IMS

Base Primitive Environment Guide and Reference

Version 9

ZES1-2337-01

IMS

Base Primitive Environment Guide and Reference

Version 9

ZES1-2337-01

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 85.

Quality Partnership Program (QPP) Edition (June 2004) (Softcopy Only)

This QPP edition replaces or makes obsolete the previous edition, ZES1-2337-00. This edition is available in softcopy format only. The technical changes for this version are summarized under "Summary of Changes" on page xiii.

Contents

Figures

Tables

About This Book

This information is available in PDF and BookManager® formats, and also as part of the IMS™ Version 9 QPP Information Center. To get the most current versions of the PDF and BookManager formats, go to the IMS Library page at www.ibm.com/software/data/ims/library.html. To get the most current versions of these books for the information center, go to the IMS V9 Vendor and Quality Partnership Program Library page at www6.software.ibm.com/dl/ims02/imsv9lib-p, where you can find updated plug-ins and instructions on how to install them in your IMS Version 9 QPP Information Center.

This book is designed to help programmers, operators, and system support personnel use the IMS Base Primitive Environment (BPE) external interfaces. BPE is a common system service base upon which Common Queue Service (CQS) and the Common Service Layer (CSL) components Operations Manager (OM), Resource Manager (RM), and the Structured Call Interface (SCI) are built.

Terminology and Related Publications

For a list of related publications, refer to "Bibliography" on page 89.

For definitions of terminology used in this manual and references to related information in other manuals, see the *IMS Version 9: Master Index and Glossary*.

How to Read Syntax Diagrams

Each syntax diagram in this book begins with a double right arrow and ends with a right and left arrow pair. Lines that begin with a single right arrow are continuation lines. You read a syntax diagram from left to right and from top to bottom, following the direction of the arrows.

Table 1 describes the conventions that are used in syntax diagrams in this information:

Table 1. How to Read Syntax Diagrams

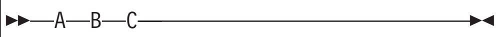
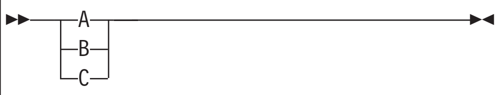
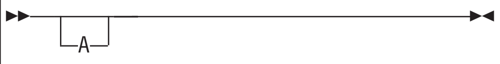
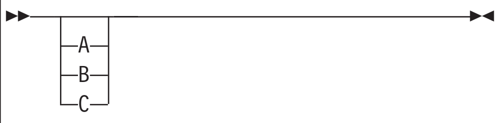
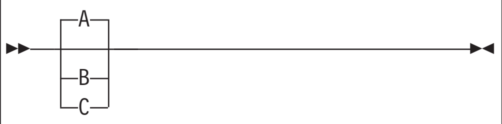
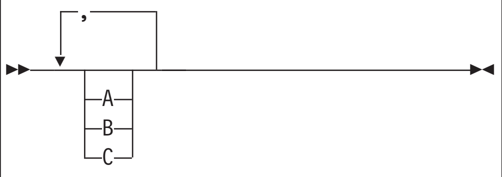
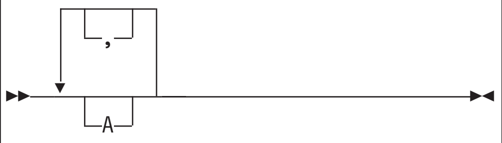
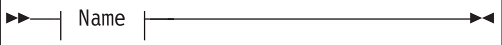
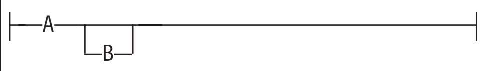
Convention	Meaning
	You must specify values A, B, and C. Required values are shown on the main path of a syntax diagram.
	You must specify value A, B, or C.
	You have the option to specify value A. Optional values are shown below the main path of a syntax diagram.
	You have the option to specify A, B, C, or none of these values.

Table 1. How to Read Syntax Diagrams (continued)

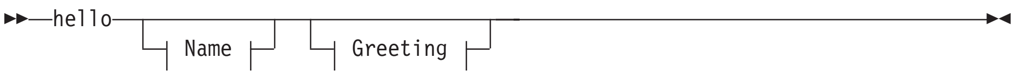
Convention	**Meaning**
	You have the option to specify `A`, `B`, `C`, or none of these values. If you don't specify a value, `A` is the default.
	You have the option to specify one, more than one, or none of the values `A`, `B`, or `C`. Any required separator for multiple or repeated values (in this example, the comma) is shown on the arrow.
	You have the option to specify value `A` multiple times. The separator in this example is optional.
 **Name:** 	Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.
Punctuation marks and numbers	Enter punctuation marks (slashes, commas, periods, parentheses, quotation marks, equal signs) and numbers exactly as shown.
Uppercase values	Keywords, their allowable synonyms, and reserved parameters appear in uppercase letters for z/OS. Enter these values exactly as shown.
Lowercase values	Keywords, their allowable synonyms, and reserved parameters appear in lowercase letters for UNIX. Enter these values exactly as shown.
Lowercase values in italic (for example, *name*)	Supply your own text or value in place of the *name* variable.
ƀ	A ƀ symbol indicates one blank position.

Other syntax conventions include the following:

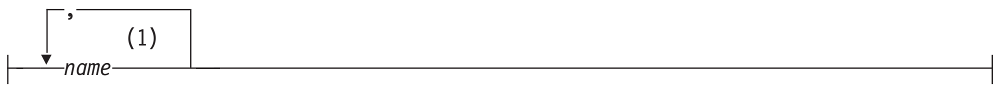
- When you enter commands, separate parameters and keywords by at least one blank if there is no intervening punctuation.
- Footnotes are shown by a number in parentheses, for example, (1).
- Parameters with number values end with the symbol #.
- Parameters that are names end with 'name'.
- Parameters that can be generic end with the symbol *.

Syntax Diagram Example

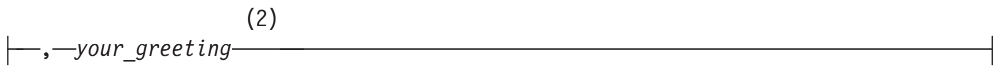
Here is an example syntax diagram that describes the `hello` command.



Name:



Greeting:



Notes:

1 You can code up to three names.

2 Compose and add your own greeting (for example, how are you?).

According to the syntax diagram, these commands are all valid versions of the `hello` command:

```
hello
hello name
hello name, name
hello name, name, name
hello, your_greeting
hello name, your_greeting
hello name, name, your_greeting
hello name, name, name, your_greeting
```

The space before the *name* value is significant. If you do not code *name*, you must still code the comma before *your_greeting*.

How to Send Your Comments

Your feedback is important in helping us provide the most accurate and highest quality information. If you have any comments about this book or any other IMS documentation, you can do one of the following:

- Go to the IMS home page at www.ibm.com/ims. There you will find an online feedback page where you can enter and submit comments.
- Send your comments by e-mail to imspubs@us.ibm.com. Be sure to include the name of the book, the part number of the book, the version of IMS, and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

How to Send Your Comments

Your feedback is important in helping us provide the most accurate and highest quality information. If you have any comments about this or any other IMS information, you can do one of the following:

- Go to the IMS Library page at www.ibm.com/software/data/ims/library.html and click the Library Feedback link, where you can enter and submit comments.
- Send your comments by e-mail to imspubs@us.ibm.com. Be sure to include the title, the part number of the title, the version of IMS, and, if applicable, the specific location of the text you are commenting on (for example, a page number in the PDF or a heading in the Information Center).

Summary of Changes

Changes to the Current® Edition of This Book for IMS Version 9

| This edition contains technical and editorial changes.

| In this edition, information on IMS Connect, specifically IMS Connect Trace Table
| types, has been added to “IMS Connect Trace Table Types” on page 11.

Changes to This Book for IMS Version 9

This edition is a draft version of the book intended for use during the Quality Partnership Program (QPP). Contents of this book are preliminary and under development. This softcopy book is available only in PDF and BookManager formats.

Library Changes for IMS Version 9

Changes to the IMS Library for IMS Version 9 include the addition of new titles, the change of one title, and a major terminology change. Changes are indicated by a vertical bar (|) to the left of the changed text.

New and Revised Titles

The following list details the major changes to the IMS Version 9 library:

- *IMS Version 9: HALDB Online Reorganization Guide*

  The library includes new information: *IMS Version 9: HALDB Online Reorganization Guide*. This information is available only in PDF and BookManager formats.
- *IMS Version 9: An Introduction to IMS*

  The library includes new information: *IMS Version 9: An Introduction to IMS*.
- The information formerly titled *IMS Version 8: IMS Java User’s Guide* is now titled *IMS Version 9: IMS Java Guide and Reference*.
- The library includes new information: *IMS Version 9: IMS Connect Guide and Reference*. This information is available only in PDF and BookManager formats.

Terminology Changes

IMS Version 9 introduces new terminology for IMS commands:

type-1 command
A command, generally preceded by a leading slash character, that can be entered from any valid IMS command source. In IMS Version 8, these commands were called *classic* commands.

type-2 command
A command that is entered only through the OM API. Type-2 commands are more flexible and can have a broader scope than type-1 commands. In IMS Version 8, these commands were called *IMSplex* commands or *enhanced* commands.

Accessibility Enhancements

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products. The major accessibility features in z/OS products, including IMS, enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

User Assistive Technologies

Assistive technology products, such as screen readers, function with the IMS user interfaces. Consult the documentation of the assistive technology products for specific information when you use assistive technology to access these interfaces.

Accessible Information

Online information for IMS Version 9 is available in BookManager format, which is an accessible format. All BookManager functions can be accessed by using a keyboard or keyboard shortcut keys. BookManager also allows you to use screen readers and other assistive technologies. The BookManager READ/MVS product is included with the z/OS base product, and the BookManager Softcopy Reader (for workstations) is available on the IMS Licensed Product Kit (CD), which you can download from the Web at www.ibm.com.

Keyboard Navigation of the User Interface

Users can access IMS user interfaces using TSO/E or ISPF. Refer to the *z/OS V1R1.0 TSO/E Primer*, the *z/OS V1R1.0 TSO/E User's Guide*, and the *z/OS V1R1.0 ISPF User's Guide, Volume 1*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Chapter 1. Introduction to Base Primitive Environment

The IMS Base Primitive Environment (BPE) is a common system service base upon which many other IMS components are built. BPE provides services such as tracing, message formatting, parsing, storage management, sub-dispatching, and serialization. In IMS Version 9, the following components use BPE:

- Common Queue Server (CQS)
- Operations Manager (OM)
- Resource Manager (RM)
- Structured Call Interface (SCI)

When an IMS component that uses BPE is started, the component loads a copy of the BPE service modules into its address space from the IMS Version 9 program libraries. The IMS component's modules are specific to that component; however, the BPE service modules are common across the various address spaces. The base system service functions are therefore identical in every address space that uses BPE. Figure 1 on page 2 shows the relationship of BPE, IMS components, and IMS program libraries.

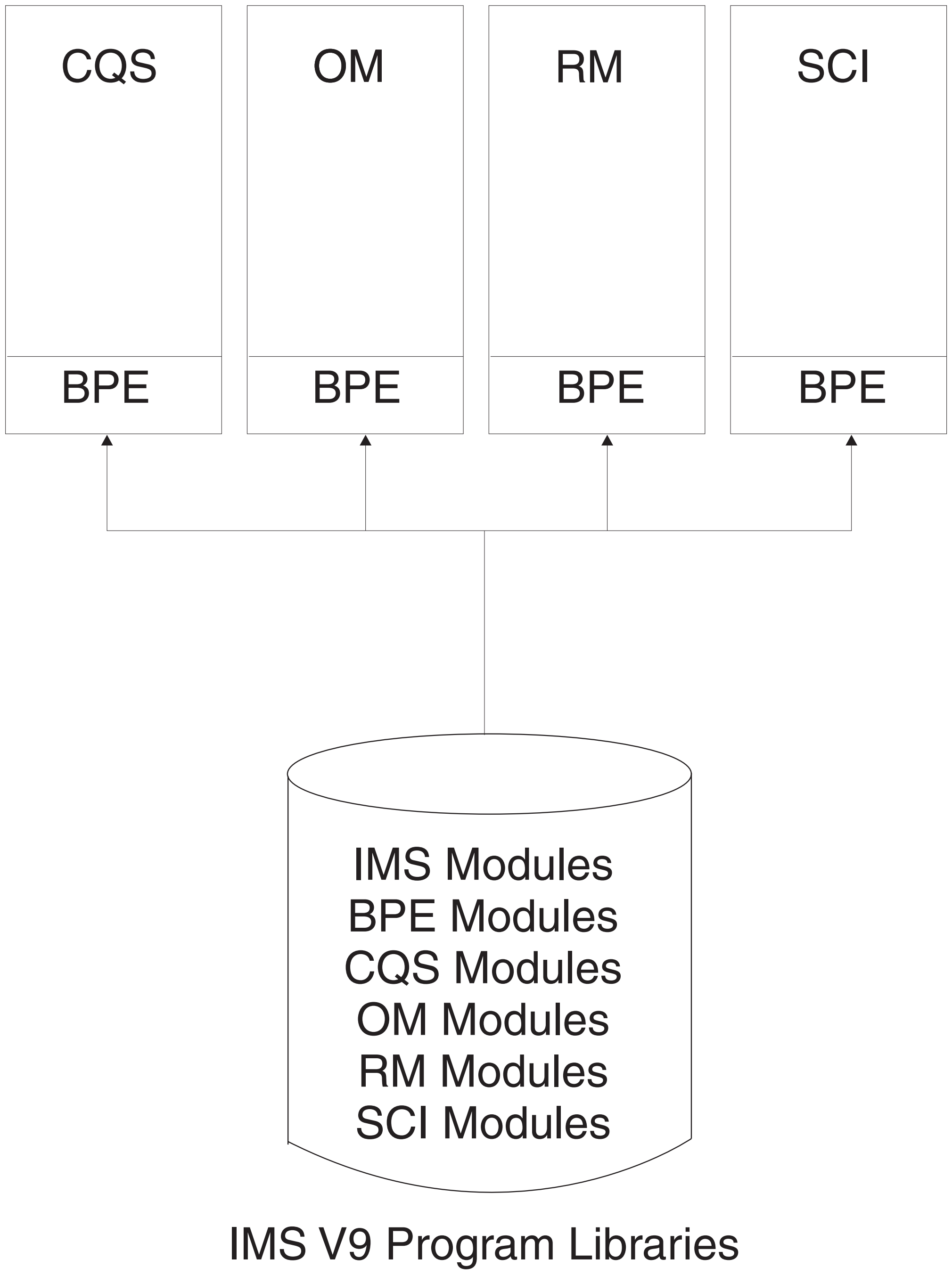


Figure 1. BPE and IMS Components

Most of the time, BPE is a hidden layer in an IMS component address space. However, you can use the following external interfaces to BPE:

Configuration
: You can configure certain attributes about an address space that uses BPE at startup by using statements in BPE PROCLIB members. For example, you can set the default level and size for BPE-managed trace tables.

Commands
: You can use the small set of commands that BPE provides to operate on BPE-managed resources. For example, you can display and change attributes of BPE-managed user exit routines and trace tables.

User exit routines
You can customize the operation of an IMS component address space by creating a user exit routine. Components running with BPE can use the BPE user exit service to call user-supplied exit routines. BPE also has two user exit routines of its own. User exit routines that are called through BPE receive control with a standard BPE user exit routine interface and are allowed to use BPE user exit routine callable services.

Messages and Abends
BPE has its own messages and abend codes. The *IMS Version 9: Messages and Codes, Volume 1* and *IMS Version 9: Messages and Codes, Volume 2* manuals document all BPE messages and abend codes.

This manual describes how to use these BPE external interfaces: configuration, commands, and user exit routines.

Chapter 2. BPE Definition and Tailoring

This chapter describes the tasks of defining and tailoring BPE settings for an IMS component using BPE.

In this chapter:

This chapter contains Product-sensitive Programming Interface information.

General BPE PROCLIB Member Information

You can specify the settings of several BPE runtime parameters through the use of BPE PROCLIB members. For example, you can set the level of BPE and IMS component trace tables, and you can associate user exit routine modules with an IMS component user exit routine type. The following rules apply to the format of all BPE PROCLIB members:

- The PROCLIB data set should have an LRECL of at least nine (80 is typical) and a fixed record format.
- The rightmost eight columns of each record are ignored, and you can use them for sequence numbers or any other notation. In the remaining columns, you code the keyword parameters. For example, if your record size is 80, you use columns 1 through 72 for your configuration data. You can use columns 73 through 80 for sequence numbers.
- Keywords can contain leading and trailing blanks.
- You can specify multiple keywords in each record.
- Use commas or spaces to delimit keywords.
- Use an asterisk (*) or pound sign (#) in column one to begin a comment. You can also include a comment anywhere within a statement by enclosing it between a slash-asterisk and an asterisk-slash pair. Comments between slash-asterisk and asterisk-slashes may span multiple lines.

  Example:

  ```
  /*This is an example of a comment within a statement*/
  ```
- Statements may be continued across multiple lines by breaking the statement at a word boundary and continuing the statement on the next line.

  Example:

  ```
  TRCLEV=(AWE,
  HIGH,BPE)
  ```
- Values coded in this PROCLIB member are case-sensitive. Use uppercase characters for all PROCLIB members.

BPE Configuration Parameter PROCLIB Member

You can use the BPE configuration parameter PROCLIB member to define BPE execution environment settings for the address space being started. Specify the PROCLIB member name by coding `BPECFG=`*member_name* on the EXEC PARM= statement in the address space startup JCL, as shown in this example:

```
EXEC CQSINIT0,PARM='BPECFG=BPECFGCQ'
```

You can use the BPE configuration parameter PROCLIB member to specify the following items:

- The language used for BPE and IMS component messages
- The trace level settings for BPE and IMS component internal trace tables
- The name of a BPE exit list PROCLIB member where configuration information for IMS component user exit routine routines is stored
- The time interval between calls to the BPE statistics exit routines

These are the keywords that are available for the BPE configuration parameter PROCLIB member:

- LANG=
- TRCLEV=
- EXITMBR=
- STATINTV=

Recommendation: Avoid coding statements in the BPE configuration member that specify definitions for the same resources multiple times. For example, avoid multiple TRCLEV statements for the same trace table type, or multiple EXITMBR statements for the same IMS component type. BPE uses the **last** statement it encounters in the member. Any values that are specified on earlier duplicate statements are ignored. A message, BPE0017I is issued for each duplicate statement found.

BPECFG= LANG Parameter

```
►►──LANG=ENU──────────────────────────────────────────────────────►◄
```

The LANG parameter specifies the language used for BPE and IMS component messages. ENU is for US English, which is currently the only supported language.

BPECFG= TRCLEV Parameter

```
►►──TRCLEV=──(type,level,ims_component──┬──────────────────┬─)────►◄
                                        └─,PAGES=num_pages─┘
```

The TRCLEV parameter specifies the trace level for a trace table and, optionally, the number of storage pages allocated for the trace table. `TRCLEV=` controls the level of tracing (the amount of detail traced) for each specified trace table type. BPE-managed trace tables are areas in storage where BPE, and the IMS component using BPE, can trace diagnostic information about events occurring within the address space.

BPE-managed trace tables are internal in-core tables only. Trace records are not written to any external data sets. Some trace table types are defined and owned by BPE itself. These are known as *system trace tables*, and are present in all IMS component address spaces that use BPE. The IMS component can also define its own trace tables. These are known as component trace tables or user-product trace tables, and are only present in address spaces of the defining IMS component. For example, trace table types defined by Common Queue Server (CQS) are only present in a CQS address space.

You can share one BPE configuration parameter PROCLIB member among several different IMS component address spaces. Any TRCLEV statements you code for system trace tables apply identically to all of the address spaces that share the PROCLIB member. TRCLEV statements for a particular IMS component trace table are processed only by address spaces running that component. For example, you could have a BPE configuration parameter PROCLIB member containing TRCLEV statements for BPE, CQS, and Resource Manager (RM) trace table types. When you start a CQS address space, only the BPE and CQS TRCLEV statements are processed. When you start an RM address space, only the BPE and TRCLEV statements are processed.

type

Specifies the type of trace table. Each trace table has a four-character type. A trace table's type refers to the kind of events that are traced into that table. For example, the BPE DISP trace table contains entries related to events in the BPE dispatcher. Refer to the type tables in "BPE Trace Table Types" on page 8 for a list of valid types for each different IMS component address space.

level

Controls how much tracing is done in the specified trace table. Each trace entry that is made has a level associated with the entry. Each trace table has a level setting controlled by the value for *level* that you specify on the TRCLEV statement for the table.

A trace entry is written only if the trace entry's level is less than or equal to the table's level setting. For example, if the trace entry level is MEDIUM, the trace | entry is added to the trace table only if the table's level is MEDIUM or HIGH. Thus, the *level* you specify controls the volume (number) of trace entries that are written to a given table.

A low setting of the *level* parameter results in fewer trace entries being made to the table. The trace table does not wrap as quickly as with a higher setting (which means that diagnostic information remains available for a longer period of time), and the performance impact is minimized. However, the trace information is not as detailed as with higher settings, so the captured information might not be sufficient to solve a problem.

A high setting of the *level* parameter results in more trace entries being written to the table. This can provide additional diagnostic information for solving a problem; however, the trace table tends to wrap more frequently, and higher settings can cause additional CPU usage.

Choose one of the following for the *level* parameter:

NONE

No tracing.

Recommendation: Do not specify NONE because no tracing, not even tracing for error conditions, is done for the specified table.

ERROR

Only trace entries for error conditions are made. ERROR is the default.

LOW

Low-volume tracing (key component events). This is the minimum recommended trace level setting for normal operation.

MEDIUM

Medium-volume tracing (most component events).

HIGH

High-volume tracing (all component events).

ims_component
Specifies the IMS component that defines the trace table type. Possible values are:

BPE
Indicates that the table is a BPE-defined (system) trace table. BPE trace tables exist in all IMS component address spaces that run with BPE.

CQS
Indicates that the table is a Common Queue Server-defined trace table type.

| **HWS**
| Indicates that the table is an IMS Connect-defined trace table type.

OM
Indicates that the table is an Operations Manager-defined trace table type.

RM
Indicates that the table is a Resource Manager-defined trace table type.

SCI
Indicates that the table is a Structured Call Interface-defined trace table type.

PAGES=*num_pages*
An optional parameter that can be used to specify the number of 4 KB pages to be allocated for the table type.

Specify a value from 1 to 32767 pages for this parameter. If BPE is unable to get the amount of storage you requested for a trace table, it will try to get a smaller number of pages to enable some tracing to be done. You can see the actual number of pages BPE obtained for each trace by issuing the `DISPLAY TRACETABLE` command.

Related Reading: For more information about the `DISPLAY TRACETABLE` command, see "BPE TRACETABLE Commands" on page 27.

If you do not use this parameter, then the trace table has the default number of pages, as specified under the description of each trace table type.

