

Field
Description

XMSPID
A standard control block identifier field.

XMSPPFLG
Processing flags.

XMSPNOMS
Indicates that wherever possible, no error or diagnostic messages should be issued.

XMSPV64
Indicates whether the input to the storage map service is in ABITS=31 or ABITS=64 format.

XMSPCODE
Function code that requests the major function to be performed.

XMSPSAR
The address of the storage map record being processed. Points to an area of storage mapped by mapping macro BLSRSASY.

XMSPBUF
The address of a buffer to contain the accessed storage when a storage access function is called.

The following function code constants are declared:

XMSPACC
1 - requests that the storage described by the storage map record be accessed and copied into the specified buffer location.

XMSPVAL
2 - requests that the storage described by the storage map record be processed by the appropriate validity check routine. Validity check routines are also known as scan routines and are defined to IPCS in parmlib members embedded in the BLSCECT parmlib member.

XMSPDIAG
3 - requests that the block described by the storage map record be designated as being in error. IPCS issues message BLS18058I, and also issues message BLS18059I if the address of the locating area for the block is known.

If an incorrect storage map service parameter list is passed to the storage map service, IPCS issues a message and sets a return code of 16.

The storage map service is supported only under IPCS. If the storage map service is requested under SNAP, the return code is zero and field ADPLCODE is set to 4.

Invoking the Service

After setting the required field, your exit routine can invoke the storage map service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL
- The address of the storage map service code (ADPLSMAP)
- The address of the storage map service parameter list (XMSP) mapped by the BLSRXMSP mapping macro

Output

The storage map service returns standard IPCS return codes to the caller. Specific storage map service return codes are:

Code
Explanation

00
All requested functions were performed successfully.

04
Attention (unusual but not necessarily erroneous) conditions were detected, but the requested function completed successfully. Most callers of the storage map service will want to treat this as a normal completion of the service.

08
Error conditions were detected, but the requested function completed successfully. Most callers of the storage map service will want to treat this as a normal completion of the service.

For example, assume that the validity check service was requested. An error condition would be indicated if the structure whose definition was retrieved contained a pointer to another structure that failed validation.

12
The request was not satisfied. This could be the result of user action such as the use of the attention mechanism or the END primary command of the dump display reporter. It could also be due to the absence of needed information in the dump data set.

16
The request was not satisfied. The reason is that errors in the current processing environment have been detected.

For example, this return code is returned if a bad input parameter list (XMSP) is passed to the storage map service. In this case, the system issues messages to explain the error that was identified.

See *z/OS MVS IPCS User's Guide* for a list of standard IPCS return codes.

The following example shows how a dump exit routine would invoke the storage map service to access storage:

```
TSTSMAP  TITLE 'TSTSMAP--Storage Map Service Usage Example'
TSTSMAP  START 0                  Sample dump processing exit
         SPACE 2                  Begin standard module prolog
*******  Standard module prolog
         SPACE 1                  Begin standard module prolog
         SAVE  (14,12),T,*        Save registers
         LR    R12,R15            Base register for TSTSMAP
         USING TSTSMAP,R12        Base register for TSTSMAP
         LA    R15,SAVEAREA       Address of local save area
         ST    R13,SAVEAREA+4     Chain input save area to local
         ST    R15,8(,R13)        Chain local save area to input
         LR    R13,R15            Establish use of local save area
         LR    ABDPLPTR,R01       Base register for ABDPL
         USING ABDPL,ABDPLPTR     Base register for ABDPL
         SPACE 2                  Begin dump formatting
*******  Use the storage map service to scan the CVT in the dump
         SPACE 1                  Begin dump formatting
         MVC   XMSPMODN,=CL8'TSTSMAP' Module requesting service
         LA    R15,SAAU           - Storage map record
         ST    R15,XMSPSAR        - Storage map record
         MVC   XMSPCODE,=Y(XMSPVAL) Validity check request
         MVC   SAAUAST,VIRTUAL#STORAGE Virtual storage address space*
                                  type
         MVC   SAAUAS2,=F'1'      Master address space ASID
         MVC   SAAULAD,ADPLCVT    - CVT (common storage)
         MVI   SAAUDTY,ZZZDTYM    STRUCTURE data type code
         MVC   SAAUDTD,NAMECVT    CVT data name
         L     R15,ADPLSERV       - Services router
         CALL  (15),((ABDPLPTR),MAP#SERVICE,XMSP) Scan the CVT
         SPACE 2                  Begin standard module epilogue
*******  Standard module epilogue
         SPACE 1                  Begin standard module epilogue
         L     R13,SAVEAREA+4     Resume use of input save area
         RETURN (14,12),T,RC=(15) Restore registers and return
         SPACE 2                  Begin data definitions
*******  Define data
         SPACE 1                  Begin data definitions
R00      EQU   0                  Register 0
R01      EQU   1                  Register 1
ABDPLPTR EQU   11                 ABDPL base register
R12      EQU   12                 Register 12
R13      EQU   13                 Register 13
R14      EQU   14                 Register 14
R15      EQU   15                 Register 15
XMSP     BLSRXMSP DSECT=NO        Storage map service parameter list
SAAU     BLSRSASY DSECT=NO        Storage map record
SAVEAREA DC    18F'0'             Register save area
MAP#SERVICE DC A(ADPLSMAP)        Storage map service request code
NAMECVT  DC    CL(L'SAAUDTD)'CVT' Data name for STRUCTURE(CVT)
VIRTUAL#STORAGE DC AL(L'SAAUAST)(ZZZASTCV) Virtual storage address  *
                                  space type
         LTORG ,                  Literal pool
         BLSABDPL ,               Common parameter list
         END   TSTSMAP            Test dump formatting exit
```

Figure 31. Example - Invoking the Storage Map Service

Symbol Service

The symbol service enables exit routines to process symbols and obtain data represented by the symbols. The symbol service differs from the get and equate symbol services by requiring callers to pass parameter list XSSP (mapped by mapping macro BLSRXSSP) that describes which symbol functions to use. For reasons of completeness and compatibility, some of the symbol service functions duplicate the functions provided by the equate symbol and get symbol services.

The functions provided by the symbol service are:

Equate Symbol
This function adds the specified symbol to the dump directory. The symbol can be for a literal value.

Get Symbol
This function retrieves the equate symbol record (ESR) for the specified symbol.

Storage Access
This function accesses the storage requested in the equate symbol record that is passed as input.

Validity check
This function allows the user to request a validity check for a control block and the creation of an SA record (BLSRSASY macro).

Check for Active TCB
This function allows the user to determine if the task described by the passed equate symbol record was active at the time of dump. Checks for active tasks are only valid on a SADMP.

Note that in OS/390 Release 10 and higher, the symbol service supports 64-bit addresses and lengths.

Requirements

The symbol service parameter list is mapped by macro BLSRXSSP. Table 14 on page 141 is a description of the fields defined by the macro, and the information required before your exit routine can invoke this service. Whenever field names are mentioned in the table, it is assumed that the symbol service parameter list was defined with a prefix of **XSSP** and that the equate symbol record was defined with a prefix of ESR. A complete listing of the ESR fields is given in Table 16 on page 144.

Table 14. The symbol service parameter list, XSSP, as mapped by macro BLSRXSSP

Field Name	Field Description	Required Information
XSSPID	A standard control block identifier field	C'XSSP1' - generated from an XSSP expansion that specifies COMPLETE.
XSSPPFLG	Processing flags **XSSPNOMS** This flag indicates that, whenever possible, no error or diagnostic messages should be issued. **XSSPDCS** This flag indicates that data characteristics have been supplied. This is necessary when externally-described data entities (e.g. SGT, PGT) are specified with the equate symbol record passed. **XSSPBIT64** This flag indicates whether BLSRESSY structures should be returned in ABITS=31 or ABITS=64 format.	
XSSPCODE	Function code that requests the major function to be performed.	Must contain a value greater than 0 and less than the highest defined code. Use the constants provided to set this field.
XSSPESR	The address of the equate symbol record being processed. Points to an area of storage mapped by BLSRESSY.	Must contain the address of a properly initialized equate symbol record.
XSSPBUF	The address of a buffer to contain the accessed storage when a storage access function is called.	Must contain the address of a buffer when the service is requested to access storage for the user.
XSSPMODN	The name of the module calling the symbol service. This is used for diagnostic error messages only.	Should contain the module name of the caller for diagnostic purposes.

Table 15 on page 142 describes the declared code constants, their required information, and their output.

Table 15. Symbol Service Function Code Constants

Constant Name	Constant Description	Required Information	Output
XSSPEQU (Equate symbol)	Requests the symbol service to add or replace the symbol, represented by the passed ESR, in the dump directory.	All ESR fields are required except ESRRDX. The fields most often set by dump exit routines are the ESRSYM, ESRAS, ESRLAD and ESRD.	No direct output is generated. The IPCS symbol table is updated with the new symbol or an identical symbol is replaced.
XSSPGET (Get symbol)	Requests the symbol service to extract the ESR for the passed symbol from the dump directory.	The ESR fields required by the get symbol service are the ESRSYM, ESRDTY and ESRDTD.	The output from this function is a completely filled in ESR.
XSSPACC (Storage access)	Requests the symbol service to access and copy the storage described by the ESR into a specified buffer location. **Note:** Some dump exit routines only need to access storage from the summary dump or a SADMP. The regular portion of an SVC dump is volatile and might not be useful. In order to accomplish this, the exit routine should use the SV or SS address space type code in the equate symbol record.	All ESR fields are required except ESRRDX. The fields most often set by dump exit routines accessing virtual storage are the ESRSYM, ESRAST, ESRAS2, ESRLAD, ESRDTY, ESRDTD and ESRDLE.	The buffer pointed to by XSSPBUF is filled in with the requested data.
XSSPACCN (Resolve symbol and get storage)	Requests the symbol service to resolve the symbol definition and then access and copy the storage described by the ESR into the specified buffer location. The main purpose of this function is to access control block storage for which validity check and find routines are defined.	The ESR fields required by the “Resolve” portion of the service are the ESRSYM, ESRDTY, and ESRDTD. The “access” function uses the callers ESRD fields together with the ESRLAD and ESRAS fields to access a section of storage and place it in the buffer pointed to by XSSPBUF.	On return to the caller, the ESR is in the state that was used to do the access and the buffer is filled in with required data.
XSSPACCV (Find, validity check, and access)	Requests the symbol service to validity check and then access and copy the storage described by the ESR into the specified buffer location. The main purpose of this function is to access control block storage for which validity check and find routines are defined.	The ESR fields required by the “Find” portion of the service are the ESRSYM, ESRDTY, and ESRDTD. The “access” function uses the callers ESRD fields together with the ESRLAD and ESRAS fields (provided by the find routine) to access a section of storage and place it in the buffer pointed to by XSSPBUF.	On return to the caller, the ESR is in the state that was used to do the access and the buffer is filled in with required data.

