

z/OS



JES Application Programming

Version 2 Release 2

Note

Before using this information and the product it supports, read the information in "Notices" on page 73.

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

© **Copyright IBM Corporation 2008, 2015.**

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

Contents

Figures	v
-------------------	---

Tables	vii
------------------	-----

About this document	ix
Who should use this document	ix
How to use this document	ix
Where to Find More Information	ix

How to send your comments to IBM	xi
If you have a technical problem.	xi

Summary of changes for z/OS Version 2 Release 2 (V2R2)	xiii
Summary of changes for z/OS Version 2 Release 1	xiii

Chapter 1. Introduction	1
--	----------

Chapter 2. JES Spool Data Set Browse	3
---	----------

Allocation	3
Specifying the Data Set Name (DALDSNAM)	4
Building the Browse Token (DALBRTKN).	4
Security	6
Errors and Return Codes	7
Using the Compatibility Interface	8
Using the ACB/RPL Interface.	8
Building the ACB.	8
Using RPL-based macros	9
Special Processing for Logical SYSLOG Data Sets	10
Special Processing for Logical EVENTLOG Data Sets	11
Return Codes.	11
End of File Processing	13
Performance	13
Secondary Subsystem Support	14
Accessing the EVENTLOG data set for job step completion codes	14
Processing EVENTLOG records.	15
Writing to the EVENTLOG data set	16
EVENTLOG macros	17
Record Prefix (IAZLGINF) mapping macro	17
STEPDATA (IAZLGSTP) mapping macro	18
RESTART (IAZLGRST) mapping macro	19
EVENTLOG data service (IAZLGDAT) macro	21
EVENTLOG data service data definition (IAZLGDDF) macro	24

Chapter 3. JES Client/Server Print Interface	27
---	-----------

Creating a CTOKEN	27
Comparing CTOKENs.	27
Obtaining Status for a Data Set.	28
Accessing a Data Set	28

Security	29
Identifying a Requestor on a Header Page	29
Listening for Events	30

Chapter 4. JES Symbol Service (IAZSYMBL)	33
---	-----------

JES system symbols.	34
JES Symbol (IAZSYMBL) macro	35
JES Symbol Service data definition (IAZSYMDF) macro	37
Return codes (JSYMRETN)	37
Reason codes (JSYMREAS)	38
Parameter list (JSYMPARM)	38
Requested operations (JSYMRQOP)	39
Defining JES symbols	40
JES symbol options (JSYMLVL)	41
Input symbol table (JSYMISYT).	41
DELETE and EXTRACT symbols	41
Symbol table (JSYTABLE).	42
Symbol entry (JSYENTRY)	43

Chapter 5. Internal reader facility	45
--	-----------

Defining the internal reader facility	45
Using the internal reader facility	45
Submitting to the internal reader from jobs or tasks.	46
Dynamically allocating the internal reader	46
Passing JCL symbols to the submitted job	47
Requesting job notification	47
Assigning the user portion of the job correlator	47
Getting feedback	48
Time-sharing logon (TSO/E) and started task (STC) flow.	48
Using the RDR procedure	49
Examples of using the RDR procedure	49
JES control statements that affect the internal reader	50
Performance considerations for JES internal reader	50
Use of unblocked records for SYSIN and SYSOUT data sets	50
Held internal readers in JES2	50
Record length of SYSIN data sets	50
SYSIN record formats	51

Chapter 6. The External Writer	53
---	-----------

Overview of the IBM-Supplied External Writer	53
Characteristics of the IBM-Supplied External Writer	53
How to Set Up and Start the External Writer	54
The External Writer Cataloged Procedure	54
How the IBM-Supplied Output Writer Routine Works	58
Functions of the output writer routine	58
How to Add Your Own Output Writing Routine	60
Replacing the IBM-Supplied Routine	60
Using Your Routine for Only Certain Jobs	61

Appendix. Accessibility 69
Accessibility features 69
Consult assistive technologies 69
Keyboard navigation of the user interface 69
Dotted decimal syntax diagrams 69

Notices 73
Policy for unsupported hardware 74

Minimum supported hardware 75
Trademarks 75

Index 77

Figures

- | | | | |
|--|----|---|----|
| 1. Submitting a Job to the Internal Reader | 46 | 3. General flow of IBM's output writing routine | 60 |
| 2. The RDR Procedure | 49 | | |

Tables

1. Additional RBA formats	11	3. Separator Routine Parameter List	65
2. External Writer Parameter List	58	4. Block Character Routine Parameter List	67

About this document

This document supports z/OS (5650-ZOS).

This document describes the application programming of JES2 and JES3. It provides the information that you need to:

- Use the Spool Data Set Browse
- Use the Server/Client Print Interface
- Use the Internal Reader
- Use the IBM supplied External Writer

Who should use this document

This document is intended for JES2 and JES3 application programmers who are using spool data set browse, server/client print interface, internal reader, and external writer.

How to use this document

Use this document in conjunction with the following documents:

- *z/OS JES2 Initialization and Tuning Guide*
- *z/OS JES3 Initialization and Tuning Guide*
- *z/OS MVS Using the Subsystem Interface*

Most referenced publications are abbreviated throughout the text; their full titles appear in “Where to Find More Information,” which follows.

Where to Find More Information

This document references the following publications for further details about specific topics. Abbreviated forms of these titles are used throughout this document. The following table lists all abbreviated titles, full titles, and their order numbers that are not listed in the z/OS[®] Information Roadmap. See that document for all z/OS publications.

Short Title Used in This document	Title	Order Number
JES2 Initialization and Tuning Guide	<i>z/OS JES2 Initialization and Tuning Guide</i>	SA22-7532
JES3 Initialization and Tuning Guide	<i>z/OS JES3 Initialization and Tuning Guide</i>	SA22-7549
MVS Using the Subsystem Interface	<i>z/OS MVS Using the Subsystem Interface</i>	SA22-7642

How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the Contact z/OS.

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
z/OS V2R2 JES Application Programming
SA32-0987-01
- The topic and page number that is related to your comment.
- The text of your comment.

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

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

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at [IBM support portal](http://ibm.com/support).

Summary of changes for z/OS Version 2 Release 2 (V2R2)

The following changes are made in z/OS Version 2 Release 2 (V2R2).

NEW

- Added “Special Processing for Logical EVENTLOG Data Sets” on page 11.
- Added new sections:
 - “Accessing the EVENTLOG data set for job step completion codes” on page 14
 - “Processing EVENTLOG records” on page 15
 - “Writing to the EVENTLOG data set” on page 16
 - “EVENTLOG macros” on page 17
 - “Record Prefix (IAZLGINF) mapping macro” on page 17
 - “STEPDATA (IAZLGSTP) mapping macro” on page 18
 - “RESTART (IAZLGRST) mapping macro” on page 19
 - “EVENTLOG data service (IAZLGDAT) macro” on page 21
 - “EVENTLOG data service data definition (IAZLGDDF) macro” on page 24

Changed

- Modified Chapter 2, “JES Spool Data Set Browse,” on page 3.
- Modified Chapter 4, “JES Symbol Service (IAZSYMBL),” on page 33.
- Modified “JES system symbols” on page 34.
- Modified “JES symbol options (JSYMLVL)” on page 41.