The trace table types that are provided by the various IMS components are described in:

- "BPE Trace Table Types"
- "CQS Trace Table Types" on page 10
- | "IMS Connect Trace Table Types" on page 11
- "OM Trace Table Types" on page 12
- "RM Trace Table Types" on page 13
- "SCI Trace Table Types" on page 14

BPE Trace Table Types

BPE provides a set of trace table types for tracing processing within BPE functions. These BPE trace table types are present in all IMS component address spaces. | TRCLEV statements specifying a component of BPE are processed for all IMS component address space types.

*****

Specifying a type of ***** enables you to set the default trace level (and, optionally, the default number of pages per trace table) for **all** BPE-defined trace table types. If you use the ***** type, make sure it is the first TRCLEV statement for

BPE-defined trace table types in your PROCLIB member. You can then code additional TRCLEV statements for specific BPE types to selectively override the defaults.

Recommendation: Code a TRCLEV statement with a type of ***** for BPE traces, specifying a level of at least LOW as your first TRCLEV statement for BPE-defined trace table types. Using this coding ensures that at least some tracing is done for all BPE trace tables. Specifying a TRCLEV of * also ensures that any new trace table types that are activated in the future are turned on in your system, even if you have not modified your BPE configuration parameter PROCLIB member to explicitly add a TRCLEV statement.

AWE

The asynchronous work element (AWE) services trace table shows details of AWE server creation and deletion and AWE processing requests.

The default number of pages for this table is 6.

CBS

The control block services trace table traces requests for control block storage.

The default number of pages for this table is 6.

CMD

The command trace table traces the first 48 characters of each command processed by BPE.

The default number of pages for this table is 2.

DISP

The dispatcher trace table traces BPE dispatcher activity.

The default number of pages for this table is 8.

ERR

The error trace table traces error events within a BPE address space.

The default number of pages for this table is 2.

Restriction: You cannot set the level for the ERR trace table. BPE forces the level to HIGH to ensure that error diagnostics are captured. Any level that you specify for the ERR trace table is ignored. You can, however, specify the number of pages for the ERR trace table on the TRCLEV statement.

HASH

The hash trace table traces events related to BPE hash table services. Currently, only OM, RM, and SCI address spaces request hash table services. For address spaces that do not use the BPE hash table services, this TRCLEV statement is ignored.

The default number of pages for this table is 8.

LATC

The latch trace table traces BPE latch management (serialization) activity. The default number of pages for this table is 8.

SSRV

The system services trace table traces general BPE system service calls.

The default number of pages for this table is 4.

STG

The storage service trace table traces storage service requests.

The default number of pages for this table is 8.

USRX

The user exit routine trace table traces activity related to exit routines (for example, loads, calls, or abends).

The default number of pages for this table is 4.

CQS Trace Table Types

CQS provides a set of trace table types for tracing processing within the CQS address space. These CQS trace table types are present only in a CQS address space. TRCLEV statements specifying a component of CQS is ignored for any other address space type.

*****

Specifying a type of ***** enables you to set the default trace level (and, optionally, the default number of pages per trace table) for **all** CQS-defined trace table types. If you use the ***** type, make sure it is the first TRCLEV statement for CQS-defined trace table types in your PROCLIB member. You can then code additional TRCLEV statements for specific CQS types to selectively override the defaults.

Recommendation: Code a TRCLEV statement with a type of ***** for CQS traces, specifying a level of at least LOW as your first TRCLEV statement for CQS-defined trace table types. Using this coding ensures that at least some tracing is done for all CQS trace tables. It also ensures that any new trace table types that are added in the future are turned on in your system, even if you have not modified your BPE configuration parameter PROCLIB member to explicitly add a TRCLEV statement.

CQS

The CQS trace table traces general activity that is not related to a specific structure.

The default number of pages for this table is 4.

ERR

The error trace table traces error events within a CQS address space.

The default number of pages for this table is 4.

Restriction: You cannot set the level for the ERR trace table. BPE forces the level to HIGH to ensure that error diagnostics are captured. Any level that you specify for the ERR trace table is ignored. You can, however, specify the number of pages for the ERR trace table on the TRCLEV statement.

INTF

The interface trace table traces activity in the interface between a CQS and its client.

The default number of pages for this table is 8.

STR

The structure trace table traces activity related to a structure. CQS defines one STR trace table for each structure pair defined to CQS.

The default number of pages for this table is 8.

IMS Connect Trace Table Types

IMS Connect provides a set of trace table types for tracing processing within IMS Connect functions. IMS Connect defines its own trace tables. These tables are known as component trace tables or user-product trace tables. You can code the following values for IMS Connect-defined trace tables:

*****
Specify as `TRCLEV=(*,`*`level`*`,HWS)`.

Specifying a type of ***** enables you to set the default trace level (and optionally, the default number of pages per trace table) for **all** IMS Connect-defined trace table types. If you use the ***** type, make sure it is the first TRCLEV statement for IMS Connect-defined trace table types in your PROCLIB member. You can then code additional TRCLEV statements for specific IMS Connect types to selectively override the defaults.

Recommendation: Code a TRCLEV statement with a type of ***** for IMS Connect traces, specifying a level of at least LOW as your first TRCLEV statement for IMS Connect-defined trace table types. Using this coding ensures that at least some tracing is done for all IMS Connect trace tables. It also ensures that any new trace table types that are added in the future will be turned on in your system, even if you have not modified your IMS Connect BPE configuration parameter PROCLIB member to explicitly add a TRCLEV statement.

CMDT
Specify as `TRCLEV=(CMDT,`*`level`*`,HWS)`.

The command trace table traces IMS Connect command activity. The default number of pages for this table is 2.

ENVT
Specify as `TRCLEV=(ENVT,`*`level`*`,HWS)`.

The interface trace table traces activity in the interface between an IMS Connect and its client. The default number of pages for this table is 2.

HWSI
Specify as `TRCLEV=(HWSI,`*`level`*`,HWS)`.

The IMS Connect to OTMA driver trace table traces communication activity between IMS Connect and OTMA drivers. The default number of pages for this table is 2.

HWSN
Specifies as `TRCLEV=(HWSN,`*`level`*`,HWS)`

The IMS Connect to local option driver trace table traces communication activity and event between local option driver and IMS Connect. The default number of pages for this table is 2.

HWSW
Specify as `TRCLEV=(HWSW,`*`level`*`,HWS)`.

The IMS Connect to TCP/IP driver trace table traces communication activity and events between TCP/IP drivers and IMS Connect. The default number of pages for this table is 2.

OTMA
Specify as `TRCLEV=(OTMA,`*`level`*`,HWS)`.

The OTMA communication driver trace table traces internal communication protocol activity (XCF calls). The default number of pages for this table is 2.

PCDR
Specifies as `TRCLEV=(PCDR,`*level*`,HWS)`

The local option driver trace table traces local option communication protocol activity. The default number of pages for this table is 2.

TCPI
Specify as `TRCLEV=(TCPI,`*level*`,HWS)`.

The TCP/IP communication driver trace table traces communication protocol activity (TCP/IP calls). The default number of pages for this table is 2.

OMDR
Specify as `TRCLEV=(OMDR,`*level*`,HWS)`.

The IMSplex driver (IPDC) trace table traces communication protocol activity (SCI calls). The default number of pages for this table is 2.

HWSO
Specify as `TRCLEV=(HWSO,`*level*`,HWS)`.

The IMSplex driver (IPDC) trace table traces communication activity and events between the IMSplex driver and IMS Connect. The default number of pages for this table is 2.

OM Trace Table Types

OM provides a set of trace table types for tracing processing within the OM address space. These OM trace table types are present only in an OM address space. TRCLEV statements specifying a component of OM are ignored for any other address space type.

*****

Specifying a type of ***** enables you to set the default trace level (and, optionally, the default number of pages per trace table) for **all** OM-defined trace table types. If you use the ***** type, make sure it is the first TRCLEV statement for OM-defined trace table types in your PROCLIB member. You can then code additional TRCLEV statements for specific OM types to selectively override the defaults.

Recommendation: Code a TRCLEV statement with a type of ***** for OM traces, specifying a level of at least LOW as your first TRCLEV statement for OM-defined trace table types. Using this coding ensures that at least some tracing is done for all OM trace tables. It also ensures that any new trace table types that are added in the future are turned on in your system, even if you have not modified your BPE configuration parameter PROCLIB member to explicitly add a TRCLEV statement.

CSL

The Common Service Layer (CSL) trace table is used for routines that are common to all common service layer managers.

The default number of pages for this table is 4.

ERR

The error trace table traces error events within an OM address space.

The default number of pages for this table is 4.

Restriction: You cannot set the level for the ERR trace table. BPE forces the level to HIGH to ensure that error diagnostics are captured. Any level that you specify for the ERR trace table is ignored. You can, however, specify the number of pages for the ERR trace table on the TRCLEV statement.

OM

The Operations Manager (OM) trace table traces events related to general OM processes.

The default number of pages for this table is 4.

PLEX

The IMSplex trace table traces OM processing for a specific IMSplex.

The default number of pages for this table is 8.

RM Trace Table Types

RM provides a set of trace table types for tracing processing within the RM address space. These RM trace table types are present only in a RM address space. TRCLEV statements specifying a component of RM are ignored for any other address space type.

*****

Specifying a type of ***** enables you to set the default trace level (and, optionally, the default number of pages per trace table) for **all** RM-defined trace table types. If you use the ***** type, make sure it is the first TRCLEV statement for RM-defined trace table types in your PROCLIB member. You can then code additional TRCLEV statements for specific RM types to selectively override the defaults.

Recommendation: Code a TRCLEV statement with a type of ***** for RM traces, specifying a level of at least LOW as your first TRCLEV statement for RM-defined trace table types. Using this coding ensures that at least some tracing is done for all RM trace tables. It also ensures that any new trace table types that are added in the future are turned on in your system, even if you have not modified your BPE configuration parameter PROCLIB member to explicitly add a TRCLEV statement.

CSL

The Common Service Layer (CSL) trace table is used for routines that are common to all common service layer managers.

The default number of pages for this table is 4.

ERR

The error trace table traces error events within an RM address space.

The default number of pages for this table is 4.

Restriction: You cannot set the level for the ERR trace table. BPE forces the level to HIGH to ensure that error diagnostics are captured. Any level that you specify for the ERR trace table is ignored. You can, however, specify the number of pages for the ERR trace table on the TRCLEV statement.

RM

The Resource Manager (RM) trace table traces events related to general RM processes.

The default number of pages for this table is 4.

PLEX

The IMSplex trace table traces RM processing for a specific IMSplex.

The default number of pages for this table is 8.

SCI Trace Table Types

SCI provides a set of trace table types for tracing processing within the SCI address space. These SCI trace table types are present only in a SCI address space. TRCLEV statements specifying a component of SCI are ignored for any other address space type.

*****

Specifying a type of ***** enables you to set the default trace level (and, optionally, the default number of pages per trace table) for **all** SCI-defined trace table types. If you use the ***** type, make sure it is the first TRCLEV statement for SCI-defined trace table types in your PROCLIB member. You can then code additional TRCLEV statements for specific SCI types to selectively override the defaults.

Recommendation: Code a TRCLEV statement with a type of ***** for SCI traces, specifying a level of at least LOW as your first TRCLEV statement for SCI-defined trace table types. Using this coding ensures that at least some tracing is done for all SCI trace tables. It also ensures that any new trace table types that are added in the future are turned on in your system, even if you have not modified your BPE configuration parameter PROCLIB member to explicitly add a TRCLEV statement.

CSL

The Common Service Layer (CSL) trace table is used for routines that are common to all common service layer address spaces.

The default number of pages for this table is 8.

ERPL

The error parameter list trace table traces a copy of the interface parameter list when an error occurs processing an SCI request or message.

The default number of pages for this table is 8.

ERR

The error trace table traces error events within an SCI address space.

The default number of pages for this table is 4.

Restriction: You cannot set the level for the ERR trace table. BPE forces the level to HIGH to ensure that error diagnostics are captured. Any level that you specify for the ERR trace table is ignored. You can, however, specify the number of pages for the ERR trace table on the TRCLEV statement.

INTF

The interface trace table traces IMSplex member interface activity (requests and messages).

The default number of pages for this table is 8.

INTP

The interface parameter trace table traces a copy of the interface parameter list during SCI request and message processing.

The default number of pages for this table is 16.

PLEX

The IMSplex trace table traces SCI processing for a specific IMSplex.

The default number of pages for this table is 8.

SCI

The Structured Call Interface (SCI) trace table traces events related to general SCI processes.

The default number of pages for this table is 8.

BPECFG= EXITMBR Parameter

```
▶▶──EXITMBR=(member_name,ims_component)──────────────────────────────────▶◀
```

The EXITMBR parameter specifies the exit list PROCLIB member name. You can specify one EXITMBR= parameter for each IMS component running with BPE, and one EXITMBR= parameter for BPE itself.

Related Reading: For more information about the BPE exit list PROCLIB member, see "BPE Exit List PROCLIB Member" on page 18.

member_name
Specifies the eight character exit list PROCLIB member name.

ims_component
Specifies the IMS component whose user exit routine modules are being defined. Possible values are:

BPE
Indicates the BPE exit routine PROCLIB member name.

CQS
Indicates the Common Queue Server exit routine PROCLIB member name.

OM
Indicates the Operations Manager (OM) exit routine PROCLIB member name.

RM
Indicates the Resource Manager (RM) exit routine PROCLIB member name.

SCI
Indicates the Structured Call Interface (SCI) exit routine PROCLIB member name.

BPECFG= STATINTV Parameter

```
▶▶──STATINTV=(number_of_seconds)─────────────────────────────────────────▶◀
```

The optional STATINTV parameter specifies the time interval, in seconds, between calls to the BPE statistics exit or exit routines. You can set STATINTV from 1 to 2147483647 (2^{31}-1). The default STATINTV value is 600 (ten minutes).

Recommendation: Specify a STATINTV value of 60 or more to avoid possible performance problems due to frequent exit routine calls.

Sample BPE Configuration File

A sample BPE configuration data set is shown in Figure 2 on page 17. This example shows a BPE configuration data set that can be shared by CQS and CSL address spaces. It contains definitions for traces for:

- BPE system
- CQS
- OM
- RM
- SCI

It also contains user exit list PROCLIB member specifications.

```
***************************************************************
*  CONFIGURATION FILE FOR BPE WITH CSL and CQS ADDRESS SPACES  *
***************************************************************

LANG=ENU                          /* Language for messages       */
                                  /* (ENU = U.S. English)        */
STATINTV=420                      /* STATS user exit interval    */
                                  /* = 420 seconds (7 minutes)   */

#
# Definitions for BPE system traces
#

TRCLEV=(*,LOW,BPE)                /* Set default for all BPE     */
                                  /* traces to LOW.              */
TRCLEV=(AWE,HIGH,BPE)             /* AWE server trace on high    */
TRCLEV=(CBS,MEDIUM,BPE)           /* Ctrl blk serv trc on medium */
TRCLEV=(DISP,HIGH,BPE,PAGES=12)   /* Dispatcher trace on high    */
                                  /* with 12 pages               */

#
# Definitions for CQS traces
#

TRCLEV=(*,MEDIUM,CQS)             /* Set default for all CQS     */
                                  /* traces to medium.           */
TRCLEV=(STR,HIGH,CQS)             /* But run STR trace on high   */

#
# Definitions for OM traces
#

TRCLEV=(*,MEDIUM,OM)              /* Set default for all OM      */
                                  /* traces to medium            */

#
# Definitions for RM traces
#

TRCLEV=(*,MEDIUM,RM)              /* Set default for all RM      */
                                  /* traces to medium            */

#
# Definitions for SCI traces
#

TRCLEV=(*,MEDIUM,SCI)             /* Set default for all SCI     */
                                  /* traces to medium            */
TRCLEV=(INTF,HIGH,SCI)            /* Intf call trace on high     */
TRCLEV=(INTP,HIGH,SCI)            /* Intf parmlist trace on high */

#
# User exit list PROCLIB member specifications
#

EXITMBR=(BPEEXIT0,BPE)            /* BPE user exit definitions   */
EXITMBR=(CQSEXIT0,CQS)            /* CQS user exit definitions   */
EXITMBR=(OMEXIT00,OM)             /* OM user exit definitions    */
EXITMBR=(RMEXIT00,RM)             /* RM user exit definitions    */
EXITMBR=(SCIEXIT0,SCI)            /* SCI user exit definitions   */
```

Figure 2. Example of a Configuration File for BPE with OM, RM, SCI, and CQS

BPE Exit List PROCLIB Member

Use the PROCLIB members specified by the EXITMBR= parameter in the BPE configuration parameter PROCLIB member to define user exit routine modules to BPE. BPE Exit List PROCLIB members are IMS-component specific. You specify one EXITMBR statement for each IMS component that provides user exit routines through BPE services. Each EXITMBR statement specifies the name of a PROCLIB member that contains the definitions for exit routines for that IMS component. You can have a separate exit list PROCLIB member for each IMS component, or you can share one exit list PROCLIB member among several IMS components.

A BPE exit list PROCLIB member associates a user exit routine type with a list of one or more user exit routine modules. Use the EXITDEF statement to define the exit routine modules to be called for a particular exit routine type. The BPE exit list PROCLIB member is processed by BPE during address space initialization. It is also processed when you enter a `REFRESH USEREXIT` command (see “BPE USEREXIT Commands” on page 35 for more information about BPE USEREXIT commands).

Recommendation: Avoid coding statements in the BPE exit routine list member that specify definitions for the same exit routine type multiple times. BPE always uses the **last** statement it encounters in the member for a particular exit routine. Any earlier statements for the same exit routine are ignored. A message, BPE0017I is issued for each duplicate statement found.

If you code the same user exit routine module name more than once in the exit routine list (EXIT=) of any single EXITDEF= statement, BPE always uses the **first** occurrence of the exit routine module name to determine the order for calling the exit routines. Duplicate names are ignored, and a message BPE0018I is issued for each duplicate name.

BPE EXITMBR= EXITDEF Parameter

```
                                    ┌─,────────┐
►►──EXITDEF=─(TYPE=type,EXITS=(───▼─exitname─┴─)──┬──────────────┬──┬──────────────────────┬─)──────►◄
                                                  └─,ABLIM=limit─┘  └─,COMP=ims_component─┘
```

The EXITDEF statement associates an exit routine type with a list of one or more exit routine modules to be called. The modules are called in the order listed. The EXITDEF statement consists of a sublist (enclosed in parentheses) containing the keywords TYPE, EXITS, ABLIM, and COMP.

TYPE=*type*
: Specifies the type of exit routine. The IMS component defines the types of exit routines that are supported.

EXITS=(*exitname,...***)**
: Specifies a list of one or more exit routine module names. The position of the exit routine in the list determines the order in which the exit routine is driven. When an exit routine returns to its caller, it indicates whether additional exit routines are to be called.

ABLIM=*limit*
: A number from 0 to 2147483647 that specifies the abend limit for the type of exit routine being defined. If the number of abends for an exit routine module

reaches the abend limit for the exit routine type, the module is removed from the exit routine list and is not called until the exit routine type is refreshed.

This parameter is optional; the default is 1. If you specify a value of 0, there is no abend limit.

Related Reading: For complete information about refreshing user exit routines, see "Refreshing User Exits in BPE" on page 41.

COMP =*ims_component*
An optional parameter that specifies the type of the IMS component that owns the exit routine being defined. Possible values are:

BPE
Base Primitive Environment

CQS
Common Queue Server

OM
Operations Manager

RM
Resource Manager

SCI
Structured Call Interface

| BPE processes only EXITDEF statements that meet the following criteria:

| • statements that do not have COMP coded

| • statements that have COMP=BPE coded

| • statements that COMP=ims_component coded (where *ims_component*
| matches the IMS component that is currently running).

The EXITDEF *types* provided by the various IMS components are described in:

- "BPE EXITDEF Types"
- "CQS EXITDEF Types" on page 20
- "OM EXITDEF Types" on page 20
- "RM EXITDEF Types" on page 20
- "SCI EXITDEF Types" on page 20

BPE EXITDEF Types

INITTERM
Called once during early BPE initialization, and once during normal termination. Refer to Chapter 4, "BPE User-Supplied Exit Routines," on page 47 for more information on this exit routine.

STATS
Called periodically (timer-driven), and once during normal address space shutdown, with statistics about BPE system functions. Optionally, the IMS component running on top of BPE can provide statistics specific to its operation. Refer to Chapter 4, "BPE User-Supplied Exit Routines," on page 47 for more information on this exit routine.

Important: All BPE-owned user exit routines are available to all IMS address spaces running with BPE.

CQS EXITDEF Types

CLNTCONN
Called during client connect and disconnect processing. Refer to the *IMS Version 9: Common Queue Server Guide and Reference* for more information on this exit routine.

INITTERM
Called during various phases of initialization and termination. Refer to the *IMS Version 9: Common Queue Server Guide and Reference* for more information on this exit routine.

OVERFLOW
Called during overflow threshold processing to select queue names for overflow processing. Refer to *IMS Version 9: Common Queue Server Guide and Reference* for more information on this exit routine.

STRSTAT
Called during checkpoint processing to allow you to gather structure statistics. Refer to the *IMS Version 9: Common Queue Server Guide and Reference* for more information on this exit routine.

STREVENT
Called for various structure events. For certain structure events, it also allows you to gather structure statistics like the STRSTAT exit routine. Refer to the *IMS Version 9: Common Queue Server Guide and Reference* for more information on this exit routine.

OM EXITDEF Types

CLNTCONN
Called during client command registration and deregistration processing.

INITTERM
Called during various phases of initialization and termination.

INPUT
| Called to view command input to the Operations Manager. This exit routine can
| either modify the command before execution or reject the command before it is
| processed.

OUTPUT
Called to view output (for example, command response) from Operations Manager to an automation client. The exit routine can modify the output before it is returned to the originator of the command.

SECURITY
Called to allow user security checking prior to command execution.

RM EXITDEF Types

CLNTCONN
Called during client connect and disconnect processing.

INITTERM
Called during various phases of initialization and termination.

SCI EXITDEF Types

CLNTCONN
Called during client connect and disconnect processing.

INITTERM
Called during various phases of initialization and termination.

Related Reading: See *IMS Version 9: Common Service Layer Guide and Reference* for more information about Operations Manager, Resource Manager, and Structured Call Interface exit routines.