Table 15. Symbol Service Function Code Constants (continued)

Constant Name	Constant Description	Required Information	Output
XSSPVAL (Validity check)	Requests the symbol service to process the storage described by the ESR using the appropriate validity check routine. Validity check routines are also known as scan routines and are defined to IPCS in parmlib members embedded in the BLSCECT parmlib member. The main result of this function is to set the return code. A return code of 8 or less generally means that the control block is usable. A return code greater than 8 indicates a bad control block or an environmental error (e.g. user attention entered).	All ESR fields are required except ESRRDX. The fields most often set by dump exit routines for virtual storage are ESRSYM, ESRAST, ESRAS2, ESRLAD, ESRDTY and ESRDTD.	The ESRD fields are changed by this service to completely describe the bounds of the control block that was validity checked.
XSSPACTV (Check for active TCB)	Requests the symbol service to check the TCB described by the ESR to determine if the task was active at the time of the dump. The results of the test are returned in the AS1 field of the ESR. A null value (X‘FFFFFFFF’) indicates that the task is not active or that the dump is not an SADMP. A CPU number (e.g. X‘00000001' for CPU 1) indicates that the task was active on that CPU at the time of the dump.	All ESR fields are required except ESRRDX. The ESR is generally the same as for a validity check call. The ESRDTY field must indicate a data type of STRUCTURE. The ESRDTD field must indicate a structure name of TCB.	The output from this function is the ESRAS1 field. The ESRAS1 field is set to X'FFFFFFFF' for the following cases: • A virtual dump is being processed • The task was not active at the time of a SADMP This field is set to X'0000000n' when the task was active at the time of a SADMP. *n* is the CPU on which the task was active.
XSSPBASE	Requests the symbol service to initialize the BLSRESSY structure. When initialized, it is passed to a standard image (assuming no remark text is present in the structure passed). Bit XSSPBIT64 will be used to determine whether the caller wants an ABITS(64) BLSRESSY record base or an ABITS(31) instance.	All information in the ESR buffer is disregarded.	The bytes of the buffer ending with the length field for remark test are initialized with the length field being set to indicate that no remark is present.

If an incorrect symbol service parameter list is passed to the symbol service, message BLS18460I is issued and a return code of 16 is set.

The symbol service is supported only under IPCS. If the symbol service is requested under SNAP, the return code is zero and field ADPLCODE is set to 4.

Invoking the Service

After setting the required fields, your exit routine can invoke the symbol service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL
- The address of the symbol service code (ADPLSSYM)
- The address of the symbol service parameter list (XSSP) mapped by the BLSRXSSP mapping macro

Output

The symbol service returns standard IPCS return codes to the caller. See Table 16 on page 144 for specific symbol service return codes:

Code
Explanation

00
All requested functions were performed successfully.

04
Attention (unusual but not necessarily erroneous) conditions were detected, but the requested function completed successfully. Most callers of the symbol service can treat this as a normal completion of the service.

08
Error conditions were detected, but the requested function completed successfully. Most callers of the symbol service can treat this as a normal completion of the service.

For example, this return code would be set if the get symbol service was requested and the structure whose definition was retrieved contained a pointer to another structure that failed validation.

12
The request was not satisfied. This could be the result of user action such as the use of the attention mechanism or the END primary command of the dump display reporter. It could also be due to the absence of needed information in the dump data set. For the validity check function it means that the requested control block failed validity check.

16
The request was not satisfied. The reason is that errors in the current processing environment have been detected.

For example, this return code is set if a bad input parameter list (XSSP) is passed to the symbol service. In this case, a message is issued to explain the error.

Macro BLSRESSY expands to define a structure containing the following fields:

Table 16. BLSRESSY expansion for ESR fields

Field Name	Field Description
ESRRID	Identifier (must contain C‘ES’ on entry to any IPCS service that accepts this structure as a parameter)
ESRRDX	Dump data set index. Used by IPCS and not required to be set by a dump exit.
ESRSYM	Symbol name. (must be 1 to 31 characters long, start with an alphabetic and not contain special characters or imbedded blanks)

Table 16. BLSRESSY expansion for ESR fields (continued)	
Field Name	**Field Description**
ESRAS	Describes the address space of the area to be defined The BLSRDATS area in the BLSRESSY area describes the address space characteristics. For information, see BLSRDATS in BLSRESSY in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).
ESRLAD	Contains the logical address of an area described in a dump.
ESRD	Describes the data characteristics of the area to be defined. The BLSRDATC and BLSRDATT areas in the BLSRESSY area describe the data characteristics. For information, see BLSRDATC and BLSRDATT in BLSRESSY in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).
ESRMAD	Address of first byte missing.
ESRF	Informational flags relating to storage access
ESRABS	Absolute storage address for the address in ESRLAD.
ESRR	Any remark text that was associated with the symbol

The structure defined by macro BLSRESSY is used as a parameter to IPCS services in two ways:

1. IPCS services might accept partial initialization of the structure. Where this is accepted, the fields that must be set upon entry are enumerated as required inputs to the service. Field ESRRID is always included in this list.
2. IPCS services might require complete, correct initialization of the structure, except field ESRRDX. Complete, correct initialization of a base structure can be accomplished in the following ways:
   - BLSRESSY can be invoked to produce an image of a complete, correct record in a CSECT during compilation. At run time that image can be transcribed into a buffer in which an “interesting” record is to be built.
   - A valid record identifier (field ESRRID), a blank symbolic name (field ESRSYM), and any valid data type (field ESRDT) can be passed to the IPCS get symbol service or the corresponding function of the IPCS symbol service.

     The service will respond by creating a complete, correct record in the buffer passed. It will return an attention return code (register 15 contains 4) to indicate that no symbol table access was performed.

IPCS services that use the structure described by macro BLSRESSY as input can change it; the services can attempt to change at least field ESRRDX. When return codes less than 12 are returned by these services, the structure returned is complete and correct.

Note: When a data set is in fast path access mode, as indicated by bit ESSYFFP, access is only allowed using ESSYAST “BL”, and the block numbers must be sequential starting from 0.

For a complete mapping of the BLSRESSY macro expansion, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Figure 32 on page 146 shows how a dump exit routine would invoke the symbol service to retrieve a symbol definition:

```
TSTSSYM  TITLE 'TSTSSYM--Symbol Service Usage Example'
TSTSSYM  START 0                  Sample dump processing exit
         SPACE 2                  Begin standard module prolog
*******  Standard module prolog
         SPACE 1                  Begin standard module prolog
         SAVE  (14,12),T,*        Save registers
         LR    R12,R15            Base register for TSTSSYM
         USING TSTSSYM,R12        Base register for TSTSSYM
         LA    R15,SAVEAREA       Address of local save area
         ST    R13,SAVEAREA+4     Chain input save area to local
         ST    R15,8(,R13)        Chain local save area to input
         LR    R13,R15            Establish use of local save area
         LR    ABDPLPTR,R01       Base register for ABDPL
         USING ABDPL,ABDPLPTR     Base register for ABDPL
         SPACE 2                  Begin dump formatting
*******  Use the symbol service to locate the CVT in the dump
         SPACE 1                  Begin dump formatting
         MVC   XSSPMODN,=CL8'TSTSSYM' Module requesting service
         LA    R15,ESAU           - Symbol record
         ST    R15,XSSPESR        - Symbol record
         MVC   XSSPCODE,=Y(XSSPGET) Validity check request
         MVC   ESAUSYM,NAMECVT    CVT symbolic name
         MVI   ESAUDTY,ZZZDTYM    STRUCTURE data type code
         MVC   ESAUDTD,NAMECVT    CVT data name
         L     R15,ADPLSERV       - Services router
         CALL  (15),((ABDPLPTR),SYMBOL#SERVICE,XSSP) Scan the CVT
         SPACE 2                  Begin standard module epilogue
*******  Standard module epilogue
         SPACE 1                  Begin standard module epilogue
         L     R13,SAVEAREA+4     Resume use of input save area
         RETURN (14,12),T,RC=(15) Restore registers and return
         SPACE 2                  Begin data definitions
*******  Define data
         SPACE 1                  Begin data definitions
R00      EQU   0                  Register 0
R01      EQU   1                  Register 1
ABDPLPTR EQU   11                 ABDPL base register
R12      EQU   12                 Register 12
R13      EQU   13                 Register 13
R14      EQU   14                 Register 14
R15      EQU   15                 Register 15
XSSP     BLSRXSSP DSECT=NO        Symbol service parameter list
ESAU     BLSRESSY DSECT=NO        Symbol record
SAVEAREA DC    18F'0'             Register save area
SYMBOL#SERVICE DC A(ADPLSSYM)     Symbol service request code
NAMECVT  DC    CL(L'ESAUDTD)'CVT' Data name for STRUCTURE(CVT)
         LTORG ,                  Literal pool
         BLSABDPL ,               Common parameter list
         END   TSTSSYM            Test dump formatting exit
```

Figure 32. Example - Invoking the Symbol Service

Table of Contents Service

The table of contents service is used by dump exit routines to add entries to the table of contents (TOC) data set.

Requirements

Before invoking this service, your exit routine must fill in the print buffer with:

- The length of the entry, which occupies the first fullword of the print buffer.
- The text of the table of contents entry. This text follows the fullword length and must not exceed 40 characters.

The table of contents service pads out the buffer with periods and inserts the current page number. It then adds the entry to the table of contents data set.

Invoking the Service

After filling the print buffer, your exit routine can invoke the table of contents service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following two consecutive parameters:

- The address of the ABDPL
- The address of the table of contents service code (ADPLSNDX)

The table of contents service can also be invoked on the control block formatter service and the format model processor service.