Summary of changes for z/OS Version 2 Release 1

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

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

Chapter 1. Introduction

JES provides several application programs to supplement its performance. This document talks about procedures and considerations when you use the following application programs:

- Chapter 2, "JES Spool Data Set Browse," on page 3
- Chapter 3, "JES Client/Server Print Interface," on page 27
- Chapter 4, "JES Symbol Service (IAZSYMBL)," on page 33
- Chapter 5, "Internal reader facility," on page 45
- The External Writer

Chapter 2. JES Spool Data Set Browse

Spool data set browse (SDSB) is a function application program that can be invoked to process spool data sets. JES provides an interface that application programs can use for this purpose. SDSB will read active, live syslog data sets. Any full data buffers that have been written to SPOOL can be read and optionally, data that has not been written (unwritten buffer support) can also be accessed.

Note: To read unwritten buffers with JES3, each of the following systems must be at z/OS V1R11, or higher:

- The system of the application.
- The system of the global processor.
- The system where the target data set is open.

If the application is running on the same system where the target data set is open, the system must be at z/OS V1R10, or higher.

Programs can use SDSB to allocate SPOOL data sets. The data set being requested is specified by passing the JES data set name along with a SPOOL browse token to MVS™ dynamic allocation. When the data set is allocated, the data set can be read using one of two methods:

1. Using the compatibility interface (DCB, GET).
2. Using the ACB/RPL interface.

You use the compatibility interface when synchronous sequential access is required. You use the ACB/RPL interface when random access to the spool file is required, or when asynchronous processing is required.

Allocation

JES recognizes an allocation of spool data set browse when a SPOOL browse token is specified on a dynamic allocation call. The SPOOL browse token is specified along with the data set name and contains optional control information JES uses to allocate the data set you want. SDSB can only be specified on dynamic allocation requests, and the following text units are required on the dynamic allocation request:

- DALDSNAM (data set name)

Note: DALDSNAM can also be set from PDBDSNAM (JES2 only) or STVSDSN that is returned by extended status.

- DALSTATS (disposition = SHR)
- DALSSREQ or DALUASSR (subsystem name)

Note: DALUASSR is the unauthorized version of DALSSREQ. They are mutually exclusive.

- DALBRTKN (browse token)

Other text units are optional. For example, DALRTDDN, used to return the ddname allocated to the data set.

Note: The DALSSREQ text unit can only be specified by authorized programs, and the allocation must be performed in authorized state. After the allocation is complete, an unauthorized task can perform I/O operations on the spool data set.

Specifying the Data Set Name (DALDSNAM)

The JES data set name is passed to dynamic allocation using the DALDSNAM key. The format of the data set name is:

```
userid.jobname.jobid.Ddskey.dsname
```

If the exact data set name is not known, the generic characters '?' and '*' can be used in the data set name. However the jobname and jobid are required. In the event that more than one data set matches the data set name requested, then the first data set that matches is allocated.

In addition to the standard JES data set name, there are some alternate data names that can be used to allocate specific JES data sets (without knowing the exact data set name) and logical data set concatenations. One alternate format is:

```
userid.jobname.jobid.jes_dsname
```

The *jes_dsname* is one of the following values:

- EVENTLOG - Job event information, such as step completion codes
- JCL - This is the input JCL (including the SYSIN data sets) exactly as submitted
- JESJCLIN - Same as JCL
- JESJCL - JCL images as output by the converter
- JESMSGLG - JES message log (WTOs issued by the job)
- JESYSMSG - JES system messages

Note:

1. When JESMSGLG is used as the data set name, if JESLOG SPIN is specified, JES2 attempts to logically concatenate the spun off JESMSGLG data sets into a single logical data set.
2. When JESYSMSG is used as the data set name, if JESLOG SPIN is specified, JES2 attempts to logically concatenate the spun off JESYSMSG data sets into a single logical data set.
3. JES3 does not concatenate spun off JESMSGLG and JESYSMSG data sets into a single logical data set. The data sets are used as individual ones.

SPOOL Data Set Browse also supports accessing the active, live SYSLOG for a system (provided that there the system is maintaining a SYSLOG in this JESPLEX). To allocate the logical SYSLOG concatenation for a system specify the following data set name (in DALDSNAM):

```
sysname.SYSLOG.SYSTEM
```

The *sysname* is the MVS system name of the system whose SYSLOG the application will examine. JES logically concatenates all the active, live SYSLOG data sets (even if there are multiple jobs) for the specified system in chronological order.

Building the Browse Token (DALBRTKN)

Before issuing the dynamic allocation request, the application must build the browse token and pass it with the DALBRTKN text unit. The format of the browse

token is mapped by macro **IAZBTOKP**. The browse token is built in text unit format with 7 subparameters. When building the browse token, the following SVC 99 text unit fields must be set:

Field	Value
S99TUKEY	DALBRTKN
S99TUNUM	7
S99TUPAR	Mapped by IAZBTOKP

The token is of fixed length and all subparameters must be coded. Each browse token subparameter contains a length followed by the data, so it will be in text unit format.

You can complete the fields in the token as follows:

BTOKPL1

Length of the browse token identifier (LENGTH(BTOKID)).

BTOKID

Browse token id (BTOK). The IAZBTOKP macro defines constant BTOKCID to be used to set this field.

BTOKPL2

Length of the token version field (LENGTH(BTOKVER)).

BTOKVER

The 2 byte version number of the token parameter list. Byte 1 (or TOKTYPE) indicates the call type. If it is set to BTOKBRWS, then this is a normal browse request. If it is set to BTOKSTKN, then this is a SPOOL token based browse request. JES3 supports only BTOKSTKN. Byte 2 (or BTOKVERS) is the parm list version and should be set to BTOKVRNM.

BTOKPL3

Length of the data pointer field.

BTOKIOTP/BTOKSPLT

Data pointer whose content is based on the first byte of BTOKVER.

BTOKIOTP

If BTOKVER is set to BTOKBRWS (JES2 only), this is a normal browse, and BTOKIOTP is either zero or the MTTR of the IOT containing the Pddb of the file to be allocated (obtained from JOEIOTTR or IOTTRACK, for example). The data set name supplied in DALDSNAM keyed text unit is used to locate the specific data set to be allocated. If BTOKIOTP is zero, the data set is located by using only the data set name.