Sample CQS User Exit List PROCLIB Member

A sample CQS user exit list PROCLIB member is shown in Figure 3. It defines the following exit routines:

- One client connection exit routine
- Two INITTERM user exit routines
- Four overflow exit routines
- One structure statistic exit routine
- One CQS structure event user exit routine

```
********************************************************************
* CQS USER EXIT LIST PROCLIB MEMBER                                *
********************************************************************

#------------------------------------------------------------------#
# DEFINE 1 CLIENT CONNECTION EXIT: CLCONX00                        #
#----------------------------------------------------------------- #
EXITDEF(TYPE=CLNTCONN,EXITS=(CLCONX00))

#------------------------------------------------------------------#
# DEFINE 2 INITTERM USER EXITS:  MYCQSIT0 AND OEMCQIT0             #
# WITH AN ABEND LIMIT OF 8.
#------------------------------------------------------------------#
EXITDEF(TYPE=INITTERM,EXITS=(MYCQSIT0,OEMCQIT0),ABLIM=8)

#------------------------------------------------------------------#
# DEFINE 4 OVERFLOW EXITS: OVERFL01, OVERFL02, OVERFL03, OVERFL04 #
#------------------------------------------------------------------#
EXITDEF(TYPE=OVERFLOW,EXITS=(OVERFL01,
                             OVERFL02,
                             OVERFL03,
                             OVERFL04))

#------------------------------------------------------------------#
# DEFINE 1 STRUCTURE STATISTIC EXIT: STRSTAT0                      #
#------------------------------------------------------------------#
EXITDEF(TYPE=STRSTAT,EXITS=(STRSTAT0))

#------------------------------------------------------------------#
# DEFINE 1 CQS STRUCTURE EVENT USER EXIT (STREVNT0) WITH           #
# NO ABEND LIMIT                                                   #
#------------------------------------------------------------------#
EXITDEF(TYPE=STREVENT,EXITS=(STREVNT0),ABLIM=0)
```

Figure 3. Example of a CQS User Exit Routine List PROCLIB Member

Sample OM User Exit List PROCLIB Member

A sample OM user exit list PROCLIB member is shown in Figure 4 on page 22. It defines the following exit routines:

- One OM init/term exit routine
- Two OM client connection exit routines
- One OM command input exit routine
- One OM command output exit routine
- Three OM security exit routines

```
********************************************************************
* OM USER EXIT LIST PROCLIB MEMBER                                 *
********************************************************************

#------------------------------------------------------------------#
# Define one OM init/term exit: OMINITRM.                          #
#------------------------------------------------------------------#
EXITDEF(TYPE=INITTERM,EXITS=(OMINITRM))

#------------------------------------------------------------------#
# Define 2 OM client connection exits: OMCLCN00 and OEMCLI00      #
# with an abend limit of 2.                                        #
#------------------------------------------------------------------#
EXITDEF(TYPE=CLNTCONN,EXITS=(OMCLCN00,OEMCLI00),ABLIM=2)

#------------------------------------------------------------------#
# Define one OM command input exit: MYCMI000                       #
# with no abend limit.                                             #
#------------------------------------------------------------------#
EXITDEF(TYPE=INPUT,EXITS=(MYCMI000),ABLIM=0)
#------------------------------------------------------------------#
# Define one OM command output exit: MYCMO000                      #
# with no abend limit.                                             #
#------------------------------------------------------------------#
EXITDEF(TYPE=OUTPUT,EXITS=(MYCMO000),ABLIM=0)

#------------------------------------------------------------------#
# Define 3 OM security exits: OMSEC000,OMSEC001, and ZZZSEC00     #
#------------------------------------------------------------------#
EXITDEF(TYPE=SECURITY,EXITS=(OMSEC000,
                             OMSEC001,
                             ZZZSEC00))
```

Figure 4. Example of an OM User Exit Routine List PROCLIB Member

Sample RM User Exit List PROCLIB Member

A sample RM user exit list PROCLIB member is shown in Figure 5. It defines the following exit routines:

- One RM init/term exit routine
- Two RM client connection exit routines

```
********************************************************************
* RM USER EXIT LIST PROCLIB MEMBER                                 *
********************************************************************

#------------------------------------------------------------------#
# Define one RM init/term exit: RMINITRM.                          #
#------------------------------------------------------------------#
EXITDEF(TYPE=INITTERM,EXITS=(RMINITRM))

#------------------------------------------------------------------#
# Define 2 RM client connection exits: RMCLCN00 and XYZCLCN0      #
# with an abend limit of 6.                                        #
#------------------------------------------------------------------#
EXITDEF(TYPE=CLNTCONN,EXITS=(RMCLCN00,XYZCLCN0),ABLIM=6)
```

Figure 5. Example of an RM User Exit Routine List PROCLIB Member

Sample SCI User Exit List PROCLIB Member

A sample SCI user exit list PROCLIB member is shown in Figure 6. It defines the following exit routines:

- One SCI init/term exit routine
- Three SCI client connection exit routines

```
**********************************************************************
* SCI USER EXIT LIST PROCLIB MEMBER                                  *
**********************************************************************

#-------------------------------------------------------------------#
# Define one SCI init/term exit: SCINITRM.                          #
#-------------------------------------------------------------------#
EXITDEF(TYPE=INITTERM,EXITS=(SCINITRM))

#-------------------------------------------------------------------#
# Define 3 SCI client connection exits: SCCLCN00, SCCLCN10,         #
# and SCCLCN20 with an abend limit of 9.                            #
#-------------------------------------------------------------------#
EXITDEF(TYPE=CLNTCONN,EXITS=(SCCLCN00,SCCLCN10,SCCLCN20),ABLIM=9)
```

Figure 6. Example of an SCI User Exit Routine List PROCLIB Member

Sample BPE User Exit List PROCLIB Member

A sample BPE user exit list PROCLIB member is shown in Figure 7. It defines the following exit routines:

- One BPE init/term exit routine
- One BPE Statistics exit routine

```
**********************************************************************
* BPE USER EXIT LIST PROCLIB MEMBER                                  *
**********************************************************************

#-------------------------------------------------------------------#
# Define one BPE init/term exit: MYINIT00.                          #
#-------------------------------------------------------------------#
EXITDEF(TYPE=INITTERM,EXITS=(MYINIT00))

#-------------------------------------------------------------------#
# Define 1 BPE Statistics exit: HHGSTAT0 with an abend limit of 42#
#-------------------------------------------------------------------#
EXITDEF(TYPE=STATS,EXITS=(HHGSTAT0),ABLIM=42)
```

Figure 7. Example of a BPE User Exit Routine List PROCLIB Member

Sample Combined User Exit List PROCLIB Member

You can combine all of the preceding user exit list PROCLIB members into a single shared member by using the COMP keyword on the EXITDEF statements, as shown in Figure 8 on page 24. It give examples of the following definitions:

- CQS user exit routines
- OM user exit routines
- RM user exit routines
- SCI user exit routines
- SCI BPE exit routines

```
*********************************************************************
* CQS USER EXIT DEFINITIONS                                         *
*********************************************************************
EXITDEF=(TYPE=CLNTCONN,EXITS=(CLCONX00),COMP=CQS)
EXITDEF=(TYPE=INITTERM,EXITS=(MYCQSIT0,OEMCQIT0),ABLIM=8,COMP=CQS)
EXITDEF=(TYPE=OVERFLOW,EXITS=(OVERFL01,
                              OVERFL02,
                              OVERFL03,
                              OVERFL04),COMP=CQS)
EXITDEF=(TYPE=STRSTAT,EXITS=(STRSTAT0),COMP=CQS)
EXITDEF=(TYPE=STREVENT,EXITS=(STREVENT),ABLIM=0,COMP=CQS)

*********************************************************************
* OM USER EXIT DEFINITIONS                                          *
*********************************************************************
EXITDEF=(TYPE=INITTERM,EXITS=(OMINITRM),COMP=OM)
EXITDEF=(TYPE=CLNTCONN,EXITS=(OMCLCN00,OEMCLI00),ABLIM=2,COMP=OM)
EXITDEF=(TYPE=INPUT,EXITS=(MYCMI000),ABLIM=0,COMP=OM)
EXITDEF=(TYPE=OUTPUT,EXITS=(MYCMO000),ABLIM=0,COMP=OM)
EXITDEF=(TYPE=SECURITY,EXITS=(OMSEC000,
                              OMSEC001,
                              ZZZSEC00),COMP=OM)

*********************************************************************
* RM USER EXIT DEFINITIONS                                          *
*********************************************************************
EXITDEF=(TYPE=INITTERM,EXITS=(RMINITRM),COMP=RM)
EXITDEF=(TYPE=CLNTCONN,EXITS=(RMCLCN00,XYZCLCN0),ABLIM=6,COMP=RM)

*********************************************************************
* SCI USER EXIT DEFINITIONS                                         *
*********************************************************************
EXITDEF=(TYPE=INITTERM,EXITS=(SCINITRM),COMP=SCI)
EXITDEF=(TYPE=CLNTCONN,EXITS=(SCCLCN00,SCCLCN10,SCCLCN20),ABLIM=9,
         COMP=SCI)

*********************************************************************
* SCI BPE EXIT DEFINITIONS                                          *
*********************************************************************
EXITDEF=(TYPE=INITTERM,EXITS=(MYINIT00),COMP=BPE)
EXITDEF=(TYPE=STATS,EXITS=(HHGSTAT0),ABLIM=42,COMP=BPE)
```

Figure 8. Example of a Combined Exit Routine List PROCLIB Member

Note: If you use a single shared user exit list PROCLIB member, change the EXITMBR statements in the BPE configuration PROCLIB member to point to the shared user exit list PROCLIB member. Here is an example of how you might change the EXITMBR statements:

```
#
# User exit list PROCLIB member specifications
#
EXITMBR=(SHREXIT0,BPE)              /* BPE user exit definitions  */
EXITMBR=(SHREXIT0,CQS)              /* CQS user exit definitions  */
EXITMBR=(SHREXIT0,OM)               /* OM user exit definitions   */
EXITMBR=(SHREXIT0,RM)               /* RM user exit definitions   */
EXITMBR=(SHREXIT0,SCI)              /* SCI user exit definitions  */
```

Chapter 3. BPE Commands

This chapter describes the Base Primitive Environment (BPE) commands.

BPE provides a set of commands that you can issue to any IMS component that is running in a BPE environment (CQS, RM, OM, SCI).

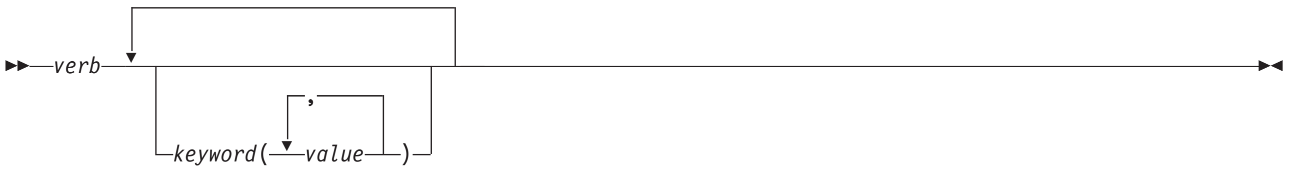
In this chapter:

BPE Command Syntax and Invocation

BPE supports two command formats: a verb-only format, and a verb-resource type format.

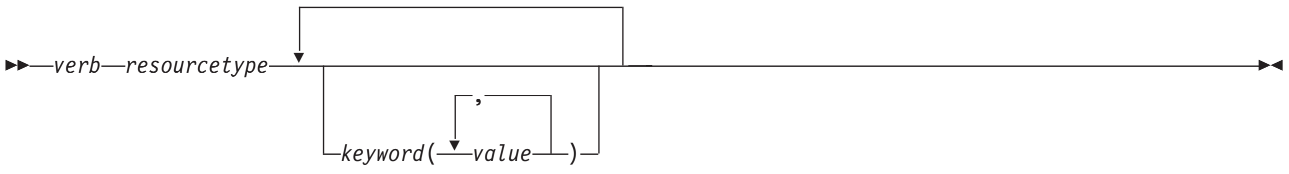
The verb-only format consists of a verb, followed by zero or more keyword-value pairs, with the values enclosed in parentheses.

BPE Verb Only Command Syntax



The verb-resource type format consists of a verb, a resource type, and zero or more keyword value pairs.

BPE Verb-Resource Type Command Syntax



verb
: A command verb representing an action. Some verb examples are DISPLAY, UPDATE, and REFRESH.

resourcetype
: The type of resource that is operated on by the verb. Some resource examples are TRACETABLE and USEREXIT.

keyword(value)
: A set of zero or more keywords and values that represent attributes, filters, or

other modifiers that apply to the command. For example, NAME() to identify the specific resources or LEVEL() to specify a trace level.

BPE Command Invocation

You can only invoke BPE commands through the z/OS MODIFY command. The following diagram illustrates the general syntax for entering commands through the modify interface.

BPE Command Invocation

```
►►──F──jobname,command──────────────────────────────────────────────────►◄
```

F The z/OS modify command.

jobname
The jobname of the address space to which the command is directed.

command
The command being issued.

BPE Wildcard Character Support

Some parameters on BPE commands support wildcard characters for pattern matching. For such parameters, you can use the following wildcard characters:

***** Matches zero or more characters

% Matches exactly one character

The following examples illustrate some uses of wildcard characters.

BE* Matches any string beginning with ″BE″, of any length. For instance: BE, BEE, BEEBLEBROX.

%%S Matches any three-character string ending with an ″S″. For instance: IMS, CQS.

R*S*T%R
Matches any string beginning and ending with ″R″, having an ″S″, followed by a ″T″ in the middle, with any number of intervening characters between the first ″R″, the ″S″, and the ″T″, and exactly one character between the ″T″ and the final ″R″. For example, ROASTER, ROSTER, RESORTER, RESCEPTOR, RSTZR.

***** Matches any string.

Specifying IMS Component Command Parameters

BPE commands enable you to display and update resources that BPE manages. Some resource types are defined and owned by BPE itself. These resource types are known as ″system resource types.″ Commands that specify system resource types can be issued to any IMS component running in a BPE environment. For example, BPE defines several BPE system trace table types like DISP, STG, and CBS. These trace tables exist in every BPE address space. Commands to display and update these trace table types can be issued to any BPE address space.

Other resource types are defined and owned by the IMS component that is using BPE services. These resource types are known as ″component resource types″ or ″user-product resource types.″ Commands that specify component resource types can only be issued to the IMS component that defines those types. For example, CQS defines several CQS-specific trace tables such as STR, CQS, and INTF. Commands to display and update these trace table types can be issued only to CQS address spaces.

BPE commands also provide the ability to restrict the resource types upon which a command operates to either those owned by BPE, or to those owned by the IMS component of the address space to which the command is issued. This is done through the OWNER keyword on commands that support OWNER. Use `OWNER(BPE)` to restrict the command operation to resource types that BPE owns and defines (system resource types). Use `OWNER`*`(component_type)`* to restrict the command operation to resource types that the IMS component address defines and owns (component resource types). Table 2 lists the valid values for the OWNER parameter, and the address space types to which they apply:

Table 2. Valid Values for OWNER Parameter

OWNER	Address Space Type
BPE	Any IMS component running in a BPE address space
CQS	Common Queue Server
OM	Operations Manager
RM	Resource Manager
SCI	Structured Call Interface

BPE TRACETABLE Commands

The TRACETABLE resource type refers to the internal BPE-managed trace tables defined either by BPE (for example: DISP, CBS, STG, LATC), or by the IMS component using BPE (for example: CQS, OM, RM, SCI). Two command verbs operate on the TRACETABLE resource type:

DISPLAY Display trace level and number of trace table pages of specified trace tables.

UPDATE Update trace level attribute of specified trace tables.

Format of BPE DISPLAY TRACETABLE Command

Use this command to display the current attribute settings for the requested trace tables.

```
                                             ┌─,──────────────────────┐
►►──┬─DISPLAY─┬──┬─TRACETABLE─┬──NAME(─▼─┬─trace_table_name──┬─┴─)──┬──────────────────┬──────────►◄
    └─DIS─────┘  └─TRTAB──────┘          └─trace_table_name*─┘      └─OWNER(─┬─BPE─┬─)─┘
                                                                             ├─CQS─┤
                                                                             ├─OM──┤
                                                                             ├─RM──┤
                                                                             └─SCI─┘
```

Usage of BPE DISPLAY TRACETABLE Command

DISPLAY | DIS
A required parameter, which displays the attributes of the specified resource.

TRACETABLE | TRTAB
A required parameter, which specifies that the resource type being acted upon is a BPE-managed trace table.

NAME(*trace_table_name*)
A required parameter, which specifies the name of the trace table types about which you want attributes displayed. You can specify a single trace table name or a list of trace table names separated by commas. Trace table names can contain wildcard characters. See "BPE Wildcard Character Support" on page 26 for more information about using wildcard characters. Trace table names can be BPE-defined trace tables or IMS component-defined trace tables.

You can display BPE-defined trace tables for any IMS component address space that is using BPE. These BPE-defined trace table types are available:

AWE	Asynchronous work element (AWE) trace table
CBS	Control block services trace table
CMD	Command trace table
DISP	Dispatcher trace table
HASH	Hash trace table
ERR	BPE Error trace table
LATC	Latch trace table
MISC	Miscellaneous trace table that is used only by IMS Service for trap traces
SSRV	System services trace table
STG	Storage service trace table
USRX	User exit routine trace table

You can display CQS-defined trace tables only for CQS address spaces. These CQS-defined trace table types are available:

CQS	CQS trace table
ERR	CQS error trace table
INTF	CQS interface trace table
STR	CQS structure trace table

You can display OM-defined trace tables only for OM address spaces. These OM-defined trace table types are available:

CSL	Common Service Layer (CSL) trace table
ERR	OM error trace table
OM	Operations Manager (OM) processes trace table
PLEX	IMSplex trace table for OM processing for a specific IMSplex

You can display RM-defined trace tables only for RM address spaces. These RM-defined trace table types are available:

CSL	Common Service Layer (CSL) trace table
ERR	RM error trace table
PLEX	IMSplex trace table for RM processing for a specific IMSplex
RM	Resource Manager (RM) processes trace table

You can display SCI-defined trace tables only for SCI address spaces. These SCI-defined trace table types are available:

CSL	Common Service Layer (CSL) trace table
ERPL	SCI Error Parameter List trace table
ERR	SCI error trace table
INTF	SCI interface trace table
INTP	SCI interface parameter trace table
PLEX	IMSplex trace table for SCI processing for a specific IMSplex
SCI	Structured Call Interface (SCI) processes trace table

OWNER(BPE | CQS | OM | RM | SCI)

An optional parameter that specifies the owner of the trace table type or types about which you want attributes displayed. You can specify one of the following values:

BPE	For all IMS components that are running in a BPE address space
CQS	For CQS address spaces only
OM	For OM address spaces only
RM	For RM address spaces only
SCI	For SCI address spaces only

The OWNER parameter acts as a filter to help you select which trace tables you want to display. For example, you could specify `NAME(*) OWNER(CQS)` to display all of the CQS-defined trace table types (CQS, ERR, STR, and INTF) in a CQS address space. You could specify `NAME(*) OWNER(BPE)` to display all of the BPE-defined trace table types in *any* BPE-managed address space. If `OWNER` is omitted, then both BPE and component trace tables might be displayed (depending on the tables specified on `NAME`).

BPE DISPLAY TRACETABLE Command Output

The DISPLAY TRACETABLE command output consists of a header line, one line per selected trace table, and one message BPE0032I line indicating that the command has completed. Here is an example.

```
BPE0030I TABLE  OWNER  LEVEL  #PAGES
BPE0000I DISP   BPE    HIGH       12
BPE0000I STR    CQS    MEDIUM      8
BPE0032I DISPLAY TRACETABLE COMMAND COMPLETED
```

These columns are in the DISPLAY TRACETABLE output:

TABLE Specifies the name of the trace table type about which information is being displayed on the current row. Either BPE or the product using BPE owns this trace table.

OWNER
Specifies the IMS component that owns the trace table. BPE-owned trace tables are system trace tables, and exist in all IMS component address spaces that use BPE. Trace tables that are specific to an IMS component show the 1- to 4-character component identifier in this column.

LEVEL
Specifies the current level setting of the trace table. A trace table's level determines the volume of trace data collected. These levels are possible:

NONE	No trace data is being written to the table.
ERROR	Only traces for error or exception conditions are being written into the table.
LOW	Only major event trace entries are written into the table.
MEDIUM	Major event trace entries and some minor event trace entries are written into the table.
HIGH	All trace entries are written into the table.
INACTV	The trace table is inactive and cannot be used. This status occurs only when BPE was unable to get any storage for the trace table. No tracing will be done for the indicated table type, and you cannot change the level for the trace table with the UPDATE TRACETABLE command. You must restart the address space in order to use the trace table again.

#PAGES
Specifies the number of 4K (4096 byte) pages allocated for the trace table type.

Command Example 1

Display the status of the BPE dispatcher trace table (DISP).

Command:

```
F CQS1,DISPLAY TRACETABLE NAME(DISP)
```

Output:

```
BPE0030I TABLE  OWNER  LEVEL  #PAGES
BPE0000I DISP   BPE    HIGH       12
BPE0032I DISPLAY TRACETABLE COMMAND COMPLETED
```

Command Example 2

Display the status of all CQS traces.

Command:

```
F CQS1,DIS TRTAB NAME(*) OWNER(CQS)
```

Output:

```
BPE0030I TABLE  OWNER  LEVEL  #PAGES
BPE0000I CQS    CQS    MEDIUM      4
BPE0000I ERR    CQS    HIGH        4
BPE0000I INTF   CQS    LOW         8
BPE0000I STR    CQS    HIGH        8
BPE0032I DISPLAY TRACETABLE COMMAND COMPLETED
```

Command Example 3

Display the status of all traces in an SCI address space.

Command:

```
F SCI,DIS TRTAB NAME(*)
```

Output:

```
BPE0030I TABLE  OWNER  LEVEL  #PAGES
BPE0000I AWE    BPE    HIGH      6
BPE0000I CBS    BPE    HIGH      6
BPE0000I CMD    BPE    HIGH      2
BPE0000I CSL    SCI    HIGH      8
BPE0000I DISP   BPE    HIGH      8
BPE0000I ERPL   SCI    HIGH      8
BPE0000I ERR    BPE    HIGH      2
BPE0000I ERR    SCI    HIGH      4
BPE0000I HASH   BPE    HIGH      8
BPE0000I INTF   SCI    HIGH      8
BPE0000I INTP   SCI    HIGH      16
BPE0000I LATC   BPE    HIGH      8
BPE0000I MISC   BPE    HIGH      1
BPE0000I PLEX   SCI    HIGH      8
BPE0000I SCI    SCI    HIGH      8
BPE0000I SSRV   BPE    HIGH      4
BPE0000I STG    BPE    HIGH      8
BPE0000I USRX   BPE    HIGH      4
BPE0032I DISPLAY TRACETABLE COMMAND COMPLETED
```

Command Example 4

Display the status of all OM traces.

Command:

```
F OM,DIS TRTAB NAME(*) OWNER(OM)
```

Output:

```
BPE0030I TABLE  OWNER  LEVEL  #PAGES
BPE0000I CSL    OM     HIGH      4
BPE0000I ERR    OM     HIGH      4
BPE0000I OM     OM     HIGH      4
BPE0000I PLEX   OM     HIGH      8
BPE0032I DISPLAY TRACETABLE COMMAND COMPLETED
```

Command Example 5

Display the status of the PLEX trace and all traces beginning with ″C″ in the RM address space.