Note: The table of contents data set must be allocated and opened. It is opened automatically whenever the print data set is opened, as long as the table of contents data set is allocated.

Output

An error message is printed if the length is zero or more than 40, or the text is all blank.

Before the table of contents service returns control to your exit routine, it blanks the buffer. Register 15 contains one of the following return codes:

Code
Meaning

00
The entry was added to the table of contents data set

04
The entry was not added

Customization

To obtain indentation of the entry, your program can initialize field ADPLLEV in BLSABDPL to values 1 through 4. Each higher level results in four more spaces of indentation.

Example

Figure 33 on page 147 illustrates the subroutine TOCRTN using the table of contents service to create a table of contents entry in the table of contents data set.

```
*
* The TOCRTN subroutine calls the TOC service to build and
* add a TOC entry to the TOC data set.
* Register 11 contains the address of the ABDPL.
*
TOCRTN   L     R02,ADPLBUF        Address of print buffer
         MVC   0(24,R02),TOCENT   Fill print buffer
         MVI   ADPLLEV,2          Indentation level 2
         L     R15,ADPLSERV       - Exit services router
         CALL  (15),((R11),REQCODE) Call Table of contents service
         L     R13,SAVEAREA+4     Resume use of input save area
         RETURN (14,12),RC=0
REQCODE  DC    A(ADPLSNDX)        Request code for service router
TOCENT   DC    A(20)
         DC    CL20'TOC entry for TOCRTN'
*-------------------------------------------------------------------
*        Define the format of the ABDPL addressed by R1 on input
*-------------------------------------------------------------------
         BLSABDPL
```

Figure 33. Example - Invoking the Table of Contents Service

WHERE Service

The WHERE service identifies in the area mapped by the BLSRPWHS mapping macro the storage map entry for the system area in which the specified address resides.

Note that in OS/390 Release 10 and higher, the WHERE service supports 64-bit addresses and lengths.

Requirements

Prior to invoking this service, your routine must do the following:

1. Initialize all storage described by a BLSRPWHS mapping macro. BLSRPWHS produces data constants when invoked within a CSECT to assist with this.
2. Place a description of the address space of interest in the format defined by macro BLSRDATS into field with suffix AS, and place the address of interest in the field with suffix LAD.
3. Update any options designated through the field with suffix PFLG to indicate the processing desired. APAR OA17842 adds support for flag bits DTYF (all selection bits), DTYFL (MODULE selector), DTYFM (STRUCTURE selector), and DTYFU (AREA selector) to permit the caller of the WHERE service to limit relationships that WHERE is to consider.

Field
Description

IDL
Indicates whether the parameter list is in ABITS=31 or ABITS=64 format

AST
Indicate the address is virtual

LAD
Address for the WHERE service

AS2
The ASID of the address

Invoking the Service

After setting the required fields, your exit routine can invoke the WHERE service by calling the exit services router whose address is in field ADPLSERV in the BLSABDPL mapping macro. Set register 1 to contain the address of the following three consecutive parameters:

- The address of the ABDPL
- The address of the WHERE service code (ADPLSWHS)
- The address of area mapped by the BLSRPWHS mapping macro, which is set to describe the address for which information is being requested

Output

If the BLSRPWHS area contains errors, IPCS issues a message. If the BLSRPWHS area is correct, the WHERE service returns information in the following fields in BLSRPWHS:

Field
Description

OUTP
WHERE output area

OLAD
Located object address

OD
Storage characteristics of object

OOFF
Offset into located object

OSAL
Length of system area name

OSAN
System area name

PWHSONOL
Length of NAME+OFFSET string

ONMO
NAME+OFFSET string

OMDP
Address of non-standard module name

When the WHERE service returns control to your exit routine, register 15 contains one of the following return codes:

Code
Meaning

00
The WHERE service completed normally.

04
The address specified has not been identified as a MODULE, STRUCTURE, or AREA.

08
An error condition was detected. Preceding this return code, there are informational messages that provide additional diagnostic aid.

12
Processing was ended. The attention interrupt key might have been pressed.

16
An error in the IPCS environment was encountered or the parameter list passed was incorrect.

Example

Figure 34 on page 149 illustrates the subroutine WHERES using the WHERE service to determine the system area in which a specified address resides.

```
*
*The WHERES subroutine calls the WHERE service to
*determine the system area that a specified
*address resides in.
*
*======] Set up for call to WHERE service
WHERES   MVC   PWHSAST,=C'CV'     Virtual storage
         MVC   PWHSAS2,=A(1)      ASID
         MVC   PWHSLAD,=A(X'01949110') Address for WHERE service
         MVI   PWHSPFL1,PWHSPUT   Print output
*======] Call the WHERE service
         L     R15,ADPLSERV       -]Exit services router
         CALL  (15),((R11),REQCODE,PWHS) Call the WHERE service
         LTR   R15,R15            Was the WHERE service successful?
         BNZ   EXIT               No.  End processing
R11      EQU   11                 Register 11 - ABDPL address
R15      EQU   15                 Register 15 - Entry point address
REQCODE  DC    A(ADPLSWHS)
PWHS     BLSRPWHS DSECT=NO        WHERE service parameter list
         BLSABDPL ,               Common parameter list
```

Figure 34. Example - Invoking the WHERE Service

Locate-Mode SWA Manager

Use the locate-mode SWA manager, IEFDQSVA, to convert 3-byte SWA virtual addresses (SVAs) to addresses and related data regarding SWA blocks in a dumped system.

See the following:

- See *z/OS MVS IPCS Commands* for information about the FINDSWA subcommand.
- See “Format Models” on page 122 for a discussion of the format models.
- See *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for information about the BLSRESSY macro.

Requirements

Control is passed to IEFDQSVA through the LINK service or through the CALL service after using the LOAD service to bring IEFQDSVA into virtual storage. If LOAD is used, balance that operation with use of DELETE after the final CALL is made.

Control is passed in 31-bit addressing mode using standard linkage conventions. The caller need not be in 31-bit addressing mode; the addressing mode information supplied in GPR 14 will be used to return control in the callers addressing mode.

- All pointers supplied must contain valid 31-bit addresses.
- Register 1 must contain the address of a 2-word parameter list.
- The first word in the parameter list must contain the address of the ABDPL.
- The second must contain the address of a SWAEPAX as described by macro IEFZB505.

Invoking the Service

Zero the storage described by SWAEPAX before the call except for fields SWVA and SWQMPA. Place the SVA to be converted in SWVA, and place the address of a BLSRESSY ABITS=31 structure in SWQMPA. The BLSRESSY structure should describe the location in the dump where the active JSCB that establishes context for SVA interpretation resides.

Output

IEFQDSVA uses standard IPCS return codes. If no serious or terminating conditions are encountered, the SWAEPAX passed as the second parameter is updated to contain the following information:

Table 17. Updated SWAEPAX

Field	Length	Description
SWBLKPTR	4	Pointer to the SWA block in the dumped system
SWBLKID	1	ID of the SWA block
SWLNGTH	4	Length of the SWA block
SWPFXPTR	4	Pointer to the SWA prefix in the dumped system
SWPFXLTH	2	Length of the SWA prefix

Obtaining Information About Coupling Facility Structures

You can issue the IXLZSTR macro from your IPCS exit to search a dump for information about coupling facility structures. To use the IXLZSTR macro, you must provide a user-written routine to access the data. See *z/OS MVS Programming: Sysplex Services Reference* for the following information:

- What information you can obtain through the use of IXLZSTR
- A sample program using IXLZSTR

For complete information about how to code the IXLZSTR macro, see *z/OS MVS Programming: Sysplex Services Reference*.

Obtaining Information About Loaded Modules

You can issue the CSVINFO macro from your IPCS exit to search a dump for information about loaded modules. To use the CSVINFO macro, you must provide a user-written module information processing routine (MIPR), which is given control by CSVINFO to process the information that CSVINFO obtains. See *z/OS MVS Programming: Authorized Assembler Services Guide* for the following information:

- What information you can obtain through the use of CSVINFO
- How to write a MIPR.

For complete information about how to code the CSVINFO macro, see *z/OS MVS Programming: Assembler Services Reference ABE-HSP*.

Quiesce IPCS Transaction

The quiesce IPCS transaction service permits exit routines to quiesce a transaction that is currently being processed by IPCS. The service allows any programs actively processing on behalf of the transaction to complete, but will not accept further requests to:

- Send output to the terminal
- Send output to the print data set
- Initiate dump directory I/O
- Retrieve dump or trace data

The service also quiesces command procedures such as CLISTs and REXX execs, unless the procedure has been coded to prevent such action, for example, by issuing the CLIST statement CONTROL NOFLUSH.

Note:

1. The service is designed to end only batch and interactive line mode transactions, not IPCS dialog transactions.
2. The service is not designed to end transactions for programs or command procedures that are in protracted loops.

Requirements

Your exit routine must have the following environment:

Minimum authorization:	Problem state with PSW key 8. **Note:** PSW keys 9-15 are not supported by TSO/E,
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Your exit routine may hold locks, but is not required to hold any.
Control parameters:	None

Before invoking the service, your exit routine must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Register 1 is not required on input. It is recommended, however, that register 1 contain the address of a fullword pointer. In the fullword pointer, the high-order bit should be set, and the low-order bits should contain the address of a fullword that contains X'0'.

13
Address of a 72-byte register save area, which is in the primary address space.

14
Return address. Bit 0 must contain 1 to indicate that control is to be returned in 31-bit addressing mode or contain 0 to indicate 24-bit addressing mode.