BTOKSPLT

If BTOKVER is set to BTOKSTKN, BTOKSPLT can be zeroes or point to a client token (returned from dynamic allocation using key DALRTCTK) or a data set token (returned by the SAPI SSI in field SSS2DSTR or the Extended Status SSI in field STVSCTKN). If BTOKIOTP is zeroes, JES will use the data set name supplied in the DALDSNAM keyed text unit to find the specific data set to allocate. If BTOKIOTP points to a client or data set token, the token is used to find the data set to allocate and the data set name supplied in the DALDSNAM keyed text unit is ignored.

BTOKPL4

Length of the job key field (LENGTH(BTOKJKEY)).

BTOKJKEY

Optional job key of the file to be allocated (for example, the job key obtained from JCTJBKEY, JQEBKEY, or SJBKEY). This field is not used if

BTOKVER is set to BTOKSTKN. This field is required if BTOKTYPE is set to BTOKBRWS and BTOKIOTP is non-zero. JES3 does not support this parameter and this field is set to zero.

BTOKPL5

Length of the ASID field (LENGTH(BTOKASID)).

BTOKASID

The 2 byte ASID of the data set owning job if active buffers are needed. If active buffers are not needed, then pass 0. If the ASID is not known, then pass X'FFFF' and JES will determine the correct ASID.

BTOKPL6

Length of the RECVR field (LENGTH(BTOKRCID)).

BTOKRCID

Eight byte userid to be used as the RECVR on the SAF call or zeros if the RECVR is not being used. JES uses this field to check authority to the browse request. When RECVR is used, the value must be left justified and padded with blanks. For JES3, this is supported for authorized callers only.

When this parameter is specified, the *logstr* field should also be used so that usage of *recvr* can be logged. However, neither JES nor SAF enforces this convention.

BTOKPL7

Length of the *logstr* field (LENGTH(BTOKLOGS)).

BTOKLSDL

Length of the *logstr* (specified in field BTOKLSDA) to be used on the SAF call used by JES to check authority to the browse request, or zero if the *logstr* is not being used.

The *logstr* length must be a value from 0 to 254.

BTOKLSDA

Text of the *logstr* if BTOKLSDL is non-zero, or zeros if the *logstr* is not being used.

The maximum length text is 254 characters.

Note: When you use the compatibility interface to read the data set, you could also use text units specifying the record format, record length, and blocksize.

Security

JES does not perform any SAF call during allocation. When the SPOOL data is opened, JES uses SAF to verify read access to a JESSPOOL resource associated with the data set. SPOOL browse uses both the standard form of the JESSPOOL class resources and modified forms for special system data sets. Any generic characters that may have been specified at allocation are replaced by the actual values for the data set allocated.

When a logical data set name was specified for DALDSNAM, then the format of the resource name passed to SAF is:

```
localnodeid.userid.jobname.jobid.jes_dsname
```

In the resource name:

localnodeid

The NJE node name of the node on which the SYSIN or SYSOUT data set currently resides. The localnodeid appears in the JES job log of every job.

userid The userid associated with the job. This is the userid RACF® used for validation when the job runs.

jobname

The name that appears in the name field of the JOB statement.

jobid The job number JES assigned to the job. The jobid appears in notification messages and the JES job log of every job.

jes_dsname

One of the following fixed names:

- EVENTLOG - Job event information, such as step completion codes
- JCL - This represents the jobs input JCL (with all SYSIN data sets)
- JESJCL - The JCL images data set as created by the conversion process
- JESMSGLOG - The JES2 job log data set
- JESYSMSG - The MVS SYSTEM messages data set

When a SYSLOG data set is allocated, the format of the resource name passed to SAF is:

localnodeid.userid.SYSLOG.SYSTEM.sysname

In the resource name:

localnodeid

The NJE node name of the node on which the SYSLOG data set resides. The localnodeid appears in the JES2 job log of every job

userid The user ID provided by the security product. If RACF is used, the user ID will be +MASTER+.

sysname

The MVS system name of the system that created the SYSLOG.

If the browse token specifies a *recvr* userid, the SAF call is performed with the RECVR parameter. When the *recvr* userid is specified, the *logstr* parameter should also be supplied.

If the data set fails the security check, the open request fails with R15=0C and an error code stored in ACBERFLG (decimal 152).

The system performs a SAF call as part of OPEN processing to ensure that the user is authorized to the data set. In JES2, if the user is not authorized, a system abend, code S913, results. In JES3, although control is returned to the application, the DCBOFOPN bit is not set and the application cannot read the data set. After the DCB has been opened, use a GET macro pointing to the DCB to read the file. When processing is complete, use a CLOSE macro to close the file. The same task that opened the DCB must be used to close it.

Errors and Return Codes

Three types of errors might occur:

- If an error occurs allocating the data set, in both JES2 and JES3, dynamic allocation returns an S99ERROR reason code of X'04F8' describing the error. You can use the DAIRFAIL service to format the error text.
- If, during allocation, JES2 detects an un-initialized data set (that is, PDBMTTR contains zeros), it fails the dynamic allocation also with an S99ERROR reason code of X'04F8'. This condition does not apply to JES3.

- If, during data set open, an error occurs:
 - JES2 sets a return code in register 15 following the OPEN and stores a reason code in the ACB.
 - JES3 does not set upon the flag ACBOPEN or DCBOFOPN returned from the OPEN macro.

Using the Compatibility Interface

After allocating the spool data set, you can use the compatibility interface to sequentially read each record. Your application program builds and opens a DCB that specifies the ddname of the allocated spool data set. The record format, record length, and block size could be specified on the allocation or in the DCB, or they could be obtained from the SYSOUT data set that was allocated.

Using the ACB/RPL Interface

Your application can use the ACB/RPL interface to obtain the most flexibility when using spool data set browse (SDSB). With this method, the application builds and opens an ACB (access method control block), and uses RPL based macros to read and position the data set. You use a subset of the ACB/RPL macros as documented for VSAM. In general, JES implements only those features required to process the data set. Other options specified on the macros are ignored. When coding the ACB and RPL macros, AM=VSAM should be specified or defaulted.

Building the ACB

After the dynamic allocation completes for the data set, your application builds and opens an ACB. You can use the GENCB service or map the ACB with IFGACB macro. The storage for the ACB must be resident below the 16 megabyte line. You then use the SHOWCB and MODCB services to display and modify selected fields of the ACB. However, some of the fields required by JES are not processed by those services, and hence they may be of limited use. The only required field for the ACB is the ddname to use. The ddname can be assigned when the spool data set is allocated, or the system can return a generated name. In either case, the ACBDDNAM must be completed before the ACB is opened.

Note: The ACB can also specify an optional exit list. The exit list is built with the EXLST macro, and can be used to specify the EODAD, SYNAD, and LERAD exits.