Command:

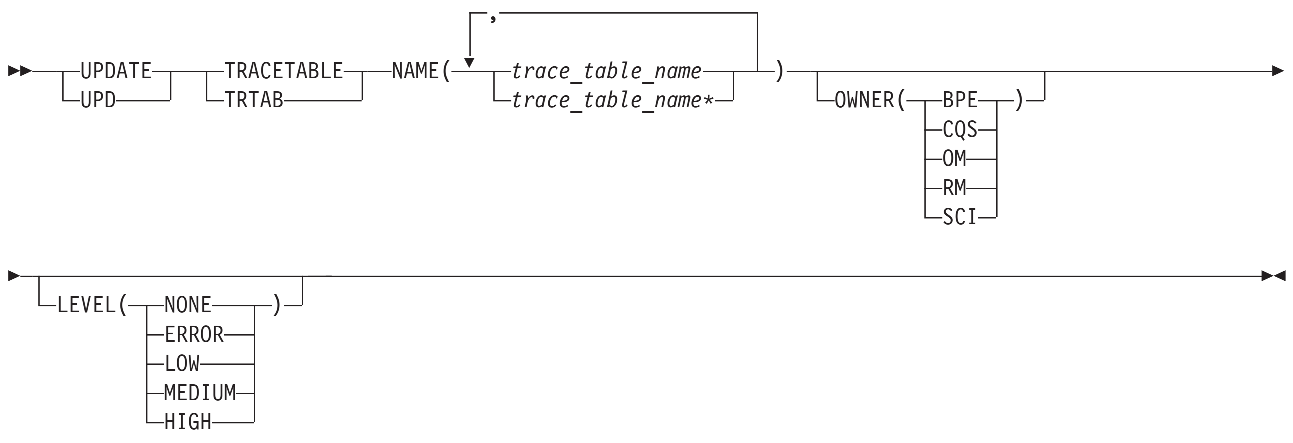
```
F RM,DIS TRTAB NAME(PLEX,C*)
```

Output:

```
BPE0030I TABLE  OWNER  LEVEL  #PAGES
BPE0000I CBS    BPE    HIGH      6
BPE0000I CMD    BPE    HIGH      2
BPE0000I CSL    RM     HIGH      4
BPE0000I PLEX   RM     HIGH      8
BPE0032I DISPLAY TRACETABLE COMMAND COMPLETED
```

Format of BPE UPDATE TRACETABLE Command

Use this command to change the trace level setting for the requested trace tables.



Usage of BPE UPDATE TRACETABLE Command

UPDATE | UPD
A required parameter, which specifies that the action against the trace table is to update its attributes.

TRACETABLE | TRTAB
A required parameter, which specifies that the resource type being acted upon is a BPE-managed trace table.

NAME(*trace_table_name***)**
A required parameter, which specifies the name of the trace table type or types that you want to update. You can specify a single trace table name or a list of trace table names separated by commas. Trace table names can contain wildcard characters. See "BPE Wildcard Character Support" on page 26 for more information about using wildcard characters. Trace table names can be BPE-defined trace tables or IMS component-defined trace tables.

You can update BPE-defined trace tables for any IMS component address space that is using BPE. These BPE-defined trace table types are available:

AWE	Asynchronous work element (AWE) trace table
CBS	Control block services trace table
CMD	Command trace table
DISP	Dispatcher trace table
ERR	BPE Error trace table
HASH	Hash trace table
LATC	Latch trace table
MISC	Miscellaneous trace table that is used only by IMS Service for trap traces
SSRV	System services trace table

STG	Storage service trace table
USRX	User exit routine trace table

You can update CQS-defined trace tables only for CQS address spaces. These CQS-defined trace table types are available:

CQS	CQS trace table
ERR	CQS error trace table
INTF	CQS interface trace table
STR	CQS structure trace table

You can update OM-defined trace tables only for OM address spaces. These OM-defined trace table types are available:

CSL	Common Service Layer (CSL) trace table
ERR	OM error trace table
OM	Operations Manager (OM) processes trace table
PLEX	IMSplex trace table for OM processing for a specific IMSplex

You can update RM-defined trace tables only for RM address spaces. These RM-defined trace table types are available:

CSL	Common Service Layer (CSL) trace table
ERR	RM error trace table
PLEX	IMSplex trace table for RM processing for a specific IMSplex
RM	Resource Manager (RM) processes trace table

You can update SCI-defined trace tables only for SCI address spaces. These SCI-defined trace table types are available:

CSL	Common Service Layer (CSL) trace table
ERPL	SCI Error Parameter List trace table
ERR	SCI error trace table
INTF	SCI interface trace table
INTP	SCI interface parameter trace table
PLEX	IMSplex trace table for SCI processing for a specific IMSplex
SCI	Structured Call Interface (SCI) processes trace table

OWNER(BPE | CQS | OM | RM | SCI)
An optional parameter that specifies the owner of the trace table type or types that you want to update. You can specify one of the following values:

BPE	For all IMS components that are running in a BPE address space
CQS	For CQS address spaces only
OM	For OM address spaces only
RM	For RM address spaces only
SCI	For SCI address spaces only

The OWNER parameter acts as a filter to help you select which trace tables you want to update. For example, you could specify `NAME(*) OWNER(CQS)` to update all of the CQS-defined trace table types (CQS, ERR, STR, and

INTF) in a CQS address space. You could specify `NAME(*) OWNER(BPE)` to update all of the BPE-defined trace table types in *any* BPE-managed address space. If `OWNER` is omitted, then both BPE and component trace tables might be updated (depending on the tables specified on `NAME`).

LEVEL(*level*)
An optional parameter that sets the new tracing level for the specified trace tables. If LEVEL is omitted, the level of the specified trace tables is not changed. These levels are possible:

NONE	No trace data is being written to the table.
ERROR	Only traces for error or exception conditions are being written into the table.
LOW	Only major event trace entries are written into the table.
MEDIUM	Major event trace entries and some minor event trace entries are written into the table.
HIGH	All trace entries are written into the table.

Important: You cannot change the level for the trace table type ERR. BPE forces the level to HIGH to ensure that error diagnostics are captured. Any level that you specify for the ERR trace table is ignored.

BPE UPDATE TRACETABLE Command Output

The `UPDATE TRACETABLE` command output consists of message BPE0032I indicating that the command has completed:

```
BPE0032I UPDATE TRACETABLE COMMAND COMPLETED
```

Command Example 1

Update the level of the BPE dispatcher trace table (DISP) to HIGH.

Command:

```
F CQS1,UPDATE TRACETABLE NAME(DISP) LEVEL(HIGH)
```

Output:

```
BPE0032I UPDATE TRACETABLE COMMAND COMPLETED
```

Command Example 2

Update the level of all SCI trace tables to MEDIUM.

Important: You cannot change the level for the trace table type ERR, even when using a wildcard character to select all tables with a given owner, as shown in this example--`NAME(*)`. BPE forces the level to HIGH to ensure that error diagnostics are captured.

Command:

```
 F SCI,UPD TRTAB NAME(*) OWNER(SCI) LEVEL(MEDIUM)
```

Output:

```
BPE0032I UPDATE TRACETABLE COMMAND COMPLETED
```

BPE DISPLAY VERSION Command

Use this command to display both the version of the IMS component that is using BPE, and the version of the BPE in use.

Format of BPE DISPLAY VERSION Command

```
►►──┬─DISPLAY─┬──┬─VERSION─┬──────────────────────────────────────────────►◄
    └─DIS─────┘  └─VER─────┘
```

Usage of BPE DISPLAY VERSION Command

DISPLAY | DIS
A required parameter, which specifies that the action against the specified resource is to display attributes of the resource.

VERSION | VER
A required parameter, which specifies that the resource types being acted upon are the version number of the IMS component **and** the BPE in the current address space.

DISPLAY VERSION Command Output

The DISPLAY VERSION command output consists of a single display output line in the format **`BPE00001`** *`comp`* **`VERSION`**=*`cv.cr.cp`* **`BPE VERSION`**=*`bv.br.bp`*.

- *`comp`* is the IMS component ID for the address space. It is one to four characters long and can have one of the following values:
  - CQS (Common Queue Server)
  - OM (Operations Manager)
  - RM (Resource Manager)
  - SCI (Structured Call Interface)
- *`cv.cr.cp`* is the full version number of the IMS component, where *cv* is the version, *cr* is the release, and *cp* is the point release. Similarly, *`bv.br.bp`* indicates the full version number of the BPE running in the address space.

Command Example 1

Display the version of a CQS address space.

Command:

```
   F CQS1,DISPLAY VERSION
```

Output:

```
BPE00001 CQS VERSION = 1.3.0  BPE VERSION = 1.4.0
```

Command Example 2

Display the version of an RM address space.

Command:

```
   F RM1,DISPLAY VERSION
```

Output:

```
BPE00001 RM VERSION = 1.1.0  BPE VERSION = 1.4.0
```

BPE USEREXIT Commands

Note: Throughout this section, the term “user exit routine” means “user-supplied exit routine.”

The USEREXIT resource type refers to the user exit types defined to and managed by either BPE or the IMS component using BPE (for example, CQS).

DISPLAY	Display attributes of specified user exit types.
REFRESH	Load new copies of the user exit modules for specified user exit types.

Format of BPE DISPLAY USEREXIT Command

Use this command to display attributes for all modules associated with the specified user exit types.

```
►►──────────────────────────────────────────────────────────────────────►◄
```

Usage of BPE DISPLAY USEREXIT Command

DISPLAY | DIS
A required parameter, which specifies that the action against the specified resource is to display attributes of the resource.

USEREXIT | USRX
A required parameter, which specifies that the resource type being acted upon is a BPE-managed user exit type.

NAME(*user_exit_type_name*)
A required parameter, which specifies the name of the user exit type or types about which you want attributes displayed. You can specify a single user exit type name or a list of user exit type names separated by commas. User exit type names can contain wildcard characters.

Related Reading:

For more information about using wildcards, see “BPE Wildcard Character Support” on page 26.

Important: The name or names specified in this parameter are the names of user exit types, *not* the names of individual user exit modules.

BPE and each address space that can use BPE have different user exit types. As specified by OWNER(BPE), BPE’s user exit types include:

INITTERM	Initialization-Termination user exit
STATS	BPE system functions statistics user exit

As specified by OWNER(CQS), the following user exit types are defined in all CQS address spaces:

CLNTCONN	Client Connection user exit
INITTERM	Initialization-Termination user exit
OVERFLOW	Queue Overflow user exit
STRSTAT	Structure statistics user exit
STREVENT	Structure event user exit

As specified by OWNER(OM), the following user exit types are defined in all OM address spaces:

CLNTCONN	Client Connection command registration and deregistration user exit

INITTERM	Initialization-Termination user exit
INPUT	Command input user exit
OUTPUT	Output user exit
SECURITY	Security checking user exit

As specified by OWNER(RM), the following user exit types are defined in all RM address spaces:

CLNTCONN	Client Connection and Disconnection user exit
INITTERM	Initialization-Termination user exit

As specified by OWNER(SCI), the following user exit types are defined in all SCI address spaces:

CLNTCONN	Client Connection and Disconnection user exit
INITTERM	Initialization-Termination user exit

Related Reading:

- See the *IMS Version 9: Common Queue Server Guide and Reference* for more information about the CQS user exit routine types.
- See the *IMS Version 9: Common Service Layer Guide and Reference* for more information about the OM, RM, and SCI user exit routine types.

OWNER(BPE | CQS | OM | RM | SCI)
An optional parameter that specifies the owner of the user exit type or types about which you want attributes displayed. You can specify one of the following values:

BPE
For all IMS components that are running in a BPE address space

CQS
For CQS address spaces only

OM
For OM address spaces only

RM
For RM address spaces only

SCI
For SCI address spaces only

The OWNER parameter acts as a filter to help you select the user exit types that you want to display. For example, you could specify `NAME(*) OWNER(CQS)` to display all of the CQS-defined user exit types in a CQS address space. If `OWNER` is omitted, then both BPE and component user exits can be displayed (depending on the exits specified on `NAME`).

SHOW(*attribute*)
An optional parameter that specifies the attributes you want to display about the requested user exits.

When you display information about user exits, each row of display output contains the requested attributes for one user exit module, in columns. Every display for user exits contains the columns labeled EXITTYPE (the type of the exit), and MODULE (the load module name of the exit). Additionally, any of the following attributes can be requested by using the SHOW parameter:

ABENDS The number of abends that have occurred in the user exit module since the last user exit refresh of that module (or since address space initialization if no refreshes have occurred). BPE keeps track of the number of abends that have occurred in each user exit module. When this number reaches the number defined on the ABLIM= parameter of the EXITDEF statement for the exit's type, BPE stops calling the module. If the user exit module is refreshed, this count is reset to zero, and BPE calls the module again.

The maximum value that can be displayed in this field is 2147483647 (2^{31}-1). If the abend count exceeds this value, 2147483647 is displayed.

ABLIM The abend limit count for the user exit type, as specified on the ABLIM= parameter on the EXITDEF statement for the user exit type in the BPE exit list PROCLIB member. This is the number of times the user exit module is allowed to abend before BPE stops calling the user exit. A value of 0 indicates that there is no abend limit.

The maximum value that can be displayed in this field is 2147483647 (2^{31}-1).

ACTIVE The number of currently active instances of the user exit. This is a point-in-time number that represents the number of calls to the user exit that have not yet returned.

The maximum value that can be displayed in this field is 999999. If the active count exceeds this value, 999999 is displayed.

CALLS The number of calls to the user exit since the last user exit refresh.

For performance reasons, serialization is not obtained when BPE collects this number. For an exit type that can run multiple instances in parallel, this number should be considered an approximation only.

The maximum value that can be displayed in this field is 2147483647 (2^{31}-1). If the call count exceeds this value, 2147483647 is displayed.

ENTRYPT The entry point address of the user exit module.

ETIME The total (cumulative) elapsed time spent in the exit module since it was last refreshed, in milliseconds.

For performance reasons, serialization is not obtained when BPE collects this number. For an exit type that can run multiple instances in parallel, this number should be considered an approximation only.

The maximum value that can be displayed in this field is 2147483647 (2^{31}-1). If the elapsed number of milliseconds exceeds this value, 2147483647 is displayed.

LOADPT The load point address of the user exit module.

OWNER The IMS component that owns the user exit type. BPE-owned user exit types are system exit types that exist in all IMS component address spaces that use BPE. User exit types that are specific to the component show the 1- to 4-character component identifier in this column (for example, CQS).

RTIME This is the local date and time that the user exit module was last refreshed (or initially loaded, if no refreshes have occurred). The format of this output field is:

```
yyyy-mm-dd hh:mm:ss.th
```

SIZE The size of the user exit load module, in bytes (displayed in hexadecimal).

TEXT 27 bytes starting from offset +04 from the module's entry point, translated to EBCDIC, with non-printable characters replaced by periods (.). This is a common location for module identification information. If your user exits contain printable identification data at this point in the module, the TEXT option enables that information to be displayed.

If the SHOW parameter is not specified, the default attributes displayed after the EXITTYPE and MODULE are OWNER, ACTIVE, and ABENDS.

The order in which you list the attributes on the SHOW parameter has no effect on the order the attributes are displayed. BPE determines the order of the attribute columns in the display output. This order is as follows:

1. OWNER
2. ACTIVE
3. ABENDS
4. ABLIM
5. CALLS
6. ETIME
7. RTIME
8. ENTRYPT
9. LOADPT
10. SIZE
11. TEXT

Important: It is possible to request so many attributes that the length of the output line is too long to display with a WTO. If this happens, the command is processed, but some lines might be truncated. The maximum line length that BPE displays is 126 characters.

BPE DISPLAY USEREXIT Command Output

The `DISPLAY USEREXIT` command output consists of a header line, one line per user exit module about which information is being displayed, and one message, BPE0032I line indicating the command has completed.

Command:

```
F CQS1,DISPLAY USEREXIT NAME(INITTERM,STRSTAT)
```

Output:

```
BPE0030I EXITTYPE MODULE   OWNER    ACTIVE     ABENDS
BPE0000I INITTERM MYINIT00 CQS           0          0
BPE0000I INITTERM ZZZINIT0 CQS           0          0
BPE0000I STRSTAT  MYSTAT00 CQS           1          2
BPE0032I DISPLAY USEREXIT COMMAND COMPLETED
```

The EXITTYPE and MODULE columns are present for all DISPLAY USEREXIT commands, regardless of what is specified on SHOW. When multiple exit modules are listed for a single user exit type, they are listed in the order in which they are called.

Command Example 1

Display the status of the CQS structure event user exit type.

Command:

```
F CQS1,DISPLAY USEREXIT NAME(STREVENT)
```

Output:

```
BPE0030I EXITTYPE MODULE   OWNER    ACTIVE     ABENDS
BPE0000I STREVENT STREVX00 CQS           1          0
BPE0000I STREVENT ZZZSTEV0 CQS           0          0
BPE0032I DISPLAY USEREXIT COMMAND COMPLETED
```

In this example, there are two structure event exit modules defined that are called for CQS structure events. STREVX00 is called first, followed by ZZZSTEV0.

Command Example 2

Display the number of calls to, the elapsed time spent in, and the abend limit for all CQS user exit types.

Command:

```
F CQS1,DIS USRX NAME(*) OWNER(CQS) SHOW(CALLS,ETIME,ABLIM)
```

Output:

```
BPE0030I EXITTYPE MODULE        ABLIM      CALLS      ETIME
BPE0000I CLNTCONN CLCONX00          0          2         12
BPE0000I INITTERM MYCQSIT0          0          1          2
BPE0000I INITTERM OEMCQIT0          0          1        162
BPE0000I OVERFLOW OVERFL01          5          3          6
BPE0000I OVERFLOW OVERFL02          5          3         19
BPE0000I OVERFLOW OVERFL03          5          3          9
BPE0000I OVERFLOW OVERFL04          5          3      15593
BPE0000I STREVENT STREVNT0         10        542        628
BPE0000I STRSTAT  STRSTAT0          1         36       1889
BPE0000I STRSTAT  STRSTA10          1         36        241
BPE0032I DIS USRX COMMAND COMPLETED
```

Command Example 3

Display the entry point, load point, and size of all of the SCI CLNTCONN user exit modules.

Command:

```
F SCI,DIS USRX NAME(CLNTCONN) SHOW(SIZE,ENTRYPT,LOADPT)
```

Output:

```
BPE0030I EXITTYPE MODULE    ENTRYPT   LOADPT     SIZE
BPE0000I CLNTCONN SCCLCN00 8B864D78 8B864D78 00000458
BPE0000I CLNTCONN SCCLCN10 8BA14200 8BA14200 00001C10
BPE0000I CLNTCONN SCCLCN20 8BA18EE8 8BA18AF0 00000AB0
BPE0032I DIS USRX COMMAND COMPLETED
```

Command Example 4

Display the first part of the module text for all of the BPE user exits in the OM address space.

Command:

```
F OM,DIS USRX NAME(*) OWNER(BPE) SHOW(TEXT)
```

Output:

```
BPE0030I EXITTYPE  MODULE   TEXT
BPE0000I INITTERM  MYINIT00 .MYINIT00+20010615+17:47...
BPE0000I STATS     HHGSTAT0 .HHGSTAT0+20010615+08:47...
BPE0032I DIS USRX COMMAND COMPLETED
```

Command Example 5

Display the refresh time for all of the RM INITTERM modules.

Command:

```
F RM,DIS USRX NAME(*) OWNER(RM) SHOW(RTIME)
```

Output:

```
BPE0030I EXITTYPE MODULE   RTIME
BPE0000I INITTERM RMINITRM 2001-06-15 16:48:22.39
BPE0032I DIS USRX COMMAND COMPLETED
```

Refreshing User Exits in BPE

The `REFRESH USEREXIT` command does two things. It causes BPE to reprocess the user exit PROCLIB members specified in the BPE configuration PROCLIB member. It also reloads the user exit modules currently listed in the user exit PROCLIB members for the types specified on the command. This command enables you to make updates to your user exits without stopping and restarting the address space.

When you enter the `REFRESH USEREXIT` command, BPE performs the following processing:

- Reads any user exit PROCLIB members that are specified on EXITMBR= statements in the BPE configuration PROCLIB member. Because BPE re-reads these members at the time you issue the command, you can edit the user exit PROCLIB members prior to issuing the `REFRESH` command and make changes to the user exit definitions. BPE does *not* re-read the main BPE configuration PROCLIB member, so you cannot change the names of the user exit PROCLIB members, only their contents.
- Loads the user exit modules specified on the EXITDEF= statements for the user exit types specified on the command.
- Quiesces all current user exits. This means that the command waits for any active exits to complete processing and delays any new calls to the current exits. This ensures that no user exit is running while the exit is being refreshed.
- | Replaces BPE control block pointers to the previous user exit modules with pointers to the newly loaded modules. These pointers are used to manage the calling of the exits.
- Resumes the user exits and allows calls to be made to the newly-loaded exits.
- Deletes the old copy of the user exits.

BPE loads the new copies of the user exit modules before deleting the old modules. If an error occurs during this process (for example, a module could not be loaded or BPE internal control block storage could not be obtained), BPE fails the command and leaves the old copies of the user exits in effect. All modules of the specified user exit types must be loaded successfully for the command to complete successfully.

When a user exit module is refreshed, its abend count is reset to zero. This means that a user exit module that had reached its abend limit (specified by the ABLIM parameter on the EXITDEF statement) and was no longer being called by BPE is again called.

Important: If you changed the ABLIM parameter for a user exit in the PROCLIB member, the *new* value of ABLIM takes effect after the refresh command.

Considerations for Refreshing User Exits

- When you refresh a user exit type, BPE reloads all exit modules defined for that type. The new copies of the modules will be at a different virtual address than the old copies. Modules that are re-entrant will operate properly. However, if your modules are not re-entrant and they store data within themselves, they must be able to tolerate being reloaded and losing the information previously stored within them.

  Attention: Code and link edit all user exit modules as re-entrant to avoid this condition.
- If you refresh a previously loaded user exit module, BPE continues to pass the same static work area that was used by the previous copy of the module. If the new version of the module has a different mapping or use of this area than the previous version, the new version must contain toleration code that can handle the old-style formatted data within this static work area.

  Recommendation: Place a version number in the static work area, so that your exits can recognize when they are using a different data structure within this work area.
- If you remove a user exit module from an EXITDEF list and refresh the exits, BPE deletes the static work area associated with the removed exit module. If you later add the module back to the EXITDEF list and refresh the exits, the module gets a new (cleared) static work area, *not* the work area it had previously.
- If your user exits are being managed by link-lookaside (LLA) using virtual lookaside facility (VLF) or an equivalent product, you must ensure that the copies of the modules being refreshed are updated in LLA prior to issuing the `REFRESH USEREXIT` command. See the *MVS Initialization and Tuning Guide* for information on LLA-managed libraries.
- If you have user exits that issue z/OS WAITs for long periods of time (for example, a WAIT for an external event that may be delayed, such as a write to operator with reply (WTOR)), then issuing a `REFRESH USEREXIT` command could cause a performance problem or work stoppage. This is because BPE has to quiesce the user exits in order to process the `REFRESH` command. BPE must wait until all currently-called user exits complete before it can perform the user exit refresh. BPE prevents any new calls to user exits until after the command completes. If a user exit has been called and does not return to BPE for a long period of time, the `REFRESH` command is delayed until the exit returns. No other user exits can be called while BPE is waiting, so the processes that are invoking the user exits are also put into a wait state.

  Recommendation: Ensure that your user exits avoid long WAITs, and avoid issuing services that might WAIT.