15
The entry point address of the quiesce IPCS transaction service, BLSUSTOP.

Invoking the Service

The quiesce IPCS transaction service is reentrant and may be loaded once and used repeatedly, as required. Invoke the service, BLSUSTOP, as follows:

- From an authorized program, different task: Schedule an IRB. See *z/OS MVS Programming: Authorized Assembler Services Guide* for information about scheduling an IRB.
- From an authorized program, same task: Issue a LOAD macro with EP=BLSUSTOP, then issue a SYNCH macro to reduce authority.
- From an unauthorized program, same task: Do one of the following:
  - Issue a LINK macro with EP=BLSUSTOP
  - Issue a LOAD macro with EP=BLSUSTOP, then call the service, using the address obtained from the LOAD macro

Output

When the quiesce IPCS transaction service returns control to your exit routine, register 15 contains one of the following hexadecimal return codes:

Code
Meaning

00
Request for quiesce processing accepted.

04
The request for quiesce processing was accepted, but IPCS detected potential error conditions.

08
The request for quiesce processing was accepted, but IPCS detected error conditions.

10
The quiesce request was rejected. This can happen for various reasons, such as:

- A shortage of virtual storage exists
- The service was invoked in an environment where IPCS was not established
- Your exit routine did not meet the specified requirements

Example

Figure 35 on page 152 illustrates the subroutine STOPIT invoking the quiesce IPCS transaction service.

```
STOPIT   START 0
         USING STOPIT,15
         LINK  EP=BLSUSTOP,PARAM=(NULLPARM),VL=1
         DROP  15
         BR    14
NULLPARM DC    F'0'
         END   STOPIT
```

Figure 35. Example - Invoking the Quiesce IPCS Transaction Service

TOD Clock Service

The time-of-day (TOD) clock service provides a caller, including your exit routine, with a TOD clock image. The service supports values from May 11,1971, at 11:56:53.685248 to January 25, 2114, at

11:50:41.055743. In the clock image, bit 0 being off does not mean that the value was within the standard epoch (which began on January 1, 1900, at 12:00 AM). Rather, it means that the value is within the first epoch (which begins on September 17, 2042, at 23:53:47.370496). The system truncates partial microseconds and does no rounding.

Two services are provided, as described in the following table.

STCK (8-bytes)	STCKE (16-bytes)	Description
BLSUXTID service	BLSUETID service	These services are made available in MIGLIB for invocation through LINK or a similar invocation mechanism. The services each take two parameters: 1. Input — a 26-character time stamp 2. Output — a buffer for a binary STCK or STCKE value The services update the output parameter, returning a binary TOD value (BLSUXTID service) or STCKE value (BLSUETID service). Return code 0 is returned if a supported time stamp is passed; otherwise return code 12 is returned.

Requirements

The caller must have the following environment for either service:

Minimum authorization: Problem state with any PSW key.

Dispatchable unit mode: Task

Cross memory mode: PASN=HASN=SASN

AMODE: 31-bit

ASC mode: Primary

Interrupt status: Enabled for I/O and external interrupts

Locks: No locks held.

Control parameters: Control parameters must be in the primary address space.

Restriction

Before invoking either service, the caller must not establish an enabled, unlocked, and task-mode (EUT) function recovery routine (FRR) by specifying EUT=YES on the SETFRR macro.

Registers

Before invoking either service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1

If using a call, the address of 2 fullwords in the primary address space:

- First fullword contains the address of an 26-byte character input area. The format of the area is: mm/dd/yyyy,hh:mm:ss.ffffff.
- The second fullword contains the address of an 8-byte time-of-day (STCK TOD) clock value for the BLSUXTID output area or contains the address of a 16-byte time-of-day (STCKE TOD) clock value for the BLSUETID output area.

Note: If using a LINK macro, specify the parameters in the PARAM parameter.

13
Address of a 72-byte register save area, which is in the primary address space.

14
Return address. Bit 0 must contain 1 to indicate that control is to be returned in 31-bit addressing mode or contain 0 to indicate 24-bit addressing mode.

15
The entry point address of the 26-character time stamp service, BLSUXTID or BLSUETID.

Invoking the Service

The TOD clock service is reentrant so that it can be loaded once and used repeatedly, as required. Invoke the service as follows:

- Issue a LINK macro with EP=*service name*.
- Issue a LOAD macro with EP=*service name*, then call the service, using the address obtained from the LOAD macro.

Output

When the TOD clock service returns control to the caller, the registers contain the following:

Register
Contents

0-14
Unchanged.

15
Return code.

The hexadecimal return codes are:

Code
Meaning

00
Successful completion.

12
Unsuccessful completion.

17-Character Time Stamp Service

The 17-character time stamp service provides a caller, including your exit routine, with an EBCDIC time stamp in 17-character format:

```
mm/dd/yy hh:mm:ss
```

Where:

mm
Month 01 - 12

dd
Day of the month 01 - 31

yy
Year within the century 00 - 99

hh
Hour in a 24-hour clock 00 - 23

mm
Minute 00 - 59

ss
Second 00 - 59

Requirements

The caller must have the following environment:

Minimum authorization:	Problem state with any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	Control parameters must be in the primary address space.

Restriction

Before invoking the service, the caller must not establish an enabled, unlocked, and task-mode (EUT) function recovery routine (FRR) by specifying EUT=YES on the SETFRR macro.

Registers

Before invoking the service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
If using a call, the address of 2 fullwords in the primary address space:

- First fullword contains the address of an 8-byte time-of-day (TOD) clock value.

  In the clock value, bit 0 is treated as on to allow the service to handle values from May 11, 1971, at 11:56:53 to September 17, 2042, at 23:53:47. In the value, the partial microseconds must be truncated and not rounded.
- Next fullword contains the address of a 17-byte output area.

Note: If using a LINK macro, specify the parameters in the PARAM parameter.

13
Address of a 72-byte register save area, which is in the primary address space.

14
Return address. Bit 0 must contain 1 to indicate that control is to be returned in 31-bit addressing mode or contain 0 to indicate 24-bit addressing mode.

15
The entry point address of the 17-character time stamp service, BLSUMTOD.

Invoking the Service

The 17-character time stamp service is reentrant and may be loaded once and used repeatedly, as required. Invoke the service, BLSUMTOD, as follows:

- Issue a LINK macro with EP=BLSUMTOD.
- Issue a LOAD macro with EP=BLSUMTOD, then call the service, using the address obtained from the LOAD macro.

Output

When the 17-character time stamp service returns control to the caller, the registers contain the following:

Register
Contents

0-14
Unchanged.

15
Return code.

The hexadecimal return codes are:

Code
Meaning

00
Successful completion.

04
Bit 0 of the input TOD clock value was off. The service formats the value as though the bit had been on. The caller should use the formatted time stamp with caution. If it is being used as part of a report, the caller might issue a message to the reader to use the time stamp with caution.

Example

Figure 36 on page 156 illustrates a LINK macro that invokes the service one time or a small number of times during the operation of an exit routine.

```
         LA    13,SAVEAREA       Prepare GPR 13 for call to BLSUMTOD
         .
         .
         .
         LINK  EP=BLSUMTOD,PARAM=(TODCLOCK,TIMESTAMP)
         .
         .
         .
SAVEAREA DS    18F               Standard GPR save area
TODCLOCK DC    X'80000000000000000' Clock value
TIMESTAMP DS   CL17              Field to receive time stamp
```

Figure 36. Example - Invoking the 17-Character Time Stamp Service with LINK

Figure 37 on page 157 illustrates a LOAD macro that loads the service, then calls it several times during the operation of an exit routine.

```
         LA    13,SAVEAREA       Prepare GPR 13 for call to BLSUMTOD
         .
         .
         .
         LOAD  EP=BLSUMTOD       Get address of BLSUMTOD in GPR 0
         .
         .     Top of processing loop
         .
         LR    15,0              Prepare GPR 15 for call to BLSUMTOD
         LA    1,PARMLIST        Prepare GPR 1 for call to BLSUMTOD
         BASSM 14,15             Prepare GPR 14 and call BLSUMTOD
*                                Change to AMODE 31 if necessary
         .
         .     Bottom of processing loop
         .
         DELETE EP=BLSUMTOD      Done using BLSUMTOD
         .
         .
         .
SAVEAREA DS    18F               Standard GPR save area
PARMLIST DC    A(TODCLOCK,TIMESTAMP) Parameter list for BLSUMTOD
TODCLOCK DC    X'80000000000000000'  Clock value
TIMESTAMP DS   CL17              Field to receive time stamp
```

Figure 37. Example - Calling the 17-Character Time Stamp Service

26-Character Time Stamp Service

The 26-character time stamp service provides a caller, including your exit routine, with an EBCDIC time stamp in 26-character format:

```
mm/dd/yyyy hh:mm:ss.ffffff
```

Where:

mm
Month 01 - 12

dd
Day of the month 01 - 31

yyyy
Year, including the century 1900 - 2099

hh
Hour in a 24-hour clock 00 - 23

mm
Minute 00 - 59

ss
Second 00 - 59

ffffff
Decimal parts of a second 000000 - 999999

Two services are provided, as described in the following table.

STCK (8-bytes)	STCKE (16-bytes)	Description
BLSUXTOD service	BLSUETOD service	These services are made available in MIGLIB for invocation through LINK or a similar invocation mechanism. The services each take two parameters: 1. Input — a binary TOD or STCKE value 2. Output — a 26-character buffer The services update the output parameter, returning a 26-character time stamp. Return code 0 is returned for successful completion of the request; otherwise return code 12 is returned.

Requirements

The caller for either service must have the following environment:

Minimum authorization: Problem state with any PSW key.

Dispatchable unit mode: Task

Cross memory mode: PASN=HASN=SASN

AMODE: 31-bit

ASC mode: Primary

Interrupt status: Enabled for I/O and external interrupts

Locks: The caller may hold locks, but is not required to hold any.

Control parameters: Control parameters must be in the primary address space.

Restriction

Before invoking either service, the caller must not establish an enabled, unlocked, and task-mode (EUT) function recovery routine (FRR) by specifying EUT=YES on the SETFRR macro.

Registers

Before invoking either service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1

If using a call, the address of 2 fullwords in the primary address space:

- First fullword contains the address of an 8-byte time-of-day (TOD) clock value.

  In the 8-byte clock value passed to BLSUXTOD, bit 0 set to off is not treated as if the value is within the standard epoch (which began on January 1, 1900, at 12:00 AM). Rather, it is treated as if the value is within the first epoch (which begins on September 17, 2042, at 23:53:47.370496).
- Next fullword contains the address of a 26-byte output area.

Note: If using a LINK macro, specify the parameters in the PARAM parameter.

13

Address of a 72-byte register save area, which is in the primary address space.

14

Return address. Bit 0 must contain 1 to indicate that control is to be returned in 31-bit addressing mode or contain 0 to indicate 24-bit addressing mode.