When the ACB is generated, your application opens it using an OPEN macro. As part of open, the system performs a SAF call to ensure that the user is authorized to the data set. In JES2, if the user is not authorized, a system abend, code S913, results. In JES3, although control is returned to the application, the ACBOPEN bit is not set and the application cannot read the data set.

Note: Unless authorization has changed between the time the data set was allocated and opened, an unauthorized user will be detected at allocation time.

After the open is successful, you use RPL based macros to read the data set. If the open is not successful, error and reason codes that describe the error are placed in the ACB.

When processing is complete, your application should issue a CLOSE macro specifying the open ACB. If a CLOSE is not performed, an automatic close occurs by the system when end-of- task occurs. The close must be done by the same task that opened the ACB.

Requesting Carriage Control

The spool data set browse (SDSB) interface can be used to obtain the carriage control character if it is contained in the record. The spool data set browse interface returns carriage control with the record only if it is requested. Your application indicates that carriage control is wanted by setting flag ACBCCTYP as follows:

- ACBCCTYP.ACBCASASA = ON indicates that ASA characters are wanted.
- ACBCCTYP.ACBCMCH = ON indicates that machine characters are wanted.

Both bits can be turned on to indicate that carriage control is required regardless of format. When a data set contains carriage control, and the application requests carriage control for that type, the control character will be returned in the first byte of the record area returned by JES.

If the data set does not contain carriage control, but the application requests carriage through the setting of ACBCCTYP, JES performs the following:

- If ACBCASASA is on, JES returns a X'40' as the carriage control.
- If ACBCMCH is on, JES returns a X'09' as the carriage control.
- If both ACBCASASA and ACBCMCH are on, JES2 returns a X'09' and JES3 returns a X'40' as the carriage control.

To obtain the exact carriage control associated with a record and an indicator of the carriage control type, an application should set the ACBCANY bit. If this bit is set, JES returns a pointer to the carriage control in RPLCCHAR and the type of carriage control in RPLOPT4.

Using RPL-based macros

All I/O requests are specified using an RPL. The RPL contains the address of an open ACB. You build the RPL using the GENCB service, or map it with the IFGRPL macro. You can then use SHOWCB and MODCB for some functions. The storage for the RPL must be resident below the 16 megabyte line.

The processing options are specified in the RPLOPTCD parameter. Only a subset of those defined by VSAM are recognized by JES. In particular, spool data set browse supports only "move mode" (OPTCD=MVE) processing. OPTCD=SYN can be specified for synchronous requests (the default), or OPTCD=ASY can be used for asynchronous processing.

Your application must obtain a buffer area where JES places the record it reads. The address of the area is placed in RPLAREA and its length in RPLBUFL. If the area is not large enough to contain the record, JES sets an error code in the RPL. The storage for the buffer may be resident above the 16 megabyte line.

Your application reads each record using a GET macro which points to the RPL. When synchronous processing is used, control returns to the application when the GET is complete and the record has been moved to the application provided area. JES sets the length of the returned record in field RPLRLEN.

When asynchronous processing is used, your application must issue a CHECK macro specifying the RPL that is to be waited on. Control then returns to your application when the function specified by the RPL is complete.

After each GET request, JES returns a token your application can use to position directly to the record if it needs to be reread. JES returns the 8 byte token in field RPLRBAR after each GET. Your application should treat RPLRBAR strictly as a token, and not depend on it to be in a specific format.

Your application can use the POINT macro to locate a previously read record. POINT specifies an RPL that contains an RPLRBAR that was returned on the GET request for the record you want. POINT positions JES to the record to be read; it does not read the record. A GET macro must be issued after the POINT to retrieve the record. Input to POINT is an 8 byte RBA value pointed to by RPLARG. This RBA value can be either a RPLRBAR value that was saved from a previous GET request, or a record number that is relative to the start of the data set (0-16,777,215). If the record that is requested is not found due to an I/O error, the result depends upon whether it is being processed by a JES2 or JES3 subsystem, as follows:

- JES2:** For JES2, if the record that is requested for POINT is not found due to an I/O error, POINT processing positions to the last record of the previous buffer that it can read. A GET request after this POINT will return records starting from the positioned record. If the record requested by GET is not found due to an I/O error, GET processing positions to a buffer that it can successfully read and continues returning the records. If records are skipped on a GET request due to I/O error and IAZDSINF is provided, the indicator DSIN1RSK(X'40') is set in DSINFLG1 to indicate the skipped records. If the I/O error is at the end of the data set (no buffers found after the I/O error), the EOF and DSIN1RSK indicators are set.
- JES3:** For JES3, if the record requested for POINT is not found, POINT processing fails, leaving the data set in an undefined state. A GET request after a failed POINT results in unpredictable data being returned. If the record requested by GET is not found due to an I/O error, the remainder of the current data set is skipped.

Each GET request returns a logical record in the file. The application is isolated from the internal format of the record. JES places the complete record in the RPLAREA buffer and its length in the RPLRLEN field. JES performs the necessary unwritten buffer support for active jobs; however, no indication of the source of the record is placed in the RPL. JES places feedback information in the RPL after each request. RPLRTNCD and RPLCND CD should be checked for satisfactory completion after each operation.

Special Processing for Logical SYSLOG Data Sets

When a SYSLOG data set is allocated by using the special data set name of sysname.SYSLOG.SYSTEM, a number of additional options are available. This is in addition to all the options available to normal SPOOL Data Set Browse processing. Additional information can be returned on a successful GET request. To obtain this information, do the following settings:

- Set RPLERMSA to the address of a data area mapped by IAZDSINF.
- Set RPLEMLEN to the length of the data area (DSINSIZ1).
- Set DSINEYE eyecatcher to the value of DSIN.

The following information can be returned:

- The source record number (relative to this data set and the beginning of the concatenation).
- The time stamp associated with the message (STCKE format).
- The source job number and data set number for the record.

In addition, POINT processing is enhanced to provide support for additional RBA formats. All new RBA formats start with a X'FF', which is followed by a 1-byte function number and a 6-byte argument. Table 1 shows the RBA values (in hex) that JES supports.

Table 1. Additional RBA formats

RBA value (in hex)	Function
FF0cccc ccccccc	Go to first occurrence of time stamp passed in <i>cccc ccccccc</i> (STCKE format)
FF01cccc ccccccc	Go to next record with a time stamp of <i>cccc ccccccc</i>
FF02cccc ccccccc	Go to previous record with a time stamp of <i>cccc ccccccc</i>
FF03rrrr rrrrrrr	Move <i>rrrr rrrrrrr</i> records (signed value) from the current record
FF04rrrr rrrrrrr	Move to absolute record <i>rrrr rrrrrrr</i> from start of the (logical) data set

Note:

1. Messages in the log might not be in chronological order. This must be considered when search for records based on a time stamp.
2. In a JES3 environment, SYSLOG data that was created prior to z/OS V1R11 is ignored when processing a POINT based on time stamp.