Format of BPE REFRESH USEREXIT Command

```
                                    ┌─,────────────────────┐
►►──┬─REFRESH─┬──┬─USEREXIT─┬──NAME(─▼─┬─user_exit_type_name──┬─┴─)──┬────────────────────────┬──►◄
    └─REF─────┘  └─USRX─────┘          └─user_exit_type_name*─┘      └─OWNER(─┬─BPE─┬─)─┘
                                                                              ├─CQS─┤
                                                                              ├─OM──┤
                                                                              ├─RM──┤
                                                                              └─SCI─┘
```

Usage of BPE REFRESH USEREXIT Command

REFRESH | REF
A required parameter, which specifies that the action against the specified resources is to refresh the resources.

USEREXIT | USRX
A required parameter, which specifies that the resource type being acted upon is a BPE-managed user exit type.

NAME(*user_exit_type_name*)
A required parameter, which specifies the name of the user exit type or types that you want to refresh. You can specify a single user exit type name or a list of user exit type names separated by commas. User exit type names can contain wildcard characters.

Related Reading: For more information about using wildcard characters, see “BPE Wildcard Character Support” on page 26.

Important: The names specified in this parameter are the names of user exit types, *not* the names of individual user exit modules.

BPE and each address space that can use BPE have different user exit types. BPE’s user exit types, as specified by OWNER(BPE), include the following:

INITTERM	Initialization-Termination user exit
STATS	BPE system functions statistics user exit

User exit types are defined in all CQS address spaces, as specified by OWNER(CQS), and include the following:

CLNTCONN	Client Connection user exit
INITTERM	Initialization-Termination user exit
OVERFLOW	Queue Overflow user exit
STRSTAT	Structure statistics user exit
STREVENT	Structure event user exit

User exit types are defined in all OM address spaces, as specified by OWNER(OM), and include the following:

CLNTCONN	Client Connection command registration and deregistration user exit
INITTERM	Initialization-Termination user exit
INPUT	Command input user exit

OUTPUT	Output user exit
SECURITY	Security checking user exit

User exit types are defined in all RM address spaces, as specified by OWNER(RM), and include the following:

CLNTCONN	Client Connection and Disconnection user exit
INITTERM	Initialization-Termination user exit

User exit types are defined in all SCI address spaces, as specified by OWNER(SCI), and include the following:

CLNTCONN	Client Connection and Disconnection user exit
INITTERM	Initialization-Termination user exit

Related Reading:

- See *IMS Version 9: Common Queue Server Guide and Reference* for more information about the CQS user exit routine types.
- See *IMS Version 9: Common Service Layer Guide and Reference* for more information about the OM, RM, and SCI user exit routine types.

OWNER(BPE | CQS | OM | RM | SCI)
An optional parameter that specifies the owner of the user exit type or types that you want to refresh. You can specify one of the following values:

BPE	For all IMS components that are running in a BPE address space.
CQS	For CQS address spaces only.
OM	For OM address spaces only.
RM	For RM address spaces only.
SCI	For SCI address spaces only.

The OWNER parameter acts as a filter to help you select the user exit types that you want to refresh. For example, you could specify `NAME(*) OWNER(CQS)` to refresh all of the CQS-defined user exit types in a CQS address space. If `OWNER` is omitted, then both BPE and component user exits can be refreshed (depending on the exits specified on `NAME`).

BPE REFRESH USEREXIT Command Output

The `REFRESH USEREXIT` command output consists of message, BPE0032I indicating that the command has completed:

```
BPE0032I REFRESH USEREXIT COMMAND COMPLETED
```

Command Example 1

Refresh all user exit modules.

Command:

```
F CQS1,REFRESH USEREXIT NAME(*)
```

Output:

```
BPE0032I REFRESH USEREXIT COMMAND COMPLETED
```

Command Example 2

Refresh all user exit modules for the OM command input and output exit types.

Command:

```
F OM,REF USRX NAME(INPUT,OUTPUT)
```

Output:

```
BPE0032I REF USRX COMMAND COMPLETED
```

Chapter 4. BPE User-Supplied Exit Routines

This chapter describes the Base Primitive Environment (BPE) user exit routines.

Note: Throughout this chapter the terms “user exit” and “user exit routine” mean “user-supplied exit routine.”

BPE user exit routines enable you to customize and monitor address spaces built on the Base Primitive Environment. BPE-defined user exit routine types are available to all IMS component address spaces that run with BPE. You write these exit routines. No sample exit routines are provided. The BPE user exit routines are given control in the address space in an authorized state.

Following is a list of the user exit routines and their functions.

BPE Initialization-Termination
Called during BPE initialization and normal BPE termination.

BPE Statistics
Called at regular intervals during the life of a BPE address space, and a final time at normal address shutdown, to gather address-space related statistics.

Related Reading: For complete information about BPE interfaces and services that are available to user exit routines, see Chapter 5, “BPE User-Supplied Exit Routine Interfaces and Services,” on page 61.

In this chapter:

- “General BPE User-Supplied Exit Routine Information”
- “BPE Initialization-Termination User-Supplied Exit Routine”
- “BPE Statistics User-Supplied Exit Routine” on page 49

This chapter contains Product-sensitive Programming Interface information.

General BPE User-Supplied Exit Routine Information

Recommendation: Write BPE user exit routines in assembler, not in a high level language. BPE does not support exit routines that run under Language Environment® for z/OS. If you write an exit routine in a high level language, and that routine runs in the Language Environment for z/OS, you might have abends or performance problems. Language Environment for z/OS is designed for applications running in key 8, problem program state. BPE user exit routines execute in key 7 supervisor state.

Related Reading: For complete information about displaying and refreshing user exit routines, see Chapter 3, “BPE Commands,” on page 25.

BPE Initialization-Termination User-Supplied Exit Routine

The Initialization-Termination (Init-Term) exit routine is called during BPE initialization and BPE normal termination. The Init-Term exit routine is **not** called during BPE **abnormal** termination. This exit routine is optional.

The Init-Term exit routine is defined as TYPE = INITTERM, COMP=BPE in the EXITDEF statement in the BPE user exit PROCLIB member pointed to by the

EXITMBR statement for the BPE exit routines. You can specify one or more user exit routines of this type. When the init-term exit point is reached, the exit routines are driven in the order they are specified by the EXITS= keyword.

Recommendation: Write the Init-Term exit routine so that it is reentrant. The Init-Term exit routine is invoked AMODE 31.

Contents of Registers on Entry

Register	Contents
1	Address of Standard BPE user exit parameter list (mapped by the BPEUXPL macro). See “Standard BPE User Exit Parameter List” on page 62 for more information.
13	Address of two pre-chained save areas. The first save area can be used by the exit routine to save registers on entry. The second save area can be used by routines that are called from the user exit routine.
14	Return address.
15	Entry point of the exit routine.

Contents of Registers on Exit

Register	Contents
15	Return code **0** Always set this to zero.

All other registers must be restored.

BPE Initialization and Termination Parameter List

On entry to the Init-Term exit routine, R1 points to a Standard BPE user exit parameter list. Field UXPL_EXITPLP in this list contains the address of the Init-Term user exit routine parameter lists (mapped by the BPEITXP macro). Table 3 provides the following information about the BPE Init-Term user exit routine parameters:

- The field name
- The offset
- The length
- The field usage
- A description of the field

Table 3. BPE Init-Term User-Supplied Exit Routine Parameter List: BPE Initialization

Field Name	Offset	Length	Field Usage	Description
BPEITXP	X'00'	N/A	N/A	DSECT label for the BPE init-term exit parameter list
BPEITXP_VERSION	X'00'	X'04'	Input	Parameter List Version Number (00000001)
BPEITXP_FUNC	X'04'	X'04'	Input	Function code **1** BPE Initialization (BPEITXP_FUNC_INIT) **2** BPE Termination (BPEITXP_FUNC_TERM)

BPE Statistics User-Supplied Exit Routine

The BPE Statistics user exit routine enables you, at regular intervals, to gather statistics related to an IMS component that is running with a BPE address space. The exit routine is also called a final time during normal shutdown of the address space. The BPE Statistics user exit routine is optional.

The statistics exit routine is called on a time-driven basis. The interval between successive statistics exit routine calls is specified on the STATINTV parameter in the BPE configuration PROCLIB member. The exit routine is first called soon after BPE initialization completes. Subsequent calls occur every STATINTV seconds after the previous call returns.

The BPE statistics exit routine is also called one final time during normal address space shutdown processing. When it is called for normal shutdown, the function code passed in the BPESTXP parameter list will be BPESTXP_FUNC_FINALSTATS (2), indicating that this is the final statistics exit routine call.

The BPE Statistics user exit routine is defined as TYPE = STATS, COMP=BPE in the EXITDEF statement in the BPE user exit PROCLIB member. You can specify one or more user exit routines of this type. When this exit routine is invoked, all routines of this type are driven in the order specified by the EXITS= keyword.

Important: All statistics passed to the BPE Statistics user exit routine are considered Diagnosis, Modification, or Tuning Information.

Recommendation: Write the BPE Statistics exit routine so that it is reentrant. It is invoked AMODE 31.

Contents of Registers on Entry

Register	Contents
1	Address of Standard BPE user exit parameter list (mapped by the BPEUXPL macro). See “Standard BPE User Exit Parameter List” on page 62 for more information.
13	Address of two pre-chained save areas. The first save area can be used by the exit routine to save registers on entry. The second save area can be used by routines that are called from the user exit routine.
14	Return address.
15	Entry point of the exit routine.

Contents of Registers on Exit

Register	Contents
15	Return code **0** Always set this to zero.

All other registers must be restored.

BPE Statistics Exit Routine Parameter List

On entry to the Statistics exit routine, R1 points to a Standard BPE user exit parameter list. Field UXPL_EXITPLP in the Standard BPE user exit parameter list contains the address of the BPE Statistics user exit routine parameter list (mapped by the BPESTXP macro). Table 4 provides the following information about the Statistics user exit routine parameters:

- The field name
- The offset
- The length
- The field usage
- A description of the field

Table 4. BPE Statistics User-Supplied Exit Routine Parameter List

Field Name	Offset	Length	Field Usage	Description
BPESTXP	X'00'	N/A	N/A	DSECT label for the BPE statistics exit parameter list
BPESTXP_VERSION	X'00'	X'04'	Input	Parameter List Version Number (00000001)
BPESTXP_FUNC	X'04'	X'04'	Input	Function code **1** Statistics (BPESTXP_FUNC_STATS) **2** Final statistics (BPESTXP_FUNC_FINALSTATS)
BPESTXP_BPESTATS_PTR	X'08'	X'04'	Input	Address of BPE system statistics area header. This header points to detailed BPE system statistics. All of the BPE statistics areas are mapped by macro BPESSTA. See Table 5 on page 52 for a description of the BPE system statistics area header.
BPESTXP_COMPSTATS_PTR	X'0C'	X'04'	Input	Address of the IMS component statistics area, or zero if none. An IMS component that runs with BPE has the ability to define its own statistics area, to be passed along with the BPE statistics area when the BPE statistics exit is called. However, not all IMS components provide their own statistics in that manner. If a component does not provide statistics, this field in the BPESTXP parameter list is zero. Refer to the *IMS Version 9: Common Queue Server Guide and Reference* or the *IMS Version 9: Common Service Layer Guide and Reference* as required to see if a specific IMS component provides a statistics area and to see the format of the area.

BPE System Statistics Area

The BPE system statistics area contains statistics on the following system resources managed by BPE.

- TCBs
- Control block services
- AWE servers
- Storage services

The field BPESTXP_BPESTATS_PTR in the BPE statistics exit parameter list points to this area. Figure 9 shows the structure of the BPE system statistics area.

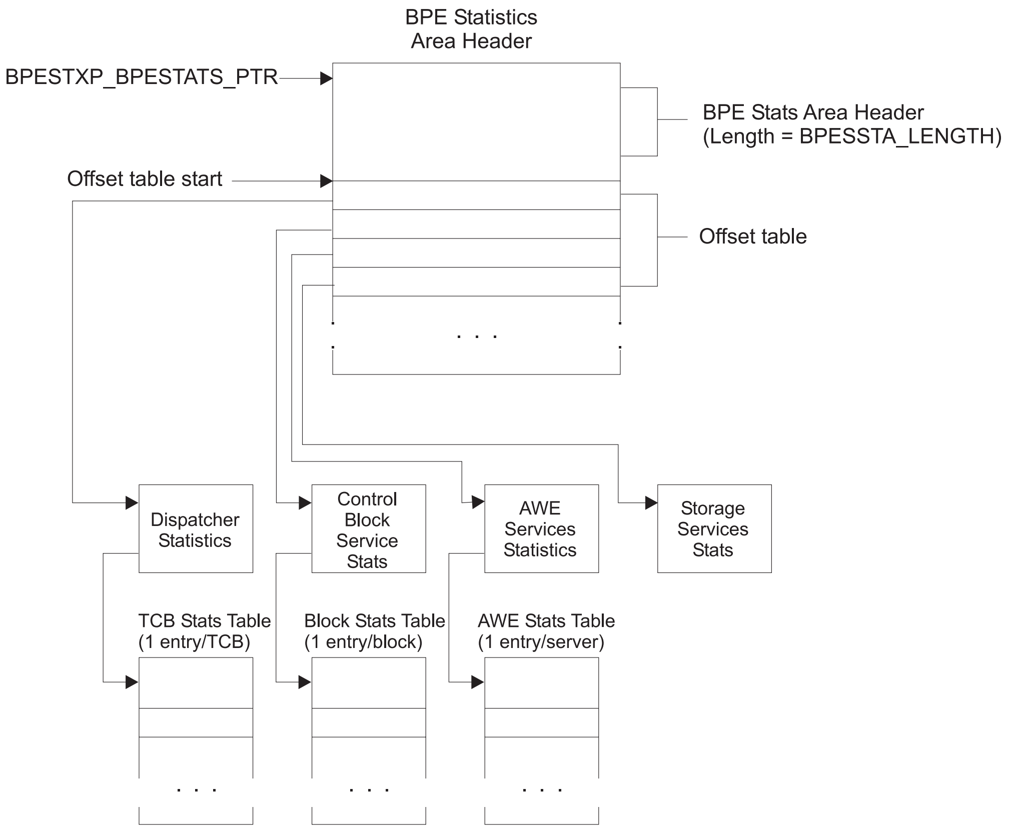

BPE Statistics Area Header
BPESTXP_BPESTATS_PTR
BPE Stats Area Header (Length = BPESSTA_LENGTH)
Offset table start
Offset table
Dispatcher Statistics
Control Block Service Stats
AWE Services Statistics
Storage Services Stats
TCB Stats Table (1 entry/TCB)
Block Stats Table (1 entry/block)
AWE Stats Table (1 entry/server)

Figure 9. BPE System Statistics Area Structure

The BPE system statistics area begins with the BPESSTA header. The header contains general information about the BPE address space and the IMS component running in it. The offset table appears immediately after the header (BPESSTA + SSTA_LENGTH). Each area for which statistics are reported is assigned a fixed slot within this table. Each slot contains the offset to the particular area's statistics block from the start of the offset table. Each area may have one or more blocks for the statistics pertaining to the area.

All "pointers" among the BPE system statistics area blocks are really offsets, not addresses. Having offsets allows statistics to be written to a log or other data set, where the original block addresses are no longer meaningful. All offsets are relative to the beginning of the DSECT in which the offset field resides.

The total length of the BPE statistics area is not fixed (static). The length depends on the resource definitions and number of active resources in the system. Many of the area blocks contain entries for each resource type.

Recommendation: Always use the lengths passed in the area fields to refer to the length of a particular statistics area. Do not use lengths generated as EQUs (assembler equates) at assembly time. Using the passed lengths ensures that your exit routine code works correctly, even if the format of the statistics areas changes in the future.

Unless otherwise indicated, the following statements are true:

- All statistics in the various BPE statistics area sections are cumulative since the start of the address space.
- Count fields are 32-bit unsigned numbers.
- Time-related double-word fields are in STCK units (bit 12 = 1 microsecond).
- Statistics are gathered without serialization for performance reasons. As a result, the statistics might not be completely consistent with each other. For example, two related statistics might be updated at different times. View the statistics as aggregate indications of the system performance, not as exact values.

Table 5 provides the following information about the fields in the BPE system statistics area:

- The field name
- The offset
- The length
- The field usage
- A description of the field

Table 5. BPE System Statistics Area

Field Name	Offset	Length	Field Usage	Description
BPESSTA	X'00'	N/A	N/A	DSECT label for the BPE system statistics area.
SSTA_ID	X'00'	X'08'	Input	Eyecatcher ("BPESSTA ").
SSTA_LENGTH	X'08'	X'04'	Input	BPESSTA header section length (SSTA_END minus BPESSTA). The offset table starts immediately after the BPESSTA header (BPESSTA + SSTA_LENGTH).
SSTA_VER	X'0C'	X'04'	Input	BPESSTA header version number within a BPE release. The current version is X'00000001' (SSTA_VER_1).
SSTA_BPEVER	X'10'	X'03'	Input	BPE version number.
	X'13'	X'01'	Input	Reserved.
SSTA_OFSTTBLLEN	X'14'	X'04'	Input	Length of offset table.
SSTA_UTYPE	X'18'	X'04'	Input	IMS component type.
SSTA_UVERSION	X'1C'	X'03'	Input	IMS component version number.
	X'1F'	X'01'	Input	Reserved.
SSTA_USYSNAME	X'20'	X'08'	Input	IMS component system name.
SSTA_JOBNAME	X'28'	X'08'	Input	Jobname of address space for which this record was created.
SSTA_STARTSTCK	X'30'	X'08'	Input	STCK at BPE start (STCK when BPE jobstep TCB was created).
SSTA_STCK	X'38'	X'08'	Input	STCK when this record was created.
STCK_LDTO	X'40'	X'08'	Input	Local time-date offset from field CVTLDTO in the CVT (the amount to add to a UTC STCK to get local time STCK, in STCK units).
SSTA_CPUID	X'48'	X'08'	Input	CPU ID from STIDP instruct.
SSTA_UPRODNUM	X'50'	X'08'	Input	Product number (comp-ID) of IMS component, in the form of *nnnn-nnn*. This field is present only if the BPE version (SSTA_BPEVER) is X'010400' or greater.

Table 5. BPE System Statistics Area (continued)

Field Name	Offset	Length	Field Usage	Description
SSTA_OSNAME	X'58'	X'08'	Input	Operating system name (from field CVTSNAME in the ECVT). This field is present only if the BPE version (SSTA_BPEVER) is X'010400' or greater.
SSTA_OSPNAME	X'60'	X'10'	Input	Operating system product name (from field ECVTPNAM in the ECVT). This field is present only if the BPE version (SSTA_BPEVER) is X'010400' or greater.
SSTA_OSPVER	X'70'	X'02'	Input	Operating system version, in EBCDIC (from field ECVTPVER in the ECVT). This field is present only if the BPE version (SSTA_BPEVER) is X'010400' or greater.
SSTA_OSPREL	X'72'	X'02'	Input	Operating system release, in EBCDIC (from field ECVTPVER in the ECVT). This field is present only if the BPE version (SSTA_BPEVER) is X'010400' or greater.
SSTA_OSPMOD	X'74'	X'02'	Input	Operating system modification level, in EBCDIC (from field ECVTPREL in the ECVT). This field is present only if the BPE version (SSTA_BPEVER) is X'010400' or greater.
SSTA_SYSCLONE	X'76'	X'02'	Input	SYSCLONE value (from field ECVTCLON in the ECVT). This field is present only if the BPE version (SSTA_BPEVER) is X'010400' or greater.
SSTA_TOTALLEN	X'78'	X'04'	Input	Total length of all statistics areas. The total length is the number of bytes from the start of the BPESSTA to the last byte of statistics data. You can use this field if you are copying the statistics data to another location (for example, to a data set) to determine the length of the data to copy. This field is present only if the BPE version (SSTA_BPEVER) is X'010400' or greater.
	X'7C'	X'04'	Input	Reserved.

The BPE statistics offset table is immediately after the BPE statistics header (BPESSTA + SSTA_LENGTH). The offset table contains offsets to the various statistics blocks in the area.

Attention: The values in the table are offsets from the start of the offset table, not from BPESSTA. You must use the offset table to locate the different statistics sections to allow for changes to the lengths of these sections.

Figure 10 on page 54 is an example of how to locate the dispatcher statistics area, assuming that R2 points to the BPESSTA header:

```
         USING      BPESSTA,R2              Address SSTA header
         LR       R3,R2                   Copy SSTA header addr
          AL        R3,SSTA_LENGTH           Add length to get ofst tble addr
         USING    SSTA_OFSTTBL,R3         Address offset table
         ICM      R4,15,SSTA_OFST_DISP    Any dispatcher section?
         BZ       NODSP                   No, can't access it
         ALR      R4,R3                   Add ofst tbl start to get addr
         USING    SSTADS,R4               Address dispatcher section
         . . .
NODSP    DS       OH                      To here if no disp sect present
```

Figure 10. Locating the Dispatcher Statistics Area

Check offset fields for values of zero before using them. A zero offset field means that the particular statistics block is not present in the area.

Table 6 provides the following information about the fields in the BPE statistics offset table:
- The field name
- The offset
- The length
- The field usage
- A description of the field

Table 6. BPE Statistics Offset Table

Field Name	Offset	Length	Field Usage	Description
SSTA_OFSTTBL	X’00’	N/A	N/A	DSECT label for the BPE statistics offset table.
SSTA_OFST_DISP	X’00’	X’04’	Input	Offset to dispatcher statistics.
SSTA_OFST_CBS	X’04’	X’04’	Input	Offset to control block services statistics.
SSTA_OFST_AWE	X’08’	X’04’	Input	Offset to AWE statistics.
SSTA_OFST_STG	X’0C’	X’04’	Input	Offset to general storage statistics. This field is present only if the BPE version (SSTA_BPEVER) is X’010400’ or greater.

Table 7 provides the following information about the fields in the BPE dispatcher statistics area:
- The field name
- The offset
- The length
- The field usage
- A description of the field

Table 7. BPE Dispatcher Statistics Area

Field Name	Offset	Length	Field Usage	Description
SSTADS	X’00’	N/A	N/A	DSECT label for BPE dispatcher statistics area.
SSTADS_ID	X’00’	X’04’	Input	Dispatcher section eyecatcher (“DISP”).
SSTADS_LENGTH	X’04’	X’04’	Input	Length of dispatcher section (includes TCB statistics table).
SSTADS_VERSION	X’08’	X’04’	Input	Dispatcher statistics version number. The current version is X’00000001’ (SSTADS_VER_1).