15
The entry point address of the 26-character time stamp service, BLSUXTOD or BLSUETOD.

Invoking the Service

The 26-character time stamp service is reentrant and may be loaded once and used repeatedly, as required. Invoke the service as follows:

- Issue a LINK macro with EP=*service name*.
- Issue a LOAD macro with EP=*service name*, then call the service, using the address obtained from the LOAD macro.

Output

When the 26-character time stamp service returns control to the caller, the registers contain the following:

Register
Contents

0-14
Unchanged.

15
Return code.

The hexadecimal return codes are:

Code
Meaning

00
Successful completion.

12
Unsuccessful completion.

Chapter 11. The IPCS Debug Tool

To assist you in debugging your IPCS exit routine, IPCS provides an exit routine debugging tool. The debug tool can be used with IPCS exit routines that use IPCS exit services invoked through the exit services router. Traps cannot be set on those services that are not invoked through the exit services router.

See Table 4 on page 10 for a list of which exit services are and are not invoked through the exit services router.

Implementing the Debug Tool

The debug tool is implemented through four IPCS subcommands:

When You Want to:	Use the Following Subcommand:
Selectively enable traps	TRAPON
Selectively disable traps	TRAPOFF
Resume trap processing from a STOP trap	GO
Display the status of currently active traps	TRAPLIST

Enabling IPCS-Supplied Traps

By using the TRAPON subcommand, traps can be set to perform trap processing both before invoking a requested service and before returning to the caller of a service after the service has completed processing.

IPCS supplies traps for the following exit services:

Exit Service:	Code Specified on the TRAPON Subcommand:
All exit services	ALL
Add symptom	ADS
Contention queue element (CQE) create	CQE
Control block formatter	CBF
Control block status (CBSTAT)	CBS
Equate symbol	EQS
Exit control table (ECT)	ECT
Expanded print	PR2
Format model processor	FMT
Get symbol	GTS
Name	NAM
Name/token lookup	NTK
Select address space identifier (ASID)	SEL
Standard print	PRT

Exit Service:	Code Specified on the TRAPON Subcommand:
Storage access	ACC
Storage map	MAP
Symbol	SYM
Table of contents	NDX
WHERE	WHS

Specify the INPUT parameter on the TRAPON subcommand to request trapping before invoking a requested service. Specify the OUTPUT parameter on the TRAPON subcommand to request trapping after the completion of a requested service. The default is NOINPUT, if INPUT is not specified, and NOOUTPUT, if OUTPUT is not specified.

Table 18 on page 162 describes the information that can be trapped and the subcommand used to enable such a trap:

Table 18. Trapping IPCS Information

Information to be Trapped:	Subcommand Example:
Contents of the exit parameter list (BLSABDPL) and its extension. The extension is addressed in the BLSABDPL.	TRAPON INPUT(ABDPL)
Data passed to a service in addition to the basic parameters. This data can be displayed only for the format processor service. Also displays the model when the model address is supplied.	TRAPON (FMT) INPUT(DATA)
Data returned by a service in addition to basic parameters. This data can be displayed only for the storage access and select ASID services.	TRAPON (ACC SEL) OUTPUT(DATA)
Parameters passed to a service. These parameters cannot be displayed for the standard print or table of contents services.	TRAPON (***code-list***|ALL) INPUT(PARMS)
Return code from the service and the service code-list.	TRAPON (***code-list***|ALL) OUTPUT(RETC)
Parameters returned by a service. This is the same parameter list that is displayed as input, but it will show any values changed by the service. These parameters cannot be displayed for the standard print or table of contents services.	TRAPON (***code-list***|ALL) OUTPUT(PARMS)
Error information. Displays specified information returned by a service only if the return code from the service is nonzero.	TRAPON (***code-list***|ALL) OUTPUT(RETC,ERROR)
Note: If only one code for an exit service is specified on the TRAPON subcommand, no parentheses are needed around the code.	

There are other parameters that can be specified on the TRAPON subcommand, and the parameters illustrated in Table 18 on page 162 can be used in combination with each other.

Output from the TRAPON Subcommand

Figure 38 on page 163 illustrates the output generated by entering the following subcommands in this sequence:

```
TRAPON FMT INPUT(PARMS,ABDPL,DATA) OUTPUT(RETC)
CBFORMAT FD40D8. MODEL(XYZMOD1) ASID(1)
```

```
*** FMT INPUT  ABDPL ***

 BLSABDPL: 0002A000
    +0000  TCB..... 00000000  ASID..... 0001      SBPL..... 00
    +0007  FLAG..... 90       BUF..... 00065008  PRNT..... 8002B4F8
    +0010  CVT..... 00FD5AA0  MEMA..... 00061390  FRMT..... 8002B9C8
    +001C  COM1..... 00000000  COM2..... 00000000  COM3..... 00000000
    +0028  COM4..... 00000000  FMT1..... 00000000  FMT2..... 00000000
    +0034  EXT..... 00065408  ABDA..... 00000000  TRFM..... 00000000
    +0040  SERV..... 8002B628  LEV..... 00        IDX..... 00
    +0046  LNCT..... 003C      LNRM..... 0037      DLEN..... 0000
    +004C  OPLN..... 0000      PRDP..... 00        NDX..... 00059738
    +0054  PGNO..... 00000001  SRA..... 00065488

 ADPLEXTN: 00065408
    +0000  OPTR..... 00000000  CPPL..... 00065340  ESYP..... 00000000
    +000C  CODE..... 0000      PFLG..... 00        EFLG..... 00
    +0010  MAXL..... 4E        SCOL..... 04        COLS..... 14
    +0014  EFCD..... 0000      RSV1..... 00000000  00000000  00000000
    +0022            00000000  00000000  00000000  00000000  00000000
    +0036            00000000  00000000  0000

*** FMT INPUT  PARM  ***

 ADPLPFMT: 000815D8
    +0000  OPT..... 80        RET..... 0000      BLC..... 01
    +0004  CHA..... XYZMOD1   PTR..... 844D3098  BAS..... 00000000
    +0014  DAC..... 0000      BLS..... 0000      BAV..... 00FD40D8
    +001C  LME..... 00000000  VCL..... 0200      OSI..... 0000
    +0024  DL1..... 00000000  DL2..... 00000000  DU1..... 00000000
    +0030  DU2..... 00000000  EXP..... 00081618  FXC..... 00000000
     ===]  FORMATTING OPTIONS:  CHECK ACRONYM.
     ===]  RETURN INFORMATION NONE.

 ADPLPFXT: 00081618
    +0000  XID..... FEXT      XOP..... 24000000  XRT..... 00000000
    +000E  XIL..... 00        XAC..... ............................
    +002E  XLX..... 00        XFC..... 00
     ===]  EXTENDED OPTIONS NAME IS IN ES RECORD(N).
     STORAGE DEFINED BY ES RECORDS.
     ===]  EXTENDED RETURN INFO NONE.
    +0040  BF0..... 00000000  ES0..... 00081320

 ES: 00081320
    +0000  RID..... ES        RDX..... 00000006  SYM.....
    +002C                                        RV1..... 40
    +003C  AST..... CV        ASH..... 0000      AS1..... FFFFFFFF
    +0044  AS2..... 00000001  AS3..... 00000000  LAD..... 00FD40D8
    +0050  DOF..... 00000000  DLE..... 0000006A  DOB..... 00
    +0059  DLB..... 00        DTY..... U         DTD.....
    +006C                                        DTE..... 40
    +007C  DIM..... 00000000  DIL..... 00000000  DF..... 00000000
    +008C  MAD..... 00000000  KEY..... 00        FS..... 00
    +00A2  FC..... 00         ABS..... 00000000  RL..... 0000
    +00B6  RT.....                                        .....
    .
    .
    .
```

Figure 38. Output from the CBFORMAT Subcommand with the IPCS Debug Tool Active (Part 1 of 2)

```
    .
    .
    .
    +0048  BF1..... 00000000  ES1..... 00000000
    +0050  BF2..... 00000000  ES2..... 00000000
    +0058  BF3..... 00000000  ES3..... 00000000
    +0060  BF4..... 00000000  ES4..... 00000000
    +0068  BF5..... 00000000  ES5..... 00000000
    +0070  BF6..... 00000000  ES6..... 00000000
    +0078  BF7..... 00000000  ES7..... 00000000
    +0080  BF8..... 00000000  ES8..... 00000000
    +0088  BF9..... 00000000  ES9..... 00000000
    +0090  BFA..... 00000000  ESA..... 00000000
    +0098  BFB..... 00000000  ESB..... 00000000
    +00A0  BFC..... 00000000  ESC..... 00000000
    +00A8  BFD..... 00000000  ESD..... 00000000
    +00B0  BFE..... 00000000  ESE..... 00000000
    +00B8  BFF..... 00000000  ESF..... 00000000

*** FMT INPUT  DATA  ***

 MODEL: 844D3098
           MODEL HEADER
    +0000  CBACR....            CBLAB.... XYZENT    MLVL..... HBB3310
    +0018  CBLEN.... 006A       ACROF.... 0000      ACLEN.... 00
    +001D  LABLN.... 08         BEGCL.... 0A        CLDST.... 00
    +0020  HFLGS.... 00         ENTOF.... 24        DEFLV.... 01
           MODEL ENTRIES

               LABT      FLGS  VIEW  DATOF  DATLN
               --------  ----  ----  -----  -----
           001 FIELD1    0400  8200  0000   0002
           002 FIELD2    0400  0200  0002   0004
           003 FIELD3    0400  8200  0006   0008
           004 FIELD4    0400  0200  000E   0002
           005 FIELD5    0400  0200  0010   0004
           006 FIELD6    0400  0200  0014   0008
           007 FIELD7    0400  8200  001C   0002
           008 FIELD8    0400  0200  001E   0004
           009 FIELD9    2400  0200  0022   0008
           010 FIELDA    0000  0400  002A   0040

 XYZENT: 00FD40D8
      FIELD1... 005F  FIELD2... F1100000  FIELD3... 00000000  00000000
      FIELD4... 0000  FIELD5... 00000000  FIELD6... 00000000  7F746568
      FIELD7... 0000  FIELD8... 00800400  FIELD9... ABCDEFGH
*** FMT RETURN CODE: 00000000 ***
```

Figure 39. Output from the CBFORMAT Subcommand with the IPCS Debug Tool Active (Part 2 of 2)

See the following:

- Chapter 10, “IPCS Exit Services,” on page 95 for the return codes from each of the IPCS exit services
- *z/OS MVS IPCS Commands* for the TRAPON subcommand

Stopping and Resuming IPCS Trap Processing

The STOP parameter on the TRAPON subcommand and the GO subcommand are used to stop and resume IPCS trap processing. The STOP parameter is only valid when using IPCS in line-mode. If you specify either OUTPUT(STOP) or INPUT(STOP) when using IPCS in the background or while in the full-screen dialog, IPCS ignores it.