POINT processing that specifies a specific time (RBA values starting with X'FF00'), an absolute record number (RBA values starting with X'FF04'), or a relative record number (RBA values starting with X'FF03') will position the data set to the first or last record of a file if the argument specified if beyond the end or before the start of the data set.

Special Processing for Logical EVENTLOG Data Sets

Additional information can be returned on a successful GET request that accesses in storage buffers for an active job. To obtain this information, do the following settings:

- Set RPLERMSA to the address of a data area mapped by IAZLGSTP.
- Set RPLEMLEN to the length of the data area (STP30LEN).
- Set STPSEYE eyecatcher to the value of ASIN.

The following information can be returned:

- Step number
- Step name
- Procedure name
- Program name

Return Codes

By using the ACB/RPL interface, the requests of JES I/O return error information in a 3-byte RPL field, RPLFDBK. The list below explains the meaning of the these bytes:

Byte Meaning

RPLRTNCD

Requests return code.

RPLNOERR - X'00'

No error encountered.

RPLLOGER - X'08'

Logic error encountered.

RPLPHYER - X'0C'

Physical error encountered.

RPLCMPON

Component processing request.

X'02'

JES2 processing request.

X'03'

JES3 processing request.

RPLERRCD

Error code associated with return code. The meaning of the error code is based on the request return code:

- For request return code RPLLOGER, RPLERRCD could be one of the following values:

RPLEODER - X'04'

Normal end of data has occurred.

RPLNOREC - X'10'

A POINT request was issued but the record specified by the RBA passed in RPLARG was not found. Another POINT must be done before any further GETs.

RPLINRBA - X'20'

An RBA associated with a POINT or a GET-UPDATE request was found to be invalid (not a recognizable format).

RPLNOVRT - X'28'

The request required the scheduling of an SRB to complete but the needed virtual storage could not be obtained.

RPLINBUF - X'2C'

On a GET request, the size of the area (RPLBUFL) passed in RPLAREA was too small to contain the record being returned. The actual record size is set in RPLRLEN. Obtain a larger area and re-issue the GET request.

RPLINACC- X'44'

Access type is not allowed for this data set. For example, an attempt to PUT to a data set that was opened for input processing.

RPLINUPD - X'5C'

PUT-UPDATE occurred before GET-UPDATE.

RPLDLCER - X'64'

PUT-UPDATE length was changed.

RPLINLEN - X'6C'

A PUT was done for a record and the specified length exceeds the JES limit of 32767 bytes.

RPLNOBFR - X'98'

A POINT request was made but all available buffers are being used by outstanding locate mode GET requests.

RPLREOB - X'20'

A GET request was made but all available buffers are being used by outstanding locate mode GET requests.

- For request return code RPLPHYER, RPLERRCD could be one of the following values:

RPLRDERD - X'04'

The read request failed because either there was a physical read error or the record read did not pass validation processing.

RPLWTERD - X'10'

A write operation encountered a physical write error.

End of File Processing

For jobs not running, the spool data set browse (SDSB) interface utilizes the end-of-file exit when no more data can be read. When using the compatibility interface, the exit is defined using the EODAD parameter of the DCB. When using the ACB/RPL interface, the end-of-file exit is defined on the EXLST macro. In addition, the return codes are placed in the RPL to indicate the end of file condition.

For jobs actively running, using the ACB/RPL interface, your application attempts to read unwritten spool buffers that reside in the address space of the active job. When all unwritten buffers have been read, the ACB/RPL utilizes the end-of-file exit. Your application can issue subsequent get requests to obtain additional data. On each get request, the ACB/RPL interface returns the unwritten buffer data, if available. Thus, an end of file condition for an active job should be considered temporary rather than permanent. As the active job creates additional data, subsequent get requests can be used to retrieve it. If no more data is available at the time of the get, end of file will be driven.

Note: This processing differs from standard access methods. Normally, a get request issued after end of file is considered as a permanent error. However, this condition should be expected when using the spool data set browse interface against an active job.

Performance

Spool data set browse (SDSB) presents a record level interface (that is, a complete record is returned on each get request). If a record is internally stored in several JES spool blocks, your application's use of the interface performs the necessary spool I/O to assemble the record segments. JES maintains only one spool buffer in storage at a time. Similarly, if a POINT macro is issued for a record not contained in the last spool block, JES initiates the I/O to read the block.

Your application can improve the performance of the interface by optimizing the number of I/O requests. For example, you might want your application to buffer previously read records in storage, rather than call the interface with POINT and GET to reread a record.

Secondary Subsystem Support

The spool data set browse (SDSB) interface supports allocation of spool data sets to secondary JES2 subsystems. Specify the subsystem name in the DALSSREQ or DALUASSR text unit and direct the allocation to the proper JES2.

Note:

1. JES3 cannot be a secondary subsystem.
2. A secondary subsystem cannot be halted until all outstanding allocations are freed.

Accessing the EVENTLOG data set for job step completion codes

The EVENTLOG data set contains key job event information. The main information tracked in the data set is the completion code for every completed job step. This enhances the single completion code currently tracked for each job. In addition, EVENTLOG will contain records identifying when a job is restarted. The job step completion code and restart information allow the user to better understand the execution of each step in their job.

Note: The EVENTLOG data set is only available to JES2 users.

The EVENTLOG data set is allocated automatically for every batch job, started task, and TSO user. It contains machine readable records and is intended to be processed by applications. The data set is not considered to be SYSOUT so it cannot be accessed by SAPI/PSO/FSS. It is non-printable and non-spinnable. EVENTLOG is processed using the Spool Data Set Browse interface.

The job step completion code information is written to EVENTLOG in STEPDATA records, which contain the human readable eyecatcher STEPDATA. The full content of the record is defined by macro IAZLGSTP. Key information in the STEPDATA record includes the step number, step name, procedure name, program name, and step completion code. See IAZLGSTP for information on all the data contained in the record.

Job restart information is stored in RESTART records, which contain the human readable eyecatcher RESTART. There is one record written to EVENTLOG when the job ends and is re-enqueued to the execution queue. This record contains the human readable information JOB TERMINATED/RE-ENQUEUED. When the job restarts execution a second record containing the human readable information JOB RESTARTED will be written to EVENTLOG. The macro IAZLGRST defines the details of other data recorded in these RESTART records in EVENTLOG.

The EVENTLOG data set will contain some key SMF records to assist the user in obtaining vital information about the job. SMF Type 30 Subtype 1, 4, and 5 records will be written to the data set. Subtype 1 provides information on the start of a work unit. Subtype 4 contains job step completion information (Step Total). Subtype 5 contains information on the termination of a work unit. For a description of the content of these SMF records, see *z/OS MVS System Management Facilities (SMF)*.