Table 7. BPE Dispatcher Statistics Area (continued)

Field Name	Offset	Length	Field Usage	Description
SSTADS_TBLOFST	X'0C'	X'04'	Input	Offset from SSTADS to the first TCB statistics table entry.
SSTADS_NUMENT	X'10'	X'02'	Input	Number of TCB statistics table entries.
SSTADS_ENTLEN	X'12'	X'02'	Input	Length of each TCB statistics entry.
SSTADS_THD#	X'14'	X'04'	Input	Global number of thread starts.
SSTADS_DISP#	X'18'	X'04'	Input	Global number of dispatches.
	X'1C'	X'04'	Input	Reserved.
SSTADS_TREALTM	X'20'	X'08'	Input	Real time (wall clock time) that the BPE address space has been running (in STCK units). This field is present only if the BPE version (SSTA_BPEVER) is X'010400' or greater.
SSTADS_TBPETM	X'28'	X'08'	Input	Total time that all BPE-managed TCBs have been dispatched by the BPE dispatcher (in STCK units). This time is cumulative and includes time for TCBs that have terminated, as well as those that are still active. This field is present only if the BPE version (SSTA_BPEVER) is X'010400' or greater.
SSTADS_TCPUTM	X'30'	X'08'	Input	Total CPU time used by all BPE-managed TCBs (in STCK units). This time is cumulative and includes time for TCBs that have terminated, as well as those that are still active. This field is present only if the BPE version (SSTA_BPEVER) is X'010400' or greater.

Table 8 provides the following information about the BPE TCB statistics table entry:
- The field name
- The offset
- The length
- The field usage
- A description of the field

Each active TCB in the system has one table entry. In the case of TCB types that support multiple TCBs, each instance of a TCB has one entry.

Important: The BPE and the IMS component running on the BPE can define a TCB type with the same name. Use the SSTADS_F1_SYS flag to differentiate between the TCBs in this case.

Table 8. BPE TCB Statistics Table Entry

Field Name	Offset	Length	Field Usage	Description
SSTADS_TTE	X'00'	N/A	N/A	DSECT label for BPE TCB statistics table entry.
SSTADS_TYPE	X'00'	X'04'	Input	TCB type.
SSTADS_FLG1	X'04'	X'01'	Input	DDB flag 1 (unlabeled bits are reserved by IBM®). **SSTADS_F1_POOL (X'08')** TCB is a pool-type TCB. **SSTADS_F1_SYS (X'10')** TCB is BPE-defined.
SSTADS_FLG2	X'05'	X'01'	Input	DDB flag 2 (unlabeled bits are reserved by IBM).

Table 8. BPE TCB Statistics Table Entry (continued)

Field Name	Offset	Length	Field Usage	Description
SSTADS_IDX	X'06'	X'01'	Input	TCB index number.
SSTADS_INUM	X'07'	X'01'	Input	TCB instance number.
SSTADS_BPE_TCBTKN	X'08'	X'08'	Input	TCB token (unique value identifying this TCB).
SSTADS_#THDCR	X'10'	X'04'	Input	Number of thread creates.
SSTADS_#THDDL	X'14'	X'04'	Input	Number of thread deletes.
SSTADS_#THDSTART	X'18'	X'04'	Input	Number of thread starts.
SSTADS_#THDDISP	X'1C'	X'04'	Input	Number of thread dispatches.
SSTADS_#SUSP	X'20'	X'04'	Input	Number of suspends.
SSTADS_#SUSPBKO	X'24'	X'04'	Input	Number of backed-out suspends.
SSTADS_REALTIME	X'28'	X'08'	Input	Wall-clock time TCB has been running (in STCK units).
SSTADS_BPETIME	X'30'	X'08'	Input	Time TCB has been dispatched by BPE dispatcher (in STCK units).
SSTADS_CPUTIME	X'38'	X'08'	Input	CPU time for this TCB (in STCK units).

Note:

- Entries in this table can remain unused if a TCB terminates after the statistics module computes the number of entries in the table, but before all of the statistics are captured.
- The entries in this table might not be in the same order every time the statistics area is generated. You must use the 8-byte TCB token to associate entries if you are computing entries from two different statistics exit calls.
- The number of entries in the statistics table might change from one BPE statistics user exit call to the next. TCBs are dynamic and might be created and destroyed as part of normal processing. The TCB statistics table represents the TCBs currently active at the time the statistics exit routine was called.

Table 9 provides the following information about the fields in the BPE control block services (CBS) statistics area:

- The field name
- The offset
- The length
- The field usage
- A description of the field

The control blocks services area contains a header with global statistics and information, followed by a table with one entry for each CBS-defined block type in the system.

Table 9. BPE Control Block Services Statistics Area

Field Name	Offset	Length	Field Usage	Description
SSTACB	X'00'	N/A	N/A	DSECT label for BPE control block services statistics area.
SSTACB_ID	X'00'	X'04'	Input	Control block services section eyecatcher ("CBS").
SSTACB_LENGTH	X'04'	X'04'	Input	Length of CBS section (includes block statistics table).
SSTACB_VERSION	X'08'	X'04'	Input	Control block services statistics version number. The current version is X'00000001' (SSTACB_VER_1).
SSTACB_TBLOFST	X'0C'	X'04'	Input	Offset from SSTACB to first control block statistics table entry.
SSTACB_NUMENT	X'10'	X'02'	Input	Number of control block statistics table entries.

Table 9. BPE Control Block Services Statistics Area (continued)

Field Name	Offset	Length	Field Usage	Description
SSTACB_ENTLEN	X'12'	X'02'	Input	Length of each control block statistics table entry.

Table 10 provides the following information about the BPE control block statistics table entry:

- The field name
- The offset
- The length
- The field usage
- A description of the field

Each control block type in the system has one table entry.

Important: The BPE and the IMS component running on the BPE can define a block type with the same name. Use the SSTACB_F1_SYS flag to differentiate between the blocks in this case.

Table 10. BPE Control Block Statistics Table Entry

Field Name	Offset	Length	Field Usage	Description
SSTACB_BTE	X'00'	N/A	N/A	DSECT label for BPE control block statistics table entry.
SSTACB_TYPE	X'00'	X'04'	Input	Block type.
SSTACB_FLG1	X'04'	X'01'	Input	CBTE flag 1 (unlabeled bits are reserved by IBM). **SSTACB_F1_COMP (X'20')** Blocks are compressible. **SSTACB_F1_SYS (X'10')** Block is a BPE block. **SSTACB_F1_FIXED (X'08')** Block storage is page fixed.
SSTACB_FLG2	X'05'	X'01'	Input	CBTE flag 2 (unlabeled bits are reserved by IBM). **SSTACB_F2_31ONLY (X'10')** Block is in 31-bit only storage. **SSTACB_F2_PAGE (X'02')** Get BPAGE on 4K page boundary. **SSTACB_F2_ANY (X'01')** Block in LOC=ANY storage.
SSTACB_IDX	X'06'	X'01'	Input	Block index number.
SSTACB_SP	X'07'	X'01'	Input	Block storage subpool.
SSTACB_#GET	X'08'	X'04'	Input	Number of gets for this block type.
SSTACB_CURBYTES	X'0C'	X'04'	Input	Current number bytes in pool.
SSTACB_MAXBYTES	X'10'	X'04'	Input	Maximum number bytes in pool.
SSTACB_#GETMAIN	X'14'	X'04'	Input	Number of GETMAINs for BPAGEs.
SSTACB_#FREEMAIN	X'18'	X'04'	Input	Number of FREEMAINs for BPAGEs.
SSTACB_CURBLKS	X'1C'	X'04'	Input	Current number blocks in pool.

Table 11 on page 58 provides the following information about the BPE AWE services statistics area:

- The field name
- The offset
- The length
- The field usage
- A description of the field

The AWE services area contains a header with global statistics and information, followed by a table with one entry for each instance of an AWE server running in the system.

Table 11. BPE AWE Services Statistics Area

Field Name	Offset	Length	Field Usage	Description
SSTAAW	X’00’	N/A	N/A	DSECT label for BPE AWE services statistics area.
SSTAAW_ID	X’00’	X’04’	Input	AWE services section eyecatcher (“AWE”).
SSTAAW_LENGTH	X’04’	X’04’	Input	Length of AWE section (includes AWE server statistics table).
SSTAAW_VERSION	X’08’	X’04’	Input	AWE services statistics version number. The current version is X’00000001’ (SSTAAW_VER_1).
SSTAAW_TBLOFST	X’0C’	X’04’	Input	Offset from SSTAAW to first AWE statistics table entry.
SSTAAW_NUMENT	X’10’	X’02’	Input	Number of AWE server statistics table entries.
SSTAAW_ENTLEN	X’12’	X’02’	Input	Length of each AWE server statistics table entry.

Table 12 provides the following information about the BPE AWE services statistics table entry:

- The field name
- The offset
- The length
- The field usage
- A description of the field

Each active AWE server running in the system has one table entry.

Important: It is possible for BPE and the IMS component running on the BPE to define an AWE server type with the same name. Use the SSTAAW_F1_SYS flag to differentiate between the identically-named BPE and user-product AWE servers.

Table 12. BPE AWE Services Statistics Table Entry

Field Name	Offset	Length	Field Usage	Description
SSTAAW_ASTE	X’00’	N/A	N/A	DSECT label for AWE server statistics table entry.
SSTAAW_TYPE	X’00’	X’04’	Input	AWE server type.
SSTAAW_SERVID	X’04’	X’04’	Input	Unique server ID number.

Table 12. BPE AWE Services Statistics Table Entry (continued)

Field Name	Offset	Length	Field Usage	Description
SSTAAW_FLG1	X'08'	X'01'	Input	Flag 1 (AQSB_FLG1) (unlabeled bits are reserved by IBM). **SSTAAW_F1_INIT X'80'** Server init is in progress. **SSTAAW_F1_TERM X'40'** All servers should terminate. **SSTAAW_F1_MULTI X'20'** Queue is multi-server.
SSTAAW_FLG2	X'09'	X'01'	Input	Flag 2 (AQHE_FLG1) (unlabeled bits are reserved by IBM). **SSTAAW_F2_GENERIC X'80'** Generic AWE server. **SSTAAW_F2_AUTO X'40'** Server is AUTOSTARTed. **SSTAAW_F2_SYSTCB X'20'** Server runs under system TCB. **SSTAAW_F2_SYS X'10'** System (BPE) server. **SSTAAW_F2_LOC24 X'08'** Thread blks in 24-bit storage. **SSTAAW_F2_FORCEMAX X'04'** Force max threads on AUTOSTART.
SSTAAW_TCBID	X'0A'	X'01'	Input	ID number of owning TCB.
SSTAAW_NUMTHDS	X'0B'	X'01'	Input	Number of server threads for this queue header.
SSTAAW_QHDR	X'0C'	X'04'	Input	Address of AWE queue header.
	X'10'	X'04'	Input	Reserved.
SSTAAW_TQCOUNT	X'14'	X'04'	Input	Number times an extra server was woken up off of the AQSB_THREADQ (multi-server queue headers only).
SSTAAW_NUMAWE	X'18'	X'1C'	Input	Number of AWEs processed off of this queue header.
SSTAAW_NUMEQS	X'1C'	X'04'	Input	Number of times one or more AWEs were dequeued from this queue header (NUMAWE/NUMDEQS is the average number of AWEs on the queue header).
SSTAAW_PROCTIME	X'20'	X'08'	Input	Cumulative time spent in processing routine for this queue header, in STCK units.
SSTAAW_NOWORK	X'28'	X'04'	Input	Number of times an AWE server was woken up and found no work (multi-server queue headers only).
SSTAAW_READYWAIT	X'2C'	X'04'	Input	Number of times an AWE server had to wait for access to the AWE ready queue (multi-server queue headers only).

Table 13 on page 60 provides the following information about the BPE storage services statistics area:

- The field name
- The offset
- The length
- The field usage

- A description of the field

Table 13. BPE Storage Services Statistics Area

Field Name	Offset	Length	Field Usage	Description
SSTASG	X'00'	N/A	N/A	DSECT label for BPE storage services statistics area.
SSTASG_ID	X'00'	X'04'	Input	Storage services section eyecatcher ("STG").
SSTASG_LENGTH	X'04'	X'04'	Input	Length of storage section.
SSTASG_VERSION	X'08'	X'04'	Input	Storage services statistics version number. The current version is X'00000001' (SSTASG_VER_1).
	X'0C'	X'0C'	Input	Reserved.
SSTASG_STGPVT24	X'18'	X'04'	Input	Number of bytes of private storage currently allocated in 24-bit storage by the BPE GETMAIN service, BPEGETM. Note that the values for the stack, control block, and buffer pool services (below) are included in this number.
SSTASG_STGPVT31	X'1C'	X'04'	Input	Number of bytes of private storage allocated in 31-bit storage by the BPE GETMAIN service, BPEGETM. Note that the values for the stack, control block, and buffer pool services (below) are included in this number.
SSTASG_STKPVT24	X'20'	X'04'	Input	Number of bytes of private storage currently allocated in 24-bit storage by the BPE stack manager service.
SSTASG_STKPVT31	X'24'	X'04'	Input	Number of bytes of private storage currently allocated in 31-bit storage by the BPE stack manager service.
SSTASG_CBPVT24	X'28'	X'04'	Input	Number of bytes of private storage currently allocated in 24-bit storage by the BPE control block service.
SSTASG_CBPVT31	X'2C'	X'04'	Input	Number of bytes of private storage currently allocated in 31-bit storage by the BPE control block service.
SSTASG_BPPVT24	X'30'	X'04'	Input	Number of bytes of private storage currently allocated in 24-bit storage by the BPE buffer pool service.
SSTASG_BPPVT31	X'34'	X'04'	Input	Number of bytes of private storage currently allocated in 31-bit storage by the BPE buffer pool service.

Chapter 5. BPE User-Supplied Exit Routine Interfaces and Services

This chapter describes the Base Primitive Environment (BPE) user exit routine interfaces and services in detail.

Note: Throughout this chapter the term “user exit routine” means “user-supplied exit routine.”

Related Reading: For information about the Base Primitive Environment’s own user exit routines, see Chapter 4, “BPE User-Supplied Exit Routines,” on page 47.

In this chapter:

This chapter contains Product-sensitive Programming Interface information.

General BPE User-Supplied Exit Routine Interface Information

Some IMS components (for example, CQS, OM, RM, and SCI) use BPE services to define and manage calls to user exit routines. BPE also has its own user exit routines (described in Chapter 4, “BPE User-Supplied Exit Routines,” on page 47). For such exit routines, BPE gives you the ability to externally specify the user exit routine modules to be called for a particular exit routine type using EXITDEF statements in BPE user exit PROCLIB members. BPE also provides a common user exit routine run time environment. The run time environment includes the following:

- A Standard BPE user exit parameter list
- Static work areas for the routines
- Dynamic work areas for the routines
- Callable services for the routines
- A recovery environment to protect against abends in the user exit routines

Recommendation: Write BPE user exit routines in assembler, not in a high level language. BPE does not support exit routines that run under Language Environment for z/OS. If you write an exit routine in a high level language, and that routine runs in the Language Environment for z/OS, you might have abends or performance problems. Language Environment for z/OS is designed for applications that run in key 8, problem program state. BPE user exit routines run in key 7 supervisor state.

Related Reading: For complete information about displaying and refreshing user exit routines, see Chapter 3, “BPE Commands,” on page 25.

Standard BPE User Exit Parameter List

All BPE-managed user exit routines receive a pointer to a Standard BPE user exit parameter list in R1. The format of this parameter list is the same for all exit routines, and is mapped by the BPEUXPL DSECT (in the BPEUXPL macro).

Table 14 provides the following information about the fields in the Standard BPE user exit parameter list:

- The field name
- The offset
- The length
- The field usage
- A description of the field

Table 14. Standard BPE User Exit Parameter List

Field Name	Offset	Length	Field Usage	Description
BPEUXPL	X'00'	N/A	N/A	DSECT label for Standard BPE user exit parameter list.
UXPL_VERSIONP	X'00'	X'04'	Input	Pointer to a word containing the Standard BPE user exit parameter list version number. The current version of the parameter list is X'00000002'. (EQU symbol UXPL_VER2.)
UXPL_CSTOKENP	X'04'	X'04'	Input	Pointer to the BPE callable services token.
UXPL_STATICWAP	X'08'	X'04'	Input	Pointer to a 256-byte static work area. Each exit routine module is assigned its own static work area. The contents of the static work area are preserved from call to call.
UXPL_DYNAMICWAP	X'0C'	X'04'	Input	Pointer to a 512-byte dynamic work area. This area is intended as working storage for a user exit routine for the duration of that exit routine's run. The contents of this area are not preserved from call to call.
UXPL_EXITPLP	X'10'	X'04'	Input	Pointer to an exit-type-specific parameter list. The exit-type-specific parameter list contains fields that are unique to the type of exit routine being called.
UXPL_CALLNEXTP	X'14'	X'04'	Input	Pointer to a byte of storage that the user exit routine can use to indicate whether to call other subsequent exit routines of the same type for the current instance of the exit routine call.
UXPL_COMPTYPEP	X'18'	X'04'	Input	Pointer to a four-byte character string containing the IMS component type for the address space in which the exit routine is being called. The string is left-justified and padded with blanks as needed to make it a four-byte string. Possible values are: **CQS** Common Queue Server; **OM** Operations Manager; **RM** Resource Manager; **SCI** Structured Call Interface. **Important:** This field is present only when the word pointed to by UXPL_VERSIONP is equal to the value of UXPL_VER2 or greater.

Table 14. Standard BPE User Exit Parameter List (continued)

Field Name	Offset	Length	Field Usage	Description
UXPL_COMPVERP	X'1C'	X'04'	Input	Pointer to a three-byte field in storage containing the version number of the IMS component for the address space in which the exit routine is being called. The version number is of the form *vvrrpp*, where: *vv* Component version number. *rr* Component release number. *pp* Component point release number. For example, if the address space type were CQS in IMS Version 9, UXPL_COMPVERP points to X'010400'. **Important:** This field is present only when the word pointed to by UXPL_VERSIONP is equal to the value of UXPL_VER2 or greater.
UXPL_BPEVERP	X'20'	X'04'	Input	Pointer to a three-byte field in storage containing the BPE version number for the address space in which the exit routine is being called. The version number is of the form *vvrrpp*, where: *vv* BPE version number. *rr* BPE release number. *pp* BPE point release number. For example, if the address space type were BPE in IMS Version 9, UXPL_BPEVERP points to X'010500'. **Important:** This field is present only when the word pointed to by UXPL_VERSIONP is equal to the value of UXPL_VER2 or greater.
UXPL_SYSIDP	X'24'	X'04'	Input	Pointer to an 8-character system ID. The system ID is a character ID string that may be used by the IMS component using BPE services (for example, the CQS ID that is derived from the CQS SSN= startup parameter). If the IMS component has not provided a system ID to BPE, then the field pointed to by this pointer will be all blanks. If the system ID is shorter than eight characters, it is padded on the right with blanks to make it eight characters. **Important:** This field is present only when the word pointed to by UXPL_VERSIONP is equal to the value of UXPL_VER2 or greater.

Work Areas Provided by BPE

Each user exit routine is passed two work areas by BPE every time the exit routine is called. The two work areas are:

- The static work area
- The dynamic work area

The Static Work Area

The static work area is pointed to by field UXPL_STATICWAP in the Standard BPE user exit parameter list. The static work area is 256 bytes in length. Each user exit routine module is assigned its own static work area that is not shared between exit routine modules of the same type. The same work area is passed every time a particular user exit routine module is called, and the contents of the work area are preserved from call to call. A user exit routine module can use the static work area to save data between calls to the exit routine. The static work area is cleared (set to zeros) the first time a user exit routine is invoked.

When a user exit module is refreshed with the REFRESH USEREXIT command, the same static work area continues to be passed to the new copy of the module that was being passed to the old copy. If a user exit module is removed from an EXITDEF list and a REFRESH USEREXIT command is issued, the static work area for the module is deleted. If the exit module is then later added back to the EXITDEF list and another REFRESH USEREXIT command is issued, the exit routine gets a new (cleared) static work area. For more information about handling the static work area across REFRESH USEREXIT commands, see "Considerations for Refreshing User Exits" on page 42.

The Dynamic Work Area

The dynamic work area is pointed to by field UXPL_DYNAMICWAP in the Standard BPE user exit parameter list. The dynamic work area is 512 bytes in length. The dynamic work area is used as working storage by a user exit routine for the current call only. The dynamic storage area's address might not be the same, nor are its contents preserved from call to call. The dynamic work area is not cleared when a user exit routine receives control; therefore, the work area might contain residual data.

Calling Subsequent Exit Routines in BPE

Each user exit routine type can have multiple exit routine modules associated with it. BPE calls each module in the order that it was specified on the EXITS= parameter of the EXITDEF= statement. The EXITDEF= statement of the BPE user exit PROCLIB member defines the list of exit routines.

Each user exit routine can decide whether subsequent exit routines in the list that are to be called upon return to BPE. For example, a list of exit routines are called to make a decision about processing for a particular resource. If exit routine ABC cannot make the decision, it can return an indication that the next exit routine in the list, routine DEF, is to be called so that it can try to make the decision. If exit routine ABC is able to make the decision, it can return an indication that the next exit routine in the list, routine DEF, need not be called because the decision has already been reached.

Field UXPL_CALLNEXTP in the Standard BPE user exit parameter list is a pointer to a byte in storage that the user exit routine can use to indicate whether to call the next exit routine in the list. If the exit routine does not set this byte, the default is to call the next exit routine in the list. If the exit routine sets this byte, it must set it to one of the following values, defined by EQUs in the BPEUXPL macro:

UXPL_CALLNEXTYES	Call the next exit routine in the list.
UXPL_CALLNEXTNO	Do not call the next exit routine in the list.

Attention: Only UXPL_CALLNEXTYES and UXPL_CALLNEXTNO are defined values for this byte. Results are unpredictable if a user exit routine sets this byte to any value **other than** those listed here.

BPE User-Supplied Exit Routine Environment

All user exit routines are given control in the following environment unless otherwise stated:

Authorization	Supervisor state, PSW key 7
Dispatchable unit mode	TCB
Cross-memory mode	None (PASN=HASN=SASN)
AMODE	31
ASC mode	Primary
Interrupt Status	Enabled
Locks	None

All user exit routines receive control with the following registers set:

Register	**Contents**
R1	Pointer to Standard BPE user exit parameter list.
R13	Pointer to the first of two pre-chained save areas. The user exit routine can use the first save area to save the registers of its caller, and can use the second save area for lower-level calls that it makes. The save areas are chained together using standard z/OS save area linkage conventions.
R14	Return address.
R15	Entry point of exit routine.

Attention: Control must be returned to the return address passed to the user exit routine in R14. R15 can be set to a return code if appropriate for the specific exit routine type being called. Ensure that all other registers are restored to the values they had when the exit routine was called.

The contents of the registers not listed here are unknown and unpredictable.

Ensure that your user exit routines do not modify any fields in any parameter list that are not explicitly documented as output fields. The results of modifying non-output fields are unpredictable.

Write your user exit routines so that they are reentrant. User exit routines in the same EXITS= list are called serially within one occurrence of a call for that exit routine type. However, it is possible for a user exit routine module to be entered simultaneously for different occurrences of a call, under different TCBs, for the same exit routine type.

An exit routine receives the same static work area, but receives another dynamic work area for each call when it is entered simultaneously. Be careful when updating fields in the static work area. They might be in the process of being changed by other instances of your exit routine module that are running in parallel.