Use the STOP parameter on the TRAPON subcommand, and use the GO subcommand, as follows:

- Stop the progress of an exit routine temporarily by specifying the STOP parameter on the TRAPON subcommand. For example, enter the following subcommand to request that an IPCS-supplied trap be enabled for the where service to display the BLSABDPL and the parameters passed to the service, but the exit service is to wait (stop) for input from you before beginning:

  ```
  TRAPON WHS INPUT(ABDPL PARMS STOP)
  ```

- Resume the exit service processing by entering the GO subcommand.

You can also enter the following subcommands before entering the GO subcommand:

Subcommand
Purpose

HELP
Obtains help information about the function, syntax, or parameters of the IPCS subcommands

NOTE
Generates a message to the IPCSPRNT data set.

TRAPLIST
Obtains the status of IPCS-supplied traps.

TRAPOFF
Disables specified IPCS-supplied traps.

TRAPON
Enables specified IPCS-supplied traps. Use to alter the trap(s) currently in effect.

See *z/OS MVS IPCS Commands* for the following subcommands:

- GO
- HELP
- NOTE
- TRAPLIST
- TRAPOFF
- TRAPON with the STOP parameter

Disabling IPCS-Supplied Traps

Use the TRAPOFF subcommand to disable IPCS-supplied traps. For example, enter TRAPOFF ALL to disable all IPCS-supplied traps.

See *z/OS MVS IPCS Commands* for the TRAPOFF subcommand.

Getting the Status of IPCS-Supplied Traps

Use the TRAPLIST subcommand to display the status of IPCS-supplied traps. For example, enter TRAPLIST ALL to get a list of all the exit services and the traps that are currently set for those services.

See *z/OS MVS IPCS Commands* for the information about the TRAPLIST subcommand.

Chapter 12. IPCS Exit Services Supported for Compatibility

For compatibility, IPCS exit services developed before MVS/SP Version 3 are supported.

Services

The services supported for compatibility are:

- **The dump index service** — used to print a table of contents for each of the significant parts of the dump.
- **The format service** — used to convert data to printable hexadecimal (if necessary) and format data in the output buffer.
- **The old storage access service** — used to obtain data from a dump data set.
- **The print service** — used to print an output line.
- **The summary dump data access service** — used to access the summary dump data contained in an SVC dump.

IPCS currently provides equivalent services that these services provided. If you are developing a new exit routine, or reworking an old exit routine, and you need any of these services, use the IPCS service referenced at the beginning of each of the service descriptions.

This entire group of services is no longer recommended for two reasons:

1. Each member of the group expects non-standard input and generates non-standard output — interfaces of a type that research has shown to be error-prone when compared to services that are passed a formal parameter list.
2. Previously defined IPCS traps that assist in verifying the correct run of exits under development are not available for these services.

Note: Although you can invoke the storage access and print services directly using addresses in the BLSABDPL and standard linkage, it is recommended that you invoke them through the exit services router. The following sections explain how to invoke most of these routines directly.

You can only access the format service directly using the ADPLFRMT field in the BLSABDPL. This routine causes the output buffer to be formatted.

See the following topics for more information:

- Table 4 on page 10 for the recommended services
- “Format Service” on page 168

Dump Index Service

IPCS provides this service through the table of contents service. See “Table of Contents Service” on page 146.

The dump index is a table of contents containing a description and page number for each of the significant parts of the dump output. A blank line precedes the descriptions for the major sections (level one). Each major section description might have any number of subordinate descriptions, indicated by indentation, for parts or subdivisions of the section. Up to four levels of subordination exist.

When your output contains more than one dump index, each index appears on a new page. The title of the dump is printed at the top of each page of index output. The title corresponds to the dump that the index describes.

The user designs each index entry. Prior to calling the index service, the user moves the index entry to the output buffer pointed to by the ADPLBUF field in the exit parameter list. Each index entry is limited to 40 characters of data preceded by a four-byte header. When you specify more than 40 characters, the entry is truncated to 40 characters. Using the ADPLNDX field in the exit parameter list, the IPCS exit routine invokes the index service.

The index service writes the index entry from the buffer to the IPCS TOC data set. If the dump index service is called and messages are being routed to the terminal, then no index entry is written to the IPCS TOC data set.

When INDEX DD is not available, the index service writes the index entry to the PRINTER DD data set. The current page number (field ADPLPGNO in the exit parameter list) is used as the page reference for that index entry. On return from the index service, the index service routine blanks out the output buffer.

The exit parameter list (ABDPL) has an optional field ADPLLEV for specifying indentation levels. Valid values that you can specify for ADPLLEV are 1-4. When you specify an incorrect value, the default is 1. The indentation level for the index entry is equal to the ADPLLEV value multiplied by 4.

For any exit routine other than TCB exit routines, an index entry is added for your control statement before giving control to your exit routine. The format for the entry is shown, where *XXXXXXXX* is your control statement name.:

```
OUTPUT FROM XXXXXXXX VERB ..............00003000
```

Figure 40 on page 168 is an example using the dump index service.

```
         USING ABDPL,R3
         LR    R3,R1               SET UP ADDRESSABILITY FOR WORKAREA
         LA    R2,LEVEL            PUT INDENTATION FACTOR IN WORKAREA
         ST    R2,ADPLLEV
         L     R4,ADPLBUF          GET ADDRESS OF OUTPUT BUFFER
         USING BUFR,R4
         MVC   BUFR(LENG),INDEXENT PUT TABLE OF CONTENTS ENTRY
*                                  INTO OUTPUT BUFFER
         LR    R1,R3               SET UP REGISTER 1 WITH ADDRESS
*                                  OF WORKAREA FOR SERVICE ROUTINE
         L     R15,ADPLNDX         GET ADDRESS OF INDEX SERVICE
         BALR  R14,R15             INVOKE INDEX SERVICE
.
LEVEL    EQU   2                   INDENTATION FACTOR
.
INDEXENT EQU   *                   TABLE OF CONTENTS ENTRY FOR THIS
*                                  USER EXIT
         DC    XL3'0'
         DC    X'1D'
         DC    CL29'USER1 TABLE OF CONTENTS ENTRY'
LENG     EQU   *-INDEXENT
BUFR     DSECT
         DS    CL132               OUTPUT BUFFER ADDRESSED BY ADPLBUF
         BLSABDPL
```

Figure 40. Example - Using the Dump Index Service

Format Service

IPCS provides this service with the format model processor service. See “Format Model Processor Service” on page 120.

The format service is available to format the output of your module. This service performs two functions:

- Converts data to printable hexadecimal, if necessary
- Formats data in the output buffer

At each invocation, the format service formats a maximum of one output line. Your routine must provide the following input:

- The address of the output buffer by passing the exit parameter list address
- The addresses of data items and/or labels by providing format patterns

The ADPLBUF field of the exit parameter list contains the address of the 133-byte buffer where the output line is formatted. ADPLBUF also contains two words of work area, ADPLFMT1 and ADPLFMT2, that the format service uses. The format patterns indicate data and labels to be inserted in the output line.

Your module can invoke the format service once and specify format patterns to create one output line or invoke the format service more than once and each time specify patterns for part of an output line. Note that if you choose to format a portion of a line, you must be careful not to overlay previously-formatted portions.

When a line is formatted, your module must invoke the print service to perform the print operation. The format service does not print the buffer.

You can invoke the format service by obtaining the address of the format service from the ADPLFRMT field in the IPCS exit parameter list, BLSABDPL, and using standard linkage conventions. The following discussion gives details on this method of invoking the format service.

When invoking the format service using standard linkage conventions, you must pass the address of the first format pattern in register 0 and the address of the exit parameter list in register 1.

When the format service passes control back to your module, register 15 contains a return code of 00 or 04. The value of the code is the same as that received by the format service when it used the storage access routine; the meanings of the codes are the same as for the storage access routine.

Figure 41 on page 169 shows how a format service is invoked. Figure 43 on page 171 shows format patterns associated with that invocation.

```
*THIS SECTION ACTUALLY OUTPUTS TWO LINES OF A TAPE UCB THEN SKIPS A LINE
OUTPUTIT  LA R0,TAPE1         SET ADDR OF LINE 1 FORMAT PATTERN
          BAL RLINK, FORMAT   GO FORMAT LINE
          BAL RLINK, PRINTIT  GO TO PRINT IT WITH THE SERVICE RTN
          LA R0, TAPEL2       SET ADDR OF LINE 2 FORMAT PATTERN
          BAL RLINK, FORMAT   GO FORMAT LINE
          BAL RLINK, PRINTIT  GO TO PRINT IT WITH THE SERVICE RTN
          BAL RLINK, PRINTIT  GO PRINT A BLANK LINE
*THIS SUBROUTINE CALLS THE FORMAT SERVICE TO AUTOMATICALLY
*FORMAT AN OUTPUT LINE OF THE UCB.  INPUT TO THIS SECTION OF CODE
*IS THE ADDRESS OF THE FIRST FORMAT PATTERN IN REG 0.  REG 1 MUST
*CONTAIN THE ADDRESS OF THE ORIGINAL EXIT PARAMETER LIST.  IF
*DATA CANNOT BE OBTAINED BY THE SERVICE, CONTROL IS
*PASSED TO AN ERROR SUBROUTINE TO PRINT ERROR MESSAGE
FORMAT    L     R15,ADPLFRMT  GET RTN ADDR FROM PARM LIST
          BALR  R14,R15       GO TO SERVICE TO FORMAT LINE
          LTR   R15,R15       WAS FORMAT SUCCESSFUL
          BCR   8,RLINK       YES, GO PRINT LINE IMMEDIATELY
          B     ERROR         NO, GO TO ERROR ROUTINE
* THIS SECTION OF CODE IS USED TO CALL THE PRINT SERVICE.
* REGISTER 1 MUST CONTAIN THE ADDRESS OF THE ORIGINAL
* EXIT PARAMETER LIST.
PRINTIT    L     R15,ADPLPRNT       GET PRINT RTN ADDR FROM PARM LIST
           BALR  R14,R15            GO TO PRINT ROUTINE
           BR    RLINK              RETURN TO SECTION OF CODE ABOVE
```

Figure 41. Example - Using the Format Service

Format Patterns

The format service uses the format pattern to locate information and position it in the output buffer. A format pattern consists of four to seven fields. One field indicates how the format service should process the pattern. Three fields describe the position, length, and address of a label. Three fields describe the position, length, and the address of a data item.