Some facilities prefer to restrict the availability of SMF record information. If that is the case, the user can control whether the SMF records are written to the EVENTLOG data set. This is controlled through the SUP_EVENTLOG_SMF keyword on the JOBDEF command. The default value for this keyword is

SUP_EVENTLOG_SMF=NO, indicating that SMF records are not being suppressed and therefore will be written to EVENTLOG. Use the command \$T JOBDEF,SUP_EVENTLOG_SMF=YES to change this default behavior and suppress the writing of SMF records to any jobs entering the system once the command is executed.

Note: The setting of the SUP_EVENTLOG_SMF keyword is captured at the time the job enters the system and it affects what is written to the EVENTLOG data set for the life of the job.

As mentioned earlier in this section, the Spool Data Set Browse (SDSB) interface can be used to access the EVENTLOG data set. The data set name supplied to SDSB can be used to control which records in the EVENTLOG data set are returned to the application on a GET request. For example, if the application wants to retrieve all records in the EVENTLOG data set it can use the fully qualified data set name:

```
userid.jobname.jobID.D0000008.EVENTLOG
```

Additionally, the application can use the following logical data set name to retrieve all EVENTLOG records:

```
userid.jobname.jobID.EVENTLOG
```

For more information on specifying a data set name to the SDSB interface, see "Specifying the Data Set Name (DALDSNAM)" on page 4.

If the application wishes to access just the STEPDATA records in the EVENTLOG data set then the following logical data set name should be allocated:

```
userid.jobname.jobID.EVENTLOG.STEPDATA
```

In a similar manner, RESTART records are accessed using the logical data set name:

```
userid.jobname.jobID.EVENTLOG.RESTART
```

There are two options for retrieving SMF records. To retrieve all SMF records written to EVENTLOG, the application would specify the logical data set name:

```
userid.jobname.jobID.EVENTLOG.SMF
```

The application can retrieve only SMF records that contain job step completion information, which are SMF Type 30 Subtype 4 records, by allocating the following logical data set name:

```
userid.jobname.jobID.EVENTLOG.SMFSTEP
```

The various logical data set names supplied above will give the user application the flexibility needed to obtain the information it needs, and only that information, from the EVENTLOG data set.

Processing EVENTLOG records

The previous section discussed the various methods for allocating the EVENTLOG data set to obtain the desired records. This section will discuss how to interpret the data returned by the SDSB interface.

If the EVENTLOG data set has been allocated in a manner where all data set records will be returned, each record will consist of a record prefix followed by the record data. The record prefix is defined by the EVENTLOG record prefix (IAZLGINF) mapping macro. The prefix contains the information needed to determine the type of record returned by the SDSB interface. The record type is contained in field LGPRTYPT, and indicates what mapping macro should be used to interpret the record data. For example, if the returned record type is LGPRSMF, then the record data will contain an SMF record type and SMF macros would be used to interpret the record data. If the record type is LGPRSTEP, then the record data contains STEPDATA and the STEPDATA mapping macro IAZLGSTP is used to interpret the record data. A record type of LGPRRST, indicates a RESTART record and RESTART mapping macro, IAZLGRST, is used to interpret the record data. For more information on these macros refer to Record prefix (IAZLGINF) mapping macro, STEPDATA (IAZLGSTP) mapping macro, and RESTART (IAZLGRST) mapping macro. For more information on SMF records and how to interpret their data see *z/OS MVS System Management Facilities (SMF)*.

The EVENTLOG data set can also be allocated in a manner where a specific set of records are returned on the SDSB interface. Allocating any of the following logical data set names returns a specific set of records:

```
userid.jobname.jobID.EVENTLOG.STEPDATA
userid.jobname.jobID.EVENTLOG.RESTART
userid.jobname.jobID.EVENTLOG.SMF
userid.jobname.jobID.EVENTLOG.SMFSTEP
userid.jobname.jobID.EVENTLOG.USER
```

In these situations, the type of data being returned is known by the requestor so no record prefix is returned. Only record data is returned and the requestor uses the proper mapping macro to interpret that record type.

Writing to the EVENTLOG data set

In addition to reading records from the EVENTLOG data set that the system writes for the job, the user application running in the job also has the ability to write records to the EVENTLOG data set. This can be a useful feature for recording key information concerning the application execution. Caution should be exercised because the EVENTLOG data set is created for every batch job, started task, and TSO user. The amount of data being written to the EVENTLOG data set and the number of jobs that data will appear in should be considered.

A request to write a record to the EVENTLOG data set is initiated by invoking the EVENTLOG data service macro, IAZLGDAT. The application provides the type of record being written, the length of the record data, and the location of the record data. Optionally, the application can provide a work area for the EVENTLOG data service. Otherwise, the service will allocate the work area it needs to perform the write.

The EVENTLOG data service will validate the request and if valid it will perform a PUT to the EVENTLOG data set. The application can write a user-type record, using the record type LGPRUSER. There is also a record subtype field that can be used to provide further categorization of the records written to EVENTLOG. The subtype is defined by the user, along with any special structure or formatting within the user record data.

Once user records are written to EVENTLOG they can be read by allocating the full EVENTLOG logical data set name or the user record-specific logical data set name:

userid.jobname.jobID.EVENTLOG.USER

For information on defining a request to write data to the EVENTLOG data set see “EVENTLOG data service (IAZLGDAT) macro” on page 21.

EVENTLOG macros

Various mapping and executable macros are used to implement the EVENTLOG Data Service and to interpret the data returned by the SDSB interface. The following sections will introduce those macros and how they are used in the EVENTLOG environment.

Record Prefix (IAZLGINF) mapping macro

The IAZLGINF mapping macro provides the layout of the IAZLGINF DSECT used to interpret data stored in the record prefix of an EVENTLOG record. The format of an EVENTLOG record prefix is:

Field Name	Description
------------	-------------

LGPLENG	Record prefix length.
----------------	-----------------------

LGPRLEN	EVENTLOG record data length, excluding record prefix length.
----------------	--

LGPRTYP	EVENTLOG record type, consisting of a type and subtype.
----------------	---

LGPRTYPT	EVENTLOG record type.
-----------------	-----------------------

Valid values are:

LGPRSTEP	STEPPDATA, containing step completion code information. Mapped by the STEPPDATA (IAZLGSTP) mapping macro.
-----------------	---

LGPRRST	RESTART, containing job restart information. Mapped by the RESTART (IAZLGRST) mapping macro.
----------------	--

LGPRSMF	SMF record type. Mapped by SMF record mapping macros.
----------------	---

LGPRSFST	SMF STEP (SMF type 30, subtype 4) records. Mapped by SMF mapping macros.
-----------------	--

LGPRUSER	USER record type. Content defined by the user.
-----------------	--

LGPRTYPS	EVENTLOG record subtype. User defined values.
-----------------	---

LGPFFLAG	EVENTLOG record flag byte. All bits OFF indicate a REQUIRED record LEVEL value.
-----------------	---

Bit value	Description
LGP1STND	STANDARD record LEVEL value.
LGP1VERB	VERBOSE record LEVEL value.