BPE User Exit Routine Performance Considerations

Some user exit routines might be called from mainline processing code. The amount and type of processing that is done by those exit routines can directly contribute to the total path length and time required to complete a unit of work.

Recommendation: Code user-supplied exit routines in ways that minimize path length and processing time as much as possible.

Operating system WAITs, SVCs, and I/O can all contribute to poor performance and should be used sparingly. When a BPE callable service exists, it is recommended that you use it, rather than the operating system equivalent, because the callable service is usually optimized to perform more efficiently in a BPE sub-dispatching environment.

Recommendation: Code your user exit routines in assembler language for the best performance. If you write exit routines in other languages, you might have performance problems. BPE does not support exit routines that run under Language Environment for z/OS.

Abends in BPE User-Supplied Exit Routines

BPE establishes a recovery environment before it calls user exit routines. In most cases, BPE recovers from any abends that occur while a user exit routine is in control, and calls the next exit routine in the list, if any is indicated. When a user exit routine abends, BPE ignores any value that the abending exit routine may have set in the byte pointed to by UXPL_CALLNEXTP. BPE resets this byte to UXPL_CALLNEXTYES and then calls the next exit routine in the list.

BPE keeps a count of the number of abends that have occurred in each user exit routine module. The first time an abend occurs in a module, BPE issues a request to create an SDUMP to capture diagnostic information about the abend. BPE also creates a SYS1.LOGREC entry for the abend and issues the message, BPE0019E, indicating which exit routine module had control when the abend occurred. For subsequent abends in an exit routine module, BPE creates a SYS1.LOGREC entry and issues the message, BPE0019E, but does not issue the request to create an SDUMP.

When the number of abends indicated by the ABLIM= parameter has been reached, BPE stops calling the abending exit routine module. The ABLIM= parameter is specified as part of the EXITDEF= statement for that type of exit routine. The default value for ABLIM= is 1 (to stop calling the exit routine after the first abend). You can change this value as required. The abend count for an exit routine is reset to zero if the exit routine type is refreshed (see "Refreshing User Exits in BPE" on page 41 for more information).

Related Reading: For more information on the ABLIM parameter, refer to "BPE USEREXIT Commands" on page 35.

BPE User-Supplied Exit Routine Callable Services

A set of callable services is provided that can be used by BPE-managed user exit routines to request certain functions from BPE. Callable services are requested by using the BPEUXCSV macro.

Recommendation: Choose the BPE service when there is a choice between using an operating system service or an equivalent BPE callable service. All callable services are Product-Sensitive Programming Interfaces (PSPIs).

BPEUXCSV Macro Description

The purpose of the BPEUXCSV macro is to issue BPE callable service requests from a user exit routine called from a BPE environment. You can use this macro

only for BPE-called exit routines (exit routines that are passed the address of a Standard BPE user exit parameter list in R1). BPE provides callable services that include the following functions:

- Get and free storage associated with the primary BPE TCB (usually jobstep). Some user exit routines can run under a different TCB each time they are called. Normally, storage obtained with GETMAIN is associated with the current TCB. If an exit routine obtained storage when it was called under one TCB and tried to free it when running under a different TCB, the storage free attempt may fail. The get storage and free storage callable services allow exit routines to get an area of storage when running under one TCB and to free it when running under a different TCB.
- Load and delete modules and associate these modules with the primary BPE TCB. Like the storage get and free services, the load and delete services handle module management when loaded and deleted from different TCBs.
- Get, retrieve, and free named storage areas. A named storage area is an area of storage that is associated with a 16-byte name. The address of the storage area can be retrieved given the name of the area. This allows different user exit routines to communicate with one another by using a common name for a shared named storage area.

When a callable service is invoked, the service may have to wait for the completion of some event. Depending on the environment at the time your user exit routine is called, such a wait can be either an OS WAIT (that is, the current TCB is suspended until the event completes) or a BPE-internal wait. For BPE-internal waits, BPE can run other ready work under the current TCB while your user exit routine is waiting for the event to complete. When the event does complete, BPE re-dispatches your exit routine's unit of work and completes the callable service request.

The possibility of waiting introduces the following situations, which your exit routine must be able to manage.

- Depending on the nature of the specific user exit routine (where and when it is called), your exit routine might be entered again for another exit routine call while the first instance of the exit routine is still waiting in a callable services request. Note that multiple concurrent calls to user exit routines are, in general, always possible. However, some user exit routines might normally be TCB-serialized (that is, their callers always run under a single TCB); these TCB-serialized routines might be entered multiple times when you use a callable service.
- Again, depending on the specific user exit routine, your exit routine might have control passed back from the BPE callable service request running under a different TCB than when it was originally called. This is because BPE provides the ability for a program that is using BPE services (such as CQS) to define a pool of TCBs. In this situation any TCB in the pool can run any unit of work that is assigned to the pool. So, your exit routine might be running under one TCB in a pool, make a callable services request, wait, and then be dispatched under a different TCB after the event completes.

BPEUXCSV Environmental Requirements

The requirements for the caller of BPEUXCSV are:

Authorization	Supervisor state, PSW key in which the user exit routine was originally called.
Dispatchable unit mode	TCB mode.

Save area	R13 must be pointing to a standard 72-byte save area.
Cross-memory mode	None (PASN=HASN=SASN).
AMODE	31-bit.
ASC mode	Primary.
Interrupt Status	Enabled.
Locks	None.

BPEUXCSV Restrictions and Limitations

BPEUXCSV can be invoked only from within a BPE-called user exit routine. BPEUXCSV is a Product-Sensitive Programming Interface.

BPEUXCSV Register Information

This macro uses R0, R1, R14, and R15 as work registers. When BPEUXCSV returns control to the caller, the contents of these registers will be changed. All other registers remain unchanged.

BPEUXCSV Performance Implications

None.

Other Macro Requirements

None.

BPEUXCSV Macro Syntax

FUNC = CALL: The FUNC = CALL function is used to invoke a callable service from a user exit routine. Figure 11 is the syntax for the CALL function.

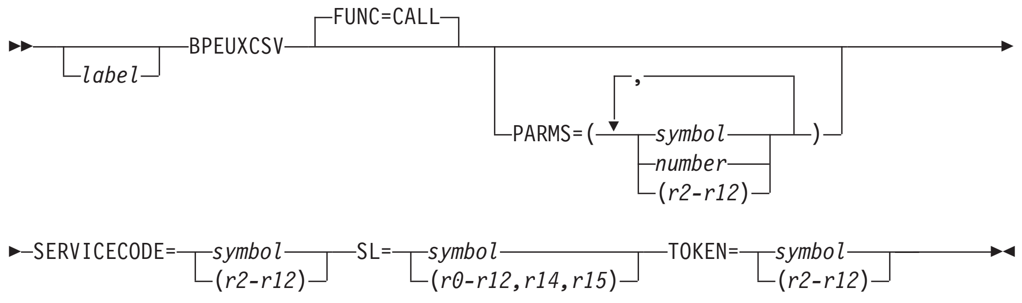


Figure 11. Syntax for BPEUXCSV Macro CALL Function

FUNC = DSECT: The FUNC = DSECT function is used to generate all of the following items:

- Return code symbols
- BPE callable service codes
- Parameter list DSECT for the BPEUXCSV CALL function

Figure 12 is the syntax for the DSECT function.

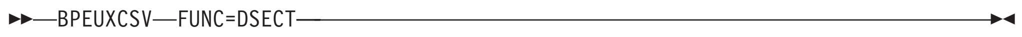


Figure 12. Syntax for BPEUXCSV Macro DSECT Function

Parameter Descriptions:

label An optional assembler label for the macro statement.

FUNC=CALL | DSECT
An optional parameter that specifies the function of the BPEUXCSV macro. The default is CALL.

CALL Invokes a BPE callable service from a user exit routine.

DSECT Generates the return code symbols, BPE callable service codes, and the parameter list DSECT for the BPEUXCSV CALL function.

PARMS=(*list_of_parameters***)**
A required parameter that specifies a list of subparameters (separated by commas) that are needed for the requested callable service. These subparameters are positional, and are specific to the service requested. Subparameters in this list may be in one of the following three forms.

symbol If coded as a symbol, the value of the symbol (for example, the result of doing an `LA R0,`*symbol*) is passed as the parameter.

number If coded as a number, the number is passed as the parameter.

(register) If coded as a register, the content of the register is passed as the parameter. Valid registers are R2 through R12.

Examples:

- If a parameter is described as "A word in storage to receive a pointer to the returned storage," you could use one of the following coding examples.

```
BPEUXCSV PARMS=(MYWORD),...

. . .

MYWORD   DS    A             Word to receive returned ptr

        - or -

LA    2,MYWORD      Get addr of word to receive ptr
BPEUXCSV PARMS=((2)),...
```

- If a parameter is described as "The number of bytes of storage to obtain," you could use one of the following coding examples.

```
BPEUXCSV PARMS=(NUMBYTES),...

. . .

NUMBYTES EQU   1024          Number of bytes to get

        - or -

BPEUXCSV PARMS=(1024),...

        - or -

LA    5,1024
BPEUXCSV PARMS=((5)),...
```

The specific parameters and parameter order for each service are described in SERVICECODE=.

SERVICECODE=*symbol* **|** **(***r2-r12***)**
A required parameter that specifies a code that identifies the particular callable service that is being requested.

If SERVICECODE is specified as a symbol, the symbol must be an EQU symbol that is equated to the function code of the requested callable service. If SERVICECODE is specified as a register, the register must contain the service code. For BPE-provided services, the appropriate EQU symbols are generated when you invoke BPEUXCSV FUNC = DSECT, and are specified as one of the following service codes.

BPEUXCSV_GETSTG	Get storage service.
BPEUXCSV_FREESTG	Free storage service.
BPEUXCSV_LOAD	Load module service.
BPEUXCSV_DELETE	Delete module service.
BPEUXCSV_NSCREATE	Create named storage service.
BPEUXCSV_NSRETRIEVE	Retrieve named storage service.
BPEUXCSV_NSDESTROY	Destroy named storage service.

SL=*symbol* **|** **(***r0-r12,r14,r15***)**
A required parameter that specifies an area in storage that is to be used as a service parameter list. The BPEUXCSV macro uses this storage to build the parameter list for the call to the callable service. The EQU symbol BPEUXCSV_MAXSL is generated by this macro and is equated to the size of the largest service parameter list required by BPE callable services. Ensure that area of storage you specify on the SL parameter is at least BPEUXCSV_MAXSL bytes in length when requesting any of the BPE callable services.

If SL is specified as a symbol, the symbol must be a label on the first byte of the area to be used as the service parameter list. If the SL parameter is specified as a register, the register must contain the address of the first byte of the area.

TOKEN=*symbol* **|** **(***r2-r12***)**
A required parameter that specifies the callable services token address that was passed to the user exit routine in the Standard BPE user exit parameter list field UXPL_CSTOKENP. If the TOKEN parameter is specified as a symbol, the symbol must be the label on a word of storage that

contains the callable services token address. If TOKEN is specified as a register, the register must contain the callable services token address.

Return from BPEUXCSV

BPEUXCSV FUNC = CALL uses general purpose registers R0, R1, R14, and R15 as work registers. On exit from the macro, R15 is set to the return code from the BPEUXCSV macro. This return code indicates the status from the callable service request router. The possible return code values in R15 are the same for all callable service requests. R0 *might* be set to a return code for the specific callable service that was requested, depending on the value that is in R15 (see R15 return codes in Table 15). The R0 return code is specific to each callable service. R1 might be set to a return value from the callable service, if applicable. See the specific service descriptions for additional information. R2 through R12 are unchanged on return from BPEUXCSV.

EQUs for the return codes in R15 are generated by BPEUXCSV FUNC = DSECT. Table 15 describes the possible return code values in R15 for FUNC = CALL, including the symbol, its value, and a description.

Table 15. FUNC=CALL Return Codes

Symbol	Value	Description
BPEUXCSV_RC_OK	X'00'	The callable service was successful.
BPEUXCSV_RC_SERV	X'04'	The specific callable service returned a non-zero return code. The return code is in the R0. Examine R0 to determine the specific reason that the request failed. The only time that the value in R0 is valid is when R15=X'04'. Otherwise, the content of R0 is unpredictable.
BPEUXCSV_RC_INVCODE	X'08'	The service code specified on SERVICECODE is invalid.
BPEUXCSV_RC_BADTOKEN	X'0C'	The callable service token passed on TOKEN is invalid.
BPEUXCSV_RC_INT	X'F4'	An internal BPE error occurred.
BPEUXCSV_RC_VERS	X'FC'	A callable services parameter list version error was encountered. The version of the parameter list generated by this macro is not valid for your current release of BPE. This is usually the result of assembling with a version of BPEUXCSV at a different level than the BPE runtime system.

BPEUXCSV Get Storage Service

The get storage service is used to obtain virtual storage. It is similar to the z/OS GETMAIN and STORAGE services; however, the storage obtained by the get storage service is always associated with the top-level BPE TCB (usually the jobstep TCB of the address space). The storage remains allocated until it is explicitly freed or until the jobstep TCB terminates. Therefore, you can rely on the fact that the storage stays allocated even if it is obtained under a subtask TCB which later terminates.

Service Code: BPEUXCSV_GETSTG

PARMS format:

PARMS=(*length,sp,opts***)** or **PARMS=(***length,sp,opts,key***)**

The following are descriptions of the parameters.

length	The length of the requested storage, in bytes.
sp	The subpool of the requested storage. This must be a valid private subpool. It cannot be a common storage subpool (such as subpool 231 or 241).
opts	Options for the storage request. *opts* is a value that is the sum of several EQU values. *opts* identifies the options you have requested for the get storage service request. A BPEUXCSV FUNC = DSECT statement must be included in your module to generate the EQUs required for this function. To specify that none of the options apply, code a zero (0) for *opts*.

BPEUXCSV_GETSTG_BELOW
Include this EQU if you want LOC = BELOW (below the line) storage. If this EQU is omitted, the storage is LOC = ANY.

BPEUXCSV_GETSTG_CLEAR
Include this EQU if you want the storage to be cleared when it is returned to you. If this EQU is omitted, the storage content is unpredictable.

BPEUXCSV_GETSTG_PAGE
Include this EQU if you want the starting address of the obtained storage to be aligned on a page boundary. If this EQU is omitted, the storage is aligned on a double-word boundary.

key The storage key of the restricted storage. *key* is an optional parameter. If coded, it indicates the storage key to be assigned to the storage returned from the get storage service. If *key* is omitted, the returned storage will be key 7 storage.

The value passed for the *key* parameter must be sixteen times the actual key value. For example, if you wanted to get key 2 storage, you would specify a value of X'20' for the *key* parameter.

Note: The *key* parameter applies *only* to subpools where **KEY=** applies on the z/OS GETMAIN macro (for example, subpool 229). It is ignored for all other subpools. You cannot, for example, request subpool 0 storage in a key other than 7.

Output: Return code EQUs are generated by BPEUXCSV FUNC = DSECT. If R15 = 0, the address of the obtained storage area is returned in R1. Otherwise, the content of R1 is unpredictable.

If R15 = 4 on return from this macro, R0 contains the reason code; Table 16 on page 73 lists these return codes, including the symbol, its value, and a description.

Table 16. Get Storage Service Return Codes

Symbol	Value	Description
BPEUXCSV_GETSTG_RCSP	X'04'	An invalid or unsupported subpool was specified. Either the subpool is not supported by z/OS, or it is a common subpool.
BPEUXCSV_GETSTG_RCLV	X'08'	A zero or negative storage length was specified. A zero storage address was specified.
BPEUXCSV_GETSTG_RCSTG	X'0C'	The storage was unable to free the requested storage.
BPEUXCSV_GETSTG_RCPARM	X'F0'	An invalid number of parameters was passed to the callable services request.
BPEUXCSV_GETSTG_RCINT	X'F4'	An internal BPE error occurred.

Examples:

- This next example shows how to get 64 bytes of storage from subpool 0. The storage is LOC = BELOW, it is aligned on a page boundary, and it is not cleared.

```
BPEUXCSV SERVICECODE=BPEUXCSV_GETSTG,                         X
      PARMS=(64,0,BPEUXCSV_GETSTG_BELOW+BPEUXCSV_GETSTG_PAGE),X
      TOKEN=UXPL_CSTOKENP,                                    X
      SL=(1)
```

- The following example shows how to get key zero storage for a length of the value in R2, from the subpool value in R3. The storage is LOC = ANY, it is not cleared, and it is double-word aligned. R4 contains the callable services token address that was passed to the user exit routine in the field UXPL_CSTOKENP.

```
BPEUXCSV SERVICECODE=BPEUXCSV_GETSTG,                        X
       PARMS=((2),(3),0, 0),                                 X
       TOKEN=(4),                                            X
       SL=WORKAREA
```

BPEUXCSV Free Storage Service

The free storage service is used to release storage that was previously obtained with the get storage service. It is similar to the z/OS FREEMAIN service. The free storage service must be used only to release storage obtained with the get storage service. It should not be used to release storage that was obtained using any other method (such as GETMAIN).

Service Code: BPEUXCSV_FREESTG

PARMS format:

PARMS=(*address,length,sp***)** or **PARMS=(***address,length,sp,key***)**

address
The address of the first byte of storage being released.

length
The number of bytes of the storage being released.

sp The subpool of the storage being released. This subpool must be the same as the subpool that was specified when the storage was obtained.

key
The storage key of the storage being released. *key* is the optional parameter. If coded, it indicates the storage key of the storage being freed. If *key* is omitted, the storage must be key 7 storage.

The value passed for the *key* parameter must be sixteen times the actual key value. For example, if you were freeing key 2 storage, you would specify a value of X'20' for the *key* parameter.

Note: The *key* parameter *only* applies to subpools where KEY= applies on the z/OS FREEMAIN macro (for example, subpool 229). It is ignored for all other subpools.

Output: Return code EQUs are generated by BPEUXCSV FUNC = DSECT. If R15 = 4 on return from this macro, R0 contains the reason code; Table 17 lists the reason codes, including the symbol, its value, and a description.

Table 17. Free Storage Service Return Codes

Symbol	Value	Description
BPEUXCSV_FREESTG_RCSP	X'04'	An invalid or unsupported subpool was specified. Either the subpool is not supported by z/OS, or it is a common subpool.
BPEUXCSV_FREESTG_RCLV	X'08'	A zero or negative storage length was specified.
BPEUXCSV_FREESTG_RCADDR	X'0C'	A zero storage address was specified.
BPEUXCSV_FREESTG_RCSTG	X'10'	The service was unable to free the requested storage.
BPEUXCSV_FREESTG_RCPARM	X'F0'	An invalid number of parameters was passed to the callable services request.

Example:

This example shows how to free STGLEN bytes starting at the byte at label MYSTG in subpool 129. STGLEN is an EQU for the number of bytes to free, and MYSTG is the label on the first byte of the area to free (*not* the label on a word pointing to the area).

```
BPEUXCSV SERVICECODE=BPEUXCSV_FREESTG,                 X
      PARMS=(MYSTG,STGLEN,129),                        X
      TOKEN=UXPL_CSTOKENP,                             X
      SL=(1)
```

BPEUXCSV Load Module Service

The Load Module Service is used to load a module from a library into storage. It is similar to the z/OS LOAD service; however, the module that is loaded is always associated with the top level BPE-TCB (usually the jobstep TCB of the address space). The module remains allocated until it is explicitly freed or until the jobstep

TCB terminates. Therefore, you can rely on the module remaining allocated, even if it is obtained under a subtask TCB that later terminates.

Service Code: BPEUXCSV_LOAD

PARMS format:
PARMS=(*modname,dcb,opts*)

modname
Identifies an eight-character field in storage containing the name of the module to be loaded. If *modname* is coded as a symbol, the symbol must be the label on the first byte of the eight-character field. If *modname* is coded as a register, the register must contain the address of the eight-character field.

dcb
The address of an opened DCB for a partitioned data set from which to load the specified module. To use the TASKLIB, STEPLIB, or JOBLIB data sets, code 0 for this parameter.

opts
Options for the load request. *opts* is a value that is the sum of several EQU values. *opts* identifies the options you have requested for the Load Module Service request. A BPEUXCSV FUNC = DSECT statement must be included in your module to generate the EQUs required for this function. To specify that none of the options apply, code 0 for *opts*.

BPEUXCSV_LOAD_FIXED	Include this EQU if you want the module to be loaded into page-fixed storage. If this EQU is omitted, the module is loaded into pageable storage. This parameter applies only if you also specify BPEUXCSV_LOAD_GLOBAL. Otherwise, BPEUXCSV_LOAD_FIXED is ignored.
BPEUXCSV_LOAD_GLOBAL	Include this EQU if you want the module to be loaded into global (common) storage. If this EQU is omitted, it is loaded into private storage.
BPEUXCSV_LOAD_EOM	Include this EQU if you specified BPEUXCSV_LOAD_GLOBAL and you want the module to be deleted only after the address space terminates. If this EQU is omitted, the module is deleted when the top-level BPE TCB terminates. BPEUXCSV_LOAD_EOM is ignored if you did not code BPEUXCSV_LOAD_GLOBAL.

Output: If R15 = 0, the address of the loaded module is returned in R1. Otherwise, the content of R1 is unpredictable.

Return code EQUs are generated by BPEUXCSV FUNC = DSECT. If R15 = 4 on return from this macro, then R0 contains the reason code; Table 18 lists the reason codes, including the symbol, its value, and a description.

Table 18. Load Module Service Return Codes

Symbol	Value	Description
BPEUXCSV_LOAD_RCNOTFND	X'04'	The specified module could not be found.

Table 18. Load Module Service Return Codes (continued)

Symbol	Value	Description
BPEUXCSV_LOAD_RCBLDL	X'08'	BLDL for the module failed due to an internal error.
BPEUXCSV_LOAD_RCLOAD	X'0C'	LOAD for the module failed. The module was found in the library, but LOAD returned a non-zero code.
BPEUXCSV_LOAD_RCPARM	X'F0'	An invalid number of parameters was passed to callable services request.
BPEUXCSV_LOAD_RCINT	X'F4'	An internal BPE error occurred.

Examples:

The following example shows how to load the module whose name is at the 8 bytes of storage, beginning at label MODNAME, from the default TASKLIB, JOBLIB, or STEPLIB data sets.

```
         BPEUXCSV SERVICECODE=BPEUXCSV_LOAD,                          X
               PARMS=(MODNAME,0,0),                                   X
               TOKEN=UXPL_CSTOKENP,                                   X
               SL=(1)

         . . .

MODNAME  DC    CL8'MODULE00'        Name of module to load
```

This next example shows how to load the module, whose name is at the 8 bytes of storage pointed to by R8, into global storage. The module is not deleted until the address space terminates (or until it is explicitly deleted). R2 contains the callable services token address that was passed to the user exit routine in the UXPL_CSTOKENP field. The module is loaded from the dataset described by DCB MYDCB.

```
         LA    8,MODNAME           R8 = addr of name of module to load
         BPEUXCSV SERVICECODE=BPEUXCSV_LOAD,                          X
               PARMS=((8),MYDCB,BPEUXCSV_LOAD_GLOBAL+BPEUXCSV_LOAD_EOM)X
               TOKEN=(2),                                             X
               SL=PRMLIST

         . . .

MODNAME  DC    CL8'MODULE00'        Name of module to load
MYDCB    DCB   DSNAME=...
```

BPEUXCSV Delete Module Service

The delete module service is used to delete a module that was previously loaded with the load module service. It is similar to the z/OS DELETE service. The delete module service must be used only to delete modules obtained with the load module service. It must not be used to delete modules that were loaded using any other method (such as z/OS LOAD).