If a series of format patterns are set up by your routine, the patterns that describe one line must be contiguous and have as a last entry a fullword of binary zeros.

The second and subsequent data and label addresses in a series are optional.

If these fields are not provided, the service uses the last address plus the length to locate a data item or label. To keep track of updated addresses, the format service uses the twelfth and thirteenth words in the exit parameter list to store label and data pointers.

See Figure 42 on page 170 for information about defining fields in a format pattern.

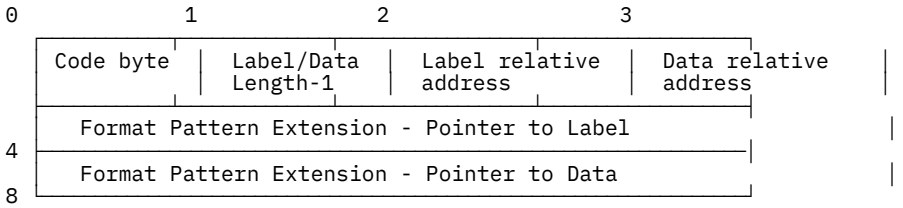

Figure 42. Format Pattern Description and Format Pattern Extension

Code byte
A 1-byte field that identifies the contents of a pattern. The settings are:

Bit	Meaning
0-1	Reserved.
2	Data is not to be converted to printable hexadecimal.
3	Data is in the caller's storage (not in the dump).
4	Data pointer follows (bit 5 must also be on). Either a dump address or a storage address. (See bit 3)
5	Data is to be placed in buffer.
6	Label pointer follows (bit 7 must also be on).
7	Label is to be placed in buffer.

If you do not set bits 4 and/or 6, then you should omit the label pointer and/or data pointer field(s) of the format pattern extension. Table 19 on page 170 shows the relationship between the bit settings of the code byte and the length and contents of the format pattern extension.

Label/Data Length-1
a one byte field that contains two length counts. The first four bits are the label length minus one. The last four bits are the data length minus one. This field is used to update label/data addresses in the parameter list if addresses are not in a pattern.

Label relative address
a one-byte field that indicates the position of labels in the buffer. Specify 0 as the relative address to indicate the first character of the buffer.

Data relative address
a one-byte field that indicates the position of data in the buffer. Specify 0 as the relative address to indicate the first character of the buffer.

Format Pattern Extension
an optional 8-byte field that contains one of the following:

- A label pointer
- A data pointer
- Both a label pointer and a data pointer
- Neither a label pointer nor a data pointer.

Bits 4 and 6 of the code byte determine the length and contents of the format pattern extension. Table 19 on page 170 shows the values for bits 4 and 6 of the code byte, and the resultant format pattern extension.

Table 19. Format Pattern Extension and Code Byte

Bit 4 of Code Byte	Bite 6 of Code Byte	Length of Format Pattern Extension	Contents of Format Pattern Extension
0	1	4 bytes	Label pointer

Table 19. Format Pattern Extension and Code Byte (continued)			
Bit 4 of Code Byte	**Bite 6 of Code Byte**	**Length of Format Pattern Extension**	**Contents of Format Pattern Extension**
1	0	4 bytes	Data pointer
1	1	8 bytes	Label pointer and data pointer
0	0	0 bytes	Format pattern extension does not exist

A *label pointer* contains the address of a character string that you want the format service to use for labels. When you include a label pointer in the pattern, you must set bits 6 and 7 of the code byte. When you do not specify a label pointer, the format service updates the label address in the exit parameter list and uses it.

A *data pointer* contains the address of the first byte of data that you want the format service to place in the output line. When this data is not in the caller's storage, you must set bit 3 of the code byte to zero; this causes the format service to use the storage access service to obtain the data. When you include a data pointer in the pattern, you must set bits 4 and 5 of the code byte. When you do not specify a data pointer, the format service updates the data address in the exit parameter list and uses it. If bit 2 is zero, the format service converts this data to printable hexadecimal.

Figure 43 on page 171 shows sample format patterns.

```
* THIS CHARACTER STRING DEFINES LABELS TO BE USED BY THE SERVICE
* FOR THE FORMATTED UCBs
*
TAPELB   DC   C'FSCTFSEQVOLISTABDMCT'    UCB LABELS
* THE FOLLOWING FIVE FORMAT PATTERNS DESCRIBE THE FIRST OF THE
* TWO LINES FOR A TAPE UNIT CONTROL BLOCK (UCB).  THE FIRST TWO
* PATTERNS ARE EXPLAINED FOR EACH FIELD.
* THIS PATTERN IDENTIFIES A LABEL AND DATA ITEM THAT WILL BE
* INSERTED IN THE FIRST LINE OF A TAPE UCB
TAPEL1   DS   0F         THIS PATTERN SETS UP THE FSCT FIELD
         DC   X'2F'      THIS CODE BYTE IS SET FOR: NO CONVERSION,
*                        DATA AND LABEL POINTERS FOLLOWING, AND LABEL
*                        AND DATA ITEM TO BE PLACED IN BUFFER.
         DC   X'31'      LABEL LENGTH-1 IS 3. DATA LENGTH-1 IS 1
         DC   FL1'12'    LABEL RELATIVE ADDRESS IN OUTPUT LINE IS 12
         DC   FL1'17'    DATA RELATIVE ADDRESS IN OUTPUT LINE IS 17
         DC   A(TAPELB)  POINTER TO STRING OF LABELS
DATADDR  DC   A(0)       ADDR OF DATA TO BE FILLED IN DURING EXECUTION
*
* NEXT PATTERN SETS THE FSEQ FIELD IN THE OUTPUT LINE.  THIS
* PATTERN USES THE SAME LABEL STRING AND DATA ADDRESS.  THE
* FORMAT SERVICE UPDATES ITS LABEL AND DATA ADDRESSES USING
* LENGTH FIELDS.  TWO WORDS IN THE EXIT PARAMETER LIST STORE THE UPDATED
* ADDRESSES.
*
         DC   X'25'      CODE BYTE SET FOR: NO CONVERSION, PLACE
*                        LABEL AND DATA IN BUFFER(NO ADDRESSES-DO UPDATE
         DC   X'31'      LABEL LENGTH(4)-1 IS 3.  DATA LENGTH(2)-1 IS 1.
         DC   FL1'28'    LABEL RELATIVE ADDRESS IN OUTPUT BUFFER IS 28.
         DC   FL1'33'    DATA RELATIVE ADDRESS IN OUTPUT LINE IS 33.
*
* THE REMAINING PATTERNS IN THIS LINE ARE CONDENSED
*
         DC   X'25',X'35',FL1'44',FL1'49'              VOLI FIELD
         DC   X'25',X'30',FL1'59',FL1'64'              STAB FIELD
         DC   X'25',X'30',FL1'76',FL1'81'              DMCT FIELD
         DC   X'00',X'00',FL1'0',FL1'0'                INDICATES END
*                                                       OF LINE
* THE SECOND LINE OF THE TAPE UCB IS DEFINED BY A SECOND SERIES OF
* FORMAT PATTERNS, TAPEL2.
```

Figure 43. Sample Format Patterns

Old Storage Access Service

Note:

1. If your exit must function in the SNAP environment or both the IPCS and SNAP environments, it is recommended that the storage access service be used instead of this service.
2. If your exit need only function in the IPCS environment, it is recommended that the storage access functions supplied by the storage map service or the symbol service be used instead of this service.

See the following topics for more information:

- "Storage Access Service" on page 135
- "Storage Map Service" on page 137
- "Symbol Service" on page 140

Your routine can pass control to the storage access routine for all references to storage in the dumped system (any reference to storage in a dumped system really refers to a record in the dump data set).

You can invoke the storage access routine in one of two ways:

- By obtaining the address of the exit services router from the ADPLSERV field of the exit parameter list, BLSABDPL.
- By obtaining the address of the storage access routine from the ADPLMEMA field of the exit parameter list, BLSABDPL, and passing control using standard linkage conventions. The following discussion illustrates this method of invoking the storage access routine.

To pass control to the storage access routine using standard linkage conventions register 0 must contain the address to be referenced and register 1 must contain the original exit parameter list address when the storage access routine receives control.

When the ADPLSAMK bit in the ADPLPRDP field equals 1, the storage access service clears the high order byte of the virtual address in register 0 before using the virtual address. For real addresses, the storage access service always clears the high order bit in register 0.

You can specify the number of bytes of data to be read (between 1 and 4096) in the ADPLDLEN field in the exit parameter list.

When the storage access routine returns control to your module, register 0 contains the address of the data requested and register 15 contains a return code. When you do not specify a length, the length of data available for reference depends on the boundary of the address requested. If the address is on a byte boundary, only one byte can be referenced. The maximum available is 4096 bytes on a 4K boundary.

In the example in Figure 44 on page 173, the subroutine MEMORY uses the storage access routine, then checks to be sure the data is returned.

Note that your routine must test the contents of register 15 for a return code of 00 to indicate that the storage access routine was successful. A code of 04 indicates that requested storage was not in the dump or the service was unable to access it.

The ASID used for accessing storage is the current ASID being processed, or, in the case of exit routines called by a control statement, the ASID in the dump header record is passed. Any ASID can be specified by the exit routine by storing the ASID in the exit parameter list.

In addition to requests for data at a specified virtual storage address, requests for data at a specified real storage address, requests for processor status records, and requests for the dump header record are satisfied. For real storage address requests, register 0 must contain the real address of the data to be read. For processor status record requests, the low-order byte of register 0 must contain the processor address.