STEPDATA (IAZLGSTP) mapping macro

The IAZLGSTP mapping macro defines the layout of the data contained in a STEPDATA record stored in the EVENTLOG data set. The format of a STEPDATA record, as defined by the STEPDATA DSECT, is:

Field Name	Description
STPSEYE	Eyecatcher. Set to "STEPDATA" for a normal STEPDATA record. Set to "ASIN" by the invoker if being used to interpret Active Step information. For more information on active step information, see "Special Processing for Logical EVENTLOG Data Sets" on page 11.
STPSLEN	Length of STEPDATA DSECT filled in.
STPSVER	Version of IAZLGSTP data returned.
STPSFLG1	Flag byte.
STP30TME	Step end time.
STP30DTE	Step end date.
STP30SID	System ID.
STP30SYN	System name where the job step ran.
STP30JBN	Job name.
STP30PGM	Program name from the EXEC statement.
STP30STM	Step name from the EXEC statement.
SPT30UIF	User identification.
STP30JNM	JES job identifier.
STP30STN	Step number.
STP30CLS	Job class (1 character).

STP30SIT
 Time from midnight of job select.

STP30STD
 Date initiator selected the job.

STP30USR
 Programmer's name.

STP30PSN
 Name of step invoking the procedure.

STP30CL8
 Job class (8 character).

STP30SSN
 Subsystem number for UNIX.

STP30EXN
 Program name.

STP30COR
 Job correlator.

STP30SCC
 Step completion code.

STP30STI
 Step/job termination indicator.

STP30ARC
 Abend reason code.

Note: The field names above, after the STP prefix, match the field names where the data is retrieved from the SMF type 30 subtype 4 record. For example, STP30SID is populated from the field SMF30SID found in the header section of the SMF type 30 subtype 4 record. For more information on these data fields, see *z/OS MVS System Management Facilities (SMF)*.

RESTART (IAZLGRST) mapping macro

The IAZLGRST mapping macro defines the layout of the data contained in a RESTART record stored in the EVENTLOG data set. The format of a RESTART record, as defined by the RSTREC DSECT, is:

Field Name	Description
RSTREYEC	Eyecatcher. Set to "RESTART".
RSTRLEN	Length of RESTART record.
RSTRVER	Version of RESTART record data.
RSTRFLAG	Flag byte.
	Flag byte
	Description
RSTRETXT	Re-enqueue text included in the record.

	RSTRRTXT	Job restarted text included in the record.
	RSTRSFLG	JCT restart flags (JCTJSFLG).
	Restart flag value	
	Description	
	RSTRSTRS	STEP restart.
	RSTRCHRS	CHECKPOINT restart.
	RSTRCNRS	Continue restart.
	RSTRHOLD	Hold the job after re-enqueue.
	RSTRBOPT	JCT job option flags (JCTJBOPT).
	Bit value	
	Description	
	RSTTHOLD	TYPRUN=HOLD.
	RSTNOLOG	No job log option.
	RSTINRDR	Job was entered on INTRDR.
	RSTRERUN	Job was rerun.
	RSTRSJF2	SJB restart flags (SJBFLG2).
	Restart flag value	
	Description	
	RST2EJST	\$EJOB,STEP was processed.
	RST2EOM	End-of-memory detected.
	RST2CNCL	CANCEL after SWA created.
	RST2HOLD	Hold job after re-enqueue.
	RSTRSJF4	SJB restart flags (SJBFLG4).
	Restart flag value	
	Description	
	RST4MEND	MSG 'ENDED'.

| **RST4MTRM**
 | MSG 'TERMINATED'.
 |
 | **RST4MREQ**
 | MSG 'RE-ENQUEUED'.
 |
 | **RST4MREX**
 | MSG 'QUEUED FOR RE-EXECUTION'.
 |
 | **RST4MRQH**
 | MSG 'RE-ENQUEUED AND HELD'.
 |
 | **RST4OCAN**
 | Operator cancelled this SJB.
 |
 | **RST4TERM**
 | Batch job has terminated.
 |
 | **RSTRTIME**
 | Time job re-enqueued/restarted (STCK).
 |
 | **RSTRSYSN**
 | JCT execution MVS system name.
 |
 | **RSTRJOBN**
 | Job name.
 |
 | **RSTRJBID**
 | Job ID.
 |
 | **RSTRSSTP**
 | Job step to restart (JCTJSSTP).
 |
 | **RSTRJCOR**
 | Job correlator.

EVENTLOG data service (IAZLGDAT) macro

| The IAZLGDAT macro is the interface to the EVENTLOG Data Service used to
 | write user records to the EVENTLOG data set. The following sections will define
 | how to invoke the service and in what environment it executes.

| The service requires a parameter list that provides the inputs necessary to write the
 | record. The IAZLGDAT macro expands code that fills in that parameter list. The
 | parameter list is defined by the IAZLGDDF mapping macro.

IAZLGDAT syntax

| The IAZLGDAT macro uses the following syntax:

```
|                   IAZLGDAT &TYPE=type,  

  |                   &SUBTYPE=subtype,  

  |                   &LEVEL=level,  

  |                   &DATA=dataptr,  

  |                   &DATALEN=datalen,  

  |                   &WORKAREA=workaptr,  

  |                   &WORKALEN=workalen,  

  |                   &MF=(E,paramlist),  

  |                   &DSECT=dsect
```

| *type* Specifies the type of record being written to the EVENTLOG data set. Valid
 | values are:

| **USER** User record type.

| *subtype* Record subtype being written to EVENTLOG. Optional parameter with no
| default. Values are defined by the user.

| *level* Level of the record being written. Indicates the importance of the record
| data. Currently used as a documentation-only field. Optional parameter,
| defaults to REQUIRED. Valid values are:

| **REQUIRED**
| This is a required entry and must be written.

| **STANDARD**
| This is a typical entry that should be written.

| **VERBOSE**
| This is an entry providing additional information and can be
| optionally written.

| *data* Address of the data being written to the EVENTLOG record. This is a
| required parameter.

| *datalen* Length of the data being written to the EVENTLOG record. Optional
| parameter, defaulting to the length of the DATA parameter field. The
| maximum length allowed is 32752 bytes. Any data beyond 32752 bytes will
| be truncated.

| *workaptr* Address of the work area that the EVENTLOG data service can use for its
| processing. This is an optional parameter. If supplied, the work area must
| be large enough to store the supplied record data plus 512 bytes. If the
| area is not this size, or larger, the request will fail with an error code in
| field LGDRETC and the value LGDRERR will be returned in R15. Field
| LGDREQSZ will contain the size of the work area required.