Service Code: BPEUXCSV_DELETE

PARMS format:
PARMS=(*modname*)

modname
Identifies an eight character field in storage containing the name of the module to be deleted. If *modname* is coded as a symbol, the symbol must be the label on the first byte of the eight character field. If *modname* is coded as a register, the register must contain the address of the eight character field.

Output: Return code EQUs are generated by BPEUXCSV FUNC = DSECT. If R15 = 4 on return from this macro, then R0 the reason code; Table 19 lists these reason codes, including the symbol, its value, and a description.

Table 19. Delete Module Service Return Codes

Symbol	Value	Description
BPEUXCSV_DELETE_RCDELETE	X'04'	The module that was specified could not be deleted.
BPEUXCSV_DELETE_RCPARM	X'F0'	An invalid number of parameters was passed to the callable services request.
BPEUXCSV_DELETE_RCINT	X'F4'	An internal BPE error occurred.

Example:

The following example shows how to delete the module whose eight character name is in the storage pointed to by R5.

```
         LA    5,MODNAME           R5=addr of name of module to delete
         BPEUXCSV SERVICECODE=BPEUXCSV_DELETE,                        X
               PARMS=((5)),                                           X
               TOKEN=UXPL_CSTOKENP,                                   X
               SL=(1)

         . . .

MODNAME  DC    CL8'MODULE00'        Name of module to delete
```

BPEUXCSV Create Named Storage Service

The create named storage service allows you to obtain an area of storage that is associated with a 16-byte name. In subsequent user exit routine calls (either for the same or different exit routine types), you can retrieve the named storage area address by providing the same name to the retrieve named storage service. Named storage services allow a set of user exit routines to share information but only if they agree on the same name. Typically, an initialization-type exit routine creates the named storage, and all subsequent exit routines retrieve the named storage address.

The name of the storage must be unique within the BPE address space. The named storage is obtained in subpool 0, LOC = ANY storage. The storage is cleared to zeros when it is created.

Service Code: BPEUXCSV_NSCREATE

PARMS format:
PARMS=(*name,length*)

name
Identifies a 16-byte field in storage containing the name to be associated with the storage obtained. The field can contain any 16-byte value (all bytes are significant). If *name* is coded as a symbol, the symbol must be the label on the first byte of the 16-byte field. If *name* is coded as a register, the register must contain the address of the 16-byte field.

length
The number of bytes of the named storage area to obtain.

Output: If R15 = 0, the address of the named storage area obtained is returned in R1. Otherwise, the content of R1 is unpredictable.

Return code EQUs are generated by BPEUXCSV FUNC = DSECT. If R15 = 4 on return from this macro, R0 contains the reason code; Table 20 lists these reason codes, including the symbol, its value, and a description.

Table 20. Create Named Storage Service Return Codes

Symbol	Value	Description
BPEUXCSV_NSCREATE_RCDUP	X'04'	The requested storage area name is already in use.
BPEUXCSV_NSCREATE_RCLV	X'08'	A zero or negative storage length was requested.
BPEUXCSV_NSCREATE_RCNAME	X'0C'	A zero name address was specified.
BPEUXCSV_NSCREATE_RCSTG	X'10'	The service was unable to obtain the requested storage.
BPEUXCSV_NSCREATE_RCPARM	X'F0'	An invalid number of parameters was passed to the callable services request.
BPEUXCSV_NSCREATE_RCINT	X'F4'	An internal BPE error occurred.

Example:

This example shows how to create a 1024-byte storage area that is associated with the 16-byte name in storage. The first byte of the named storage area is at label MYNAME.

```
         BPEUXCSV SERVICECODE=BPEUXCSV_NSCREATE,                      X
               PARMS=(MYNAME,1024),                                   X
               TOKEN=UXPL_CSTOKENP,                                   X
               SL=(1)

         . . .

MYNAME   DC    CL16'SHARED_STOR_1024'  "Name" of named storage
```

BPEUXCSV Retrieve Named Storage Service

The retrieve named storage service allows you to retrieve the address of a named area of storage that was previously created with the create named storage service.

Service Code: BPEUXCSV_NSRETRIEVE

PARMS format:
PARMS=(*name*)

name
: Identifies a 16-byte field in storage containing the name of the named storage area. The field can contain any 16-byte value (all bytes are significant). If *name* is coded as a symbol, the symbol must be the label on the first byte of the 16-byte field. If *name* is coded as a register, the register must contain the address of the 16-byte field.

Output: If R15 = 0, the address of the named storage area retrieved is returned in R1. Otherwise, the content of R1 is unpredictable.

Return code EQUs are generated by BPEUXCSV FUNC = DSECT. If R15 = 4 on return from this macro, R0 contains the reason code; Table 21 lists these reason codes, including the symbol, its value, and a description.

Table 21. Retrieve Named Storage Service Return Codes

Symbol	Value	Description
BPEUXCSV_NSRETRIEVE_RCNONE	X'04'	No named storage area is associated with the specified name.
BPEUXCSV_NSRETRIEVE_RCNAME	X'08'	A zero name address was specified.
BPEUXCSV_NSRETRIEVE_RCPARM	X'F0'	An invalid number of parameters was passed to the callable services request.
BPEUXCSV_NSRETRIEVE_RCINT	X'F4'	An internal BPE error occurred.

Example:

This example shows how to retrieve the address of the named storage area associated with the 16-byte name in storage at the address contained in R6.

```
         LA   6,MYNAME
         BPEUXCSV SERVICECODE=BPEUXCSV_NSRETRIEVE,                    X
               PARMS=((6)),                                           X
               TOKEN=UXPL_CSTOKENP,                                   X
               SL=(1)

         . . .

 MYNAME   DC    CL16'SHARED_STOR_1024'  "Name" of named storage
```

BPEUXCSV Destroy Named Storage Service

The destroy named storage service is used to delete a previously created named storage area. No other user exit routine should access this storage after you destroy it.

Service Code: BPEUXCSV_NSDESTROY

PARMS format:
PARMS=(*name*)

name
: Identifies a 16-byte field in storage containing the name of the named storage

area. The field can contain any 16-byte value (all bytes are significant). If *name* is coded as a symbol, the symbol must be the label on the first byte of the 16-byte field. If *name* is coded as a register, the register must contain the address of the 16-byte field.

Output: Return code EQUs are generated by BPEUXCSV FUNC = DSECT. If R15 = 4 on return from this macro, R0 contains the reason code; Table 22 lists these reason codes, including the symbol, its value, and a description.

Table 22. Destroy Named Storage Service Return Codes

Symbol	Value	Description
BPEUXCSV_NSDESTROY_RCNONE	X'04'	No named storage area is associated with the specified name.
BPEUXCSV_NSDESTROY_RCNAME	X'08'	A zero name address was specified.
BPEUXCSV_NSDESTROY_RCPARM	X'F0'	An invalid number of parameters was passed to the callable services request.
BPEUXCSV_NSDESTROY_RCINT	X'F4'	An internal BPE error occurred.

Example:

The following example shows how to destroy the named storage area associated with the 16-byte name in storage whose first byte is at label NSNAME.

```
         BPEUXCSV SERVICECODE=BPEUXCSV_NSDESTROY,                    X
               PARMS=(NSNAME),                                        X
               TOKEN=UXPL_CSTOKENP,                                   X
               SL=(1)

         . . .

NSNAME   DC    XL16'01C1C2C300000000F1F2F3F4006D2748'  Binary names OK
```

BPE Callable Service Example: Sharing Data Among Exit Routines

As an example of the use of callable services, consider the case where you have a set of user exit routines of varying types that all need to share some common information. For this example, assume that the following three types of exit routines are being used:

- An *initialization exit routine* that gets control when the address space is first started. Assume that this exit routine runs before any mainline processing is done (so you can be sure that the other two exit routines won't be called until the initialization exit routine has returned).
- A *processing exit routine* that gets control whenever a particular event occurs in the address space that needs user exit routine provided information.
- A *termination exit routine* that gets control when the address space is ending.

Important: These particular user exit routines are presented here for example purposes only. These examples should not be assumed to be usable exit routines.

Sample Initialization Exit Routine

The initialization exit routine uses the create named storage service to obtain a 16-byte area of storage with the name ZZZ_EXIT_AREA. The storage is mapped by the following DSECT (which is assumed to be available in all of the modules).

```
ZZZ_EXIT_AREA    DSECT ,
ZZZ_TABLE_NAME   DS    CL8         Name of table module
ZZZ_TABLE_ADDR   DS    A           Address of table module
                 DS    F           Available
ZZZ_EXIT_AREA_L  EQU   *-ZZZ_EXIT_AREA
```

The initialization exit routine then uses the load module service to load a module named ZZZUXTB0 (a table that is needed in this example to pass information to the other user exit routines). The initialization exit routine stores the name of the table module in the named storage area field ZZZ_TABLE_NAME, and the address of the loaded table in field ZZZ_TABLE_ADDR. A routine using a table may not be required for your application.

A sample initialization exit routine that performs these functions is shown in Figure 13 on page 82. Note that the code shown in Figure 13 on page 82, Figure 14 on page 83, and Figure 15 on page 84 is mainline path only. To keep the examples simple, error paths and exception handling code are not shown.

```
INITEXIT CSECT ,
INITEXIT AMODE 31
INITEXIT RMODE ANY
         STM   14,12,12(13)           Save caller's registers
         LR    12,15                  Move module entry pt to R12
         USING INITEXIT,12            Address module base register
         L     13,8(,13)              Chain to 2nd provided save area
         LR    11,1                   Move exit parmlist to R11
         USING BPEUXPL,11             Address std BPE user exit PL
         L     10,UXPL_DYNAMICWAP     Get 512-byte dynamic storage ptr
         USING DYNSTG,10              Address module's dynamic storage
         BPEUXCSV SERVICECODE=BPEUXCSV_NSCREATE,  Create named stg     X
               PARMS=(NSNAME,ZZZ_EXIT_AREA_L),     for the exits       X
               TOKEN=UXPL_CSTOKENP,                                    X
               SL=UXCSVPL
         LTR   15,15                  Did NSCreate work?
         BNZ   ERROR1                 No, go handle error

         LR    9,1                    Yes, named storage ptr to R9
         USING ZZZ_EXIT_AREA,9        Address using "ZZZ" DSECT
         MVC   ZZZ_TABLE_NAME,TBLNAME  Set name of table module
         BPEUXCSV SERVICECODE=BPEUXCSV_LOAD,      Load the table       X
               PARMS=(TBLNAME,0,0),                module for the      X
               TOKEN=UXPL_CSTOKENP,                exits               X
               SL=UXCSVPL
         LTR   15,15                  Did LOAD work?
         BNZ   ERROR2                 No, go handle error
         ST    1,ZZZ_TABLE_ADDR       Yes, save table ptr in named stg

         . . .                        Do any other init exit functions

         XR    15,15                  Set zero return code
         L     13,4(,13)              Back up to caller's save area
         L     14,12(,13)             Restore caller's R14
         LM    0,12,20(13)            Restore caller's R0-R12
         BR    14                     Return to caller
         DROP  9,10,11,12             Release USING registers

NSNAME   DC    CL16'ZZZ_EXIT_AREA   '  Const for named storage
TBLNAME  DC    CL8'ZZZUXTB0'          Const for table module name
         LTORG ,

DYNSTG   DSECT ,                      Dynamic storage DSECT
UXCSVPL  DS    XL(BPEUXCSV_MAXSL)     Space for BPEUXCSV parmlist
         . . .                        Other dynamic storage fields
         BPEUXPL  FUNC=DSECT          Include user exit parmlist
         BPEUXCSV FUNC=DSECT          Include BPEUXCSV symbols
         END
```

Figure 13. Sample Initialization Exit Routine

Sample Processing Exit Routine

The processing exit routine obtains the address of the table module that was loaded by the initialization exit routine. For optimum performance, the processing exit routine uses the first word of the static work area that BPE passes to save the address of the shared storage area.

On entry, the processing exit routine checks this word of storage. If this word is non-zero, the processing routine uses this address as the shared storage area pointer. If the first word is zero, the processing exit routine invokes the named storage retrieve service to get the address of the shared storage. The processing exit routine then saves the address in the static storage area. This technique minimizes the number of BPE requests for callable services that this exit routine must make (because it needs to do the retrieve only once; on subsequent calls, the address of the shared storage area is available in the static work area).

A sample processing exit routine that performs these functions is shown in Figure 14.

```
PROCEXIT CSECT ,
PROCEXIT AMODE 31
PROCEXIT RMODE ANY
         STM   14,12,12(13)           Save caller's registers
         LR    12,15                  Move module entry pt to R12
         USING PROCEXIT,12            Address module base register
         L     13,8(,13)              Chain to 2nd provided save area
         LR    11,1                   Move exit parmlist to R11
         USING BPEUXPL,11             Address std BPE user exit PL
         L     10,UXPL_DYNAMICWAP     Get 512-byte dynamic storage ptr
         USING DYNSTG,10              Address module's dynamic storage
         L     9,UXPL_STATICWAP       Get 256-byte static storage ptr
         ICM   8,15,0(9)              Is shared stg ptr set?
         BNZ   GOTSHRD                Yes, continue
         BPEUXCSV SERVICECODE=BPEUXCSV_NSRETRIEVE, Get named stg addr  X
               PARMS=(NSNAME),                                         X
               TOKEN=UXPL_CSTOKENP,                                    X
               SL=UXCSVPL
         LTR   15,15                  Did NSRetrieve work?
         BNZ   ERROR1                 No, go handle error
         LR    8,1                    Yes, set shrd stg ptr in R8
         ST    8,0(,9)                Save in static stg for next time
GOTSHRD  DS    0H
         USING ZZZ_EXIT_AREA,8        Address using "ZZZ" DSECT
         L     7,ZZZ_TABLE_ADDR       Get table address

         . . .                        Do process exit functions
         XR    15,15                  Set zero return code
         L     13,4(,13)              Back up to caller's save area
         L     14,12(,13)             Restore caller's R14
         LM    0,12,20(13)            Restore caller's R0-R12
         BR    14                     Return to caller

         DROP  8,10,11,12             Release USING registers
NSNAME   DC    CL16'ZZZ_EXIT_AREA   ' Const for named storage

         LTORG ,

DYNSTG   DSECT ,                      Dynamic storage DSECT
UXCSVPL  DS    XL(BPEUXCSV_MAXSL)       Space for BPEUXCSV parmlist
         . . .                          Other dynamic storage fields
         BPEUXPL  FUNC=DSECT          Include user exit parmlist
         BPEUXCSV FUNC=DSECT          Include BPEUXCSV symbols
         END
```

Figure 14. Sample Processing Exit Routine

Sample Termination Exit Routine

The termination exit routine locates the shared storage area, deletes the loaded table module using the name that was saved in the shared storage area, and then destroys the shared area.

A sample termination exit routine that performs these functions is shown in Figure 15 on page 84.

```
TERMEXIT CSECT ,
TERMEXIT AMODE 31
TERMEXIT RMODE ANY
         STM   14,12,12(13)           Save caller's registers
         LR    12,15                  Move module entry pt to R12
         USING TERMEXIT,12            Address module base register
         L     13,8(,13)              Chain to 2nd provided save area
         LR    11,1                   Move exit parmlist to R11
         USING BPEUXPL,11             Address std BPE user exit PL
         L     10,UXPL_DYNAMICWAP     Get 512-byte dynamic storage ptr
         USING DYNSTG,10              Address module's dynamic storage
         BPEUXCSV SERVICECODE=BPEUXCSV_NSRETRIEVE, Get named stg addr  X
               PARMS=(NSNAME),                                         X
               TOKEN=UXPL_CSTOKENP,                                    X
               SL=UXCSVPL
         LTR   15,15                  Did NSRetrieve work?
         BNZ   ERROR1                 No, go handle error
         LR    8,1                    Yes, set shrd stg ptr in R8
         USING ZZZ_EXIT_AREA,8        Address using "ZZZ" DSECT
         BPEUXCSV SERVICECODE=BPEUXCSV_DELETE,     Delete table        X
               PARMS=(ZZZ_TABLE_NAME),              module             X
               TOKEN=UXPL_CSTOKENP,                                    X
               SL=UXCSVPL
         LTR   15,15                  Did DELETE work?
         BNZ   ERROR2                 No, go handle error
         BPEUXCSV SERVICECODE=BPEUXCSV_NSDESTROY,  Destroy named stg   X
               PARMS=(NSNAME),                                         X
               TOKEN=UXPL_CSTOKENP,                                    X
               SL=UXCSVPL
         DROP  8                      R8 no longer is "ZZZ" area
         LTR   15,15                  Did NSDestroy work?
         BNZ   ERROR3                 No, go handle error
         . . .                        Do other term exit functions

         XR    15,15                  Set zero return code
         L     13,4(,13)              Back up to caller's save area
         L     14,12(,13)             Restore caller's R14
         LM    0,12,20(13)            Restore caller's R0-R12
         BR    14                     Return to caller
         DROP  10,11,12               Release USING registers

NSNAME   DC    CL16'ZZZ_EXIT_AREA   '  Const for named storage

         LTORG ,
DYNSTG   DSECT ,                      Dynamic storage DSECT
UXCSVPL  DS    XL(BPEUXCSV_MAXSL)       Space for BPEUXCSV parmlist
         . . .                          Other dynamic storage fields

         BPEUXPL  FUNC=DSECT          Include user exit parmlist
         BPEUXCSV FUNC=DSECT          Include BPEUXCSV symbols
         END
```

Figure 15. Sample Termination Exit Routine

Notices

Programming Interface Information

This publication is intended to help the customer use the IMS Base Primitive Environment (BPE) external interfaces. The *IMS Version 9: Base Primitive Environment Guide and Reference* primarily documents product-sensitive programming interface and associated guidance information provided by IMS.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of IMS. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

General-use programming interfaces allow the customer to write programs that obtain the services of IMS.

General-use programming interface and associated guidance information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking: General-use programming interface and associated guidance information....

Diagnosis, modification or tuning information is provided to help the customer diagnose, modify, or tune IMS.

Attention: Do not use this diagnosis, modification or tuning information as a programming interface.

Diagnosis, Modification or Tuning Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking: Diagnosis, Modification or Tuning Information....

Trademarks

Bibliography

This bibliography lists all of the information in the IMS Version 9 library.

- *External Security Interface (RACROUTE) Macro Reference for MVS™*, GC28-1366
- *MVS/DFP Access Method Services for VSAM Catalog*, SC26-4570
- *MVS/ESA Initialization and Tuning Reference*, SC28-1452
- *MVS/ESA Programming: Authorized Assembler Services Guide*, GC28-1467
- *MVS/ESA Setting Up a Sysplex*, GC28-1449
- *MVS/ESA Programming: Sysplex Services Guide*, GC28-1495
- *MVS/ESA System Commands*, GC28-1442
- *MVS/ESA Programming: Assembler Services Guide*, GC28-1466
- *OS/390 MVS Setting Up a Sysplex*, GC28-1779

IMS Version 9 Library

ZES1-2330	ADB	*IMS Version 9: Administration Guide: Database Manager*
ZES1-2331	AS	*IMS Version 9: Administration Guide: System*
ZES1-2332	ATM	*IMS Version 9: Administration Guide: Transaction Manager*
ZES1-2333	APDB	*IMS Version 9: Application Programming: Database Manager*
ZES1-2334	APDG	*IMS Version 9: Application Programming: Design Guide*
ZES1-2335	APCICS	*IMS Version 9: Application Programming: EXEC DLI Commands for CICS and IMS*
ZES1-2336	APTM	*IMS Version 9: Application Programming: Transaction Manager*
ZES1-2337	BPE	*IMS Version 9: Base Primitive Environment Guide and Reference*
ZES1-2338	CR	*IMS Version 9: Command Reference*
ZES1-2339	CQS	*IMS Version 9: Common Queue Server Guide and Reference*
ZES1-2340	CSL	*IMS Version 9: Common Service Layer Guide and Reference*
ZES1-2341	CG	*IMS Version 9: Customization Guide*
ZES1-2342	DBRC	*IMS Version 9: DBRC Guide and Reference*
ZES1-2343	DGR	*IMS Version 9: Diagnosis Guide and Reference*
ZES1-2344	FAST	*IMS Version 9: Failure Analysis Structure Tables (FAST) for Dump Analysis*
ZES1-2346	OLR	*IMS Version 9: HALDB Online Reorganization Guide*
ZES1–2380	CT	*IMS Version 9: IMS Connect Guide and Reference*
ZES1-2347	JGR	*IMS Version 9: IMS Java Guide and Reference*
ZES1-2348	IIV	*IMS Version 9: Installation Volume 1: Installation Verification*
ZES1-2349	ISDT	*IMS Version 9: Installation Volume 2: System Definition and Tailoring*
ZES1-2350	INTRO	*IMS Version 9: An Introduction to IMS*
ZES1-2351	MIG	*IMS Version 9: Master Index and Glossary*
ZES1-2352	MC1	*IMS Version 9: Messages and Codes, Volume 1*
ZES1-2353	MC2	*IMS Version 9: Messages and Codes, Volume 2*
ZES1-2354	OTMA	*IMS Version 9: Open Transaction Manager Access Guide and Reference*
ZES1-2355	OG	*IMS Version 9: Operations Guide*
GC17-7831	RPG	*IMS Version 9: Release Planning Guide*
ZES1-2358	URDBTM	*IMS Version 9: Utilities Reference: Database and Transaction Manager*
ZES1-2359	URS	*IMS Version 9: Utilities Reference: System*

Supplementary Publications

GC17-7825	LPS	*IMS Version 9: Licensed Program Specifications*
ZES1-2357	SOC	*IMS Version 9: Summary of Operator Commands*

Publication Collections

LK3T-7213	CD	IMS Version 9 Softcopy Library
LK3T-7144	CD	IMS Favorites
LBOF-7789	Hardcopy and CD	Licensed Bill of Forms (LBOF): IMS Version 9 Hardcopy and Softcopy Library

	Publication Collections	
SBOF-7790	Hardcopy	Unlicensed Bill of Forms (SBOF): IMS Version 9 Unlicensed Hardcopy Library
SK2T-6700	CD	OS/390 Collection
SK3T-4270	CD	z/OS Software Products Collection
SK3T-4271	DVD	z/OS and Software Products DVD Collection

	Accessibility Titles Cited in this Book	
SA22-7787		z/OS V1R1.0 TSO Primer
SA22-7794		z/OS V1R1.0 TSO/E User's Guide
SC34-4822		z/OS V1R1.0 ISPF User's Guide, Volume 1

Index

Program Number: 5655-J38

IBM Confidential
Printed in USA

ZES1-2337-01

Spine information:

IBM IMS Base Primitive Environment Guide and Reference Version 9