In SADMP, the processor address is between X'0' and X'3F'. In an SVC dump the only valid processor address is X'0'. For information on the processor status record mapping macro, see "Invoking with the Exit Services Router" on page 98. Register 0 is not used for header record requests.

The type of read operation requested is determined from the parameter list field ADPLPRDP. When more than one bit in this field is on, the request is incorrect and a non-zero value is returned in register 15. When no bits are on, a request for data at a virtual storage address is processed.

Note: The length field ADPLDLEN is not applicable for processor status and header record reads.

```
* THIS ROUTINE IS USED FOR ALL REFERENCES TO STORAGE IN THE SYSTEM.
* 'PREG' ON ENTRY CONTAINS THE ADDRESS WANTED FROM THE SYSTEM; IT
* ALSO USES 'PREG' TO RETURN TO THE CALLER THE ADDRESS OF THAT
* REQUESTED DATA.  "R1" ALWAYS CONTAINS THE ADDRESS OF THE
* ORIGINAL PARAMETER LIST.
* IF THE REQUESTED ADDRESS IS NOT AVAILABLE, CONTROL IS PASSED
* TO THE ADDRESS IN 'ERROR'.
MEMORY  LR    R0,PREG        SET REQUESTED ADDR TO REGISTER 0
        L     R15,ADPLMEMA   GET STORAGE ACCESS ADDRESS FROM
*                            PARM LIST
        BALR  R14,R15        GO TO STORAGE ROUTINE
        LR    PREG,R0        RETURN REQUESTED ADDR IS SAME REG
        LTR   R15,R15        IS REQUESTED ADDRESS AVAILABLE
        BC    8,USEDATA      YES--USE IT
        B     ERROR          NO--GO TO ERROR ROUTINE
USEDATA L     RADDR,0(PREG)  GET DATA AT REQUESTED ADDRESS
```

Figure 44. Example - Using the Storage Access Routine

Print Service

IPCS now provides this service through the standard print service and the expanded print service. See “Standard Print Service” on page 133 and “Expanded Print Service” on page 116.

You can invoke the print service in one of two ways:

- Obtain the address of the exit services router from the ADPLSERV field of the exit parameter list, BLSABDPL.
- Obtain the address of the print service from the ADPLPRNT field of the exit parameter list, BLSABDPL, and pass control using standard linkage conventions. The following discussion illustrates this method of invoking the print service.

To pass control to the print service using standard linkage conventions register 1 must contain the address of the original exit parameter list address when the print service receives control.

Field ADPLPRNT in the exit parameter list is the address of the print routine that your module can use to write out the buffer area.

Your module must pass control to the print routine using standard linkage conventions. Upon entry to the print routine, register 1 must contain the address of the original exit parameter list.

The print routine causes the output buffer pointed to by the parameter list to be printed and returns a new 133-character buffer (set to blanks) to your module.

The output buffer address points to a 133-character work area where a print line is constructed. This buffer is blank on entry. The carriage control character is provided for you. When a print line is built, the print routine must be given control to cause the line to be printed. To print a blank line, your module must pass control with a blank buffer. If bit 6 in the exit parameter list field ADPLFLAG is on when the print service receives control, the output buffer is printed at the top of the next new page. Bit 6 is turned off when control is returned to the exit routine. Output is based on the message routing parameter used. If ADPLPRT is on then the routing parameter is overridden and output is sent to the terminal.

In the example in Figure 45 on page 174, the subroutine PRINTIT uses the print service subroutine.

```
* THIS SECTION OUTPUTS A LINE OF A TAPE UCB
OUTPUTIT   LA    R0,TAPEL2          SET ADDR OF LINE 1 FORMAT PATTERN
           BAL   RLINK,FORMAT       GO TO FORMAT LINE
           BAL   RLINK,PRINTIT      GO PRINT IT WITH SERV RTN
           BAL   RLINK,PRINTIT      GO PRINT A BLANK LINE
           B     NEXTLINE           GO TO PROCESS NEXT LINE
* THIS SECTION OF CODE IS USED TO CALL THE PRINT SERVICE
* ROUTINE. REGISTER 1 MUST CONTAIN THE ADDRESS OF THE ORIGINAL
* EXIT PARAMETER LIST.
PRINTIT    L     R15,ADPLPRNT       GET PRINT RTN ADR FROM PARM LIST
           BALR  R14,R15            GO TO PRINT ROUTINE
           BR    RLINK              RETURN TO SECTION OF CODE ABOVE
*THIS SUBROUTINE CALLS THE FORMAT SERVICE TO AUTOMATICALLY
*FORMAT AN OUTPUT LINE OF THE UCB.  INPUT TO THIS SECTION OF CODE
*IS THE ADDRESS OF THE FIRST FORMAT PATTERN IN REG 0.  REG 1 MUST
*CONTAIN THE ADDRESS OF THE ORIGINAL EXIT PARAMETER LIST.  IF
*DATA CANNOT BE OBTAINED BY THE SERVICE ROUTINE, CONTROL IS
*PASSED TO AN ERROR SUBROUTINE TO PRINT ERROR MESSAGE
FORMAT     L     R15,ADPLFRMT  GET RTN ADDR FROM PARM LIST
          BALR  R14,R15       GO TO SERVICE ROUTINE TO FORMAT LINE
          LTR   R15,R15       WAS FORMAT SUCCESSFUL
          BCR   8,RLINK       YES, GO PRINT LINE IMMEDIATELY
          B     ERROR         NO, GO TO ERROR ROUTINE
```

Figure 45. Example - Using the Print Service Routine

Summary Dump Data Access Service

The summary dump (SUMDUMP) data access service (IEAVTFRD) allows your formatting routine to access the 31-bit portion of summary dump data contained in an SVC dump. IEAVTFRD is reenterable and can be invoked in 24- or 31-bit addressing mode.

Control is passed to IEAVTFRD with the CALL macro, using standard linkage conventions. Register 1 must contain the address of the exit parameter list. IEAVTFRD uses the ADPLCOM1 field of the exit parameter list during its processing. This word must be set to zero prior to the first invocation of IEAVTFRD, and must not be changed between invocations.

Summary dump data is returned as variable length records. Each record contains a header that describes the data in the record. This header is described by mapping macro IHASMDLR.

Note: Fields SMDLRSTK and SMDLRAST always contain binary zeros in z/OS V1R2 and later.

Summary dump records are sorted on the first invocation of IEAVTFRD. However, only one of the sorted records is returned for each call to IEAVTFRD. The records are returned in sorted address order.

You should invoke IEAVTFRD to read all of the available records, rather than stopping when a particular record has been read.

Storage obtained by IEAVTFRD is not freed until the end of the summary dump data is reached (return code 8 or greater). If all the records are not read, storage remains allocated. The system will free it at the end of the session or earlier, but it may impact other activities until this is done.

When IEAVTFRD returns control to your module, register 0 contains the address of a simulated summary dump record. (This is only feasible for summary dump data in the 31-bit portion of address spaces. Data outside the range is ignored.) Register 15 contains a return code. For nonzero return codes, IEAVTFRD prints an explanatory message in the dump output.

Possible return codes from IEAVTFRD are:

Code
Meaning

00
Register 0 contains the address of a summary dump record.

04
Register 0 contains the address of a partial summary dump record. During summary dump processing, it was necessary to omit parts of this record. All portions of the record beyond the first storage not recorded are replaced with asterisks (X'5C') to maintain original relative displacements.

08
No data is returned. The normal end of the summary dump data has been reached.

12
No data is returned. The end of the summary dump data has been reached before the expected normal end.

16
There is no summary dump data available in the dump.

20
The CVT could not be located in the dump.

28
IEAVTFRD was unable to obtain sufficient storage for its processing.

32
The recovery termination manager control table (RTCT) could not be located in the dump.*

36
The SMWK could not be located in the dump.*

* Information in the SMWK is used to determine whether a summary dump was in progress at the time of a stand-alone dump. The RTCT is used to locate the SMWK.

Note: Return codes 20 through 36 are not returned in z/OS V1R2 or later. No SADUMP support is provided.

Specifying Format Subroutines for Summary Dump Records

Format subroutines for summary dump records are no longer invoked by verb exit SUMDUMP. Alteration of load module IEAVTFSD to cause such routines to be given control is no longer supported.

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

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Documentation with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1)

are mutually exclusive alternatives. If you hear the lines `3.1 USERID` and `3.1 SYSTEMID`, your syntax can include either `USERID` or `SYSTEMID`, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The `*` symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element `*FILE` with dotted decimal number 3 is given the format `3 \* FILE`. Format `3* FILE` indicates that syntax element `FILE` repeats. Format `3* \* FILE` indicates that syntax element `* FILE` repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines `5.1*, 5.1 LASTRUN, and 5.1 DELETE` mean that if you use more than one of the `LASTRUN` and `DELETE` syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line `2.1 %OP1` means that you must refer to separate syntax fragment `OP1`.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example `5? NOTIFY`). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines `5 ?, 5 NOTIFY`, and `5 UPDATE`, you know that the syntax elements `NOTIFY` and `UPDATE` are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the `!` symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines `2? FILE, 2.1! (KEEP), and 2.1 (DELETE)`, you know that `(KEEP)` is the default option for the `FILE` keyword. In the example, if you include the `FILE` keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the `FILE` keyword is omitted, the default `FILE(KEEP)` is used. However, if you hear the lines `2? FILE, 2.1, 2.1.1! (KEEP)`, and `2.1.1 (DELETE)`, the default option KEEP applies only to the next higher dotted decimal number, `2.1` (which does not have an associated keyword), and does not apply to `2? FILE`. Nothing is used if the keyword `FILE` is omitted.

*** indicates an optional syntax element that is repeatable**
The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line `5.1* data area`, you know that you can include one data area, more than one data area, or no data area. If you hear the lines `3* , 3 HOST, 3 STATE`, you know that you can include `HOST, STATE`, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write `HOST STATE`, but you cannot write `HOST HOST`.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line `6.1+ data area`, you must include at least one data area. If you hear the lines `2+, 2 HOST,` and `2 STATE`, you know that you must include `HOST, STATE,` or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, 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

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming Interface Information

This book primarily documents information that is NOT intended to be used as Programming Interfaces of z/OS.

This book also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of z/OS. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

Trademarks

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Product Number: 5650-ZOS

SA23-1383-40