| If this parameter is not supplied, the EVENTLOG Data Service will allocate
| its own work area storage.

| *workalen* Length of the provided work area. Optional parameter, defaulting to the
| length of the WORKAREA parameter.

| **MF** Indicator for macro execution:

| **L** Allocates storage for the LGDTPLST DSECT that is used as the
| input parameter list to the IAZLGDAT service. MF must be
| invoked once with this indicator to set aside storage for the
| parameter list.

| **E** Generates the call to the IAZLGDAT service. Requires the list form
| (L) to have been previously specified.

| **Parmlst**
| Label used to reference the input parameter list, LGDTPLST, that is
| passed to the IAZLGDAT service.

| *dsect* Indicates whether the DSECT for the parameter list should be generated
| (YES) or not generated (NO). The default is blank. When used, this
| keyword is specified by itself. DSECT=YES indicates generating a DSECT
| statement along with the input parameter list. DSECT=NO indicates not
| generating a DSECT statement but to generate the input parameter list,
| and is typically used to imbed the input parameter list into another
| DSECT. Providing no DSECT parameter indicates this is an executable
| form of the macro.

Input register information

The IAZLGDAT service requires a non-standard save area address in register 13. This save area must be 128 bytes to allow the IAZTLGDAT service to save the 16 8-byte registers.

Output register information

The IAZLGDAT service affects the registers in the following manner after exiting the macro:

Register

Content

R0/AR0

Destroyed. R0 is used as a work register.

R1/AR1

Destroyed. R1 is used as a work register.

R2-R13

Unchanged.

R14

Destroyed. Used as a return address.

R15/AR15

Destroyed. R15 contains a return code.

Return code information

The IAZLGDAT service supplies a return code in register 15 after exit. The following return codes are possible:

Return Code

Meaning and Results

- 0 The IAZLGDAT service was called successfully.
- 4 The record was not written to EVENTLOG.
- 8 No storage available for work area.
- 12 Invalid LEVEL requested for the record.
- 16 Parameter list address or length is zeros.
- 20 Request not processed due to error. For more information, refer to field LGDRETCO.

Environment

Minimum Authorization: Problem or Supervisor state, with any PSW key

Dispatchable unit mode: Task

Cross Memory Mode: PASN=HASN=SASN

AMODE: 31 or 64 bit

ASC mode: Primary

Locks: none

Restrictions

None.

EVENTLOG data service data definition (IAZLGDDF) macro

The EVENTLOG data service data definition macro, IAZLGDDF, is used to map the input parameter list passed to the IAZLGDAT EVENTLOG data service. The LGDTPLST DSECT defines the service's input parameter list. The IAZLGDDF macro is invoked through the IAZLGDAT macro in the following manner:

IAZLGDAT DSECT=YES|NO

Input parameter list definition:

DSECT=YES

Generates a DSECT statement for the parameter list structure.

DSECT=NO

Does not generate a DSECT statement, and can be used to reserve storage for the parameter list in an existing DSECT.

IAZLGDDF parameter list return codes (LGDRETCD)

The following EVENTLOG data service data definition return codes are provided:

Name	Meaning
------	---------

LGDRSUCC	Successful completion.
-----------------	------------------------

LGDRNOTW	Record not written.
-----------------	---------------------

LGDRNOST	No storage available for work area.
-----------------	-------------------------------------

LGDRBADP	Parameter list address or length is zero.
-----------------	---

LGDRERR	Request not processed due to error. For more information, refer to the LGDRETCD field.
----------------	--

The following return codes can be provided in field LGDRETCD when the return code in R15 comes back with the value LGDRERR:

Name	Meaning
------	---------

LGDRBADF	Invalid function requested.
-----------------	-----------------------------

LGDRNACB	ACB for the EVENTLOG data set is not found.
-----------------	---

LGDRNOUT	The pointer to the record data or the data length is zeros.
-----------------	---

LGDRINWA	Insufficient space provided in the caller work area. Look in field LGDREQSZ for the work area size needed to process this data record.
-----------------	--

LGDRMAXL	Data length is greater than the maximum allowed. The record will be truncated.
-----------------	--

LGDRLHLD	Local lock already held.
-----------------	--------------------------

LGDRFRR	FRR active, request can not be completed.
----------------	---

| **LGDRBADT**
| Record type is invalid.

| **LGDRBADL**
| LEVEL indicator is invalid.

| **LGDRNAUT**
| Caller is not running authorized.

| **LGDRABND**
| Abend occurred while processing the request.

| **LGDRRECS**
| Record type(s) are being suppressed.

| **IAZLGDAT service input parameter list (LGDTPLST)**

| The LGDTPLST data structure is the input parameter list to the IAZLGDAT
| EVENTLOG data service. It is built by the code expanded by the invocation of the
| IAZLGDAT service macro. The parameter list fields are:

| **Field name**
| **Description**

| **LGDEYE**
| Eyecatcher. Expected value is LGDTPLST.

| **LGDLLEN**
| Length of LGDTPLST parameter list.

| **LGDFUNC**
| Service function to perform.

| **LGDLEVEL**
| LEVEL of record being written. Byte flag.

| **Bit value**
| **Description**

| - First two bits OFF indicates REQUIRED.

| **LGDLSTND**
| STANDARD level.

| **LGDLVERB**
| VERBOSE level.

| **LGDRTYPT**
| Record type to be written.

| **LGDRTYPS**
| Record subtype to be written.

| **LGDDLLEN**
| Length of record data to be written.

| **LGDWALEN**
| Length of caller-supplied work area.

| **LGDDATAP**
| Address of the record data to be written (64 bit).

| **LGDDATA4**
| Address of the record data to be written (31 bit).

| **LGDWORKP**
| Address of the caller-supplied work area (64 bit).

| **LGDWORK4**

| Address of the caller-supplied work area (31 bit).

| **LGDRETCD**

| Return code generated by this request.

| **LGDREQSZ**

| Size of caller-supplied work area required to process this record data.

| The following return codes can be provided in field LGDRETCD when the return
| code in R15 comes back with the value LGDRERR:

| **Name Meaning**

| **LGDRBADF**

| Invalid function requested.

| **LGDRNACB**

| ACB for the EVENTLOG data set is not found.

| **LGDRNOUT**

| The pointer to the record data or the data length is zeros.

| **LGDRINWA**

| Insufficient space provided in the caller work area. Look in field
| LGDREQSZ for the work area size needed to process this data record.

| **LGDRMAXL**

| Data length is greater than the maximum allowed. The record will be
| truncated.

| **LGDRLHLD**

| Local lock already held.

| **LGDRFRR**

| FRR active, request can not be completed.

| **LGDRBADT**

| Record type is invalid.

| **LGDRBADL**

| LEVEL indicator is invalid.

| **LGDRNAUT**

| Caller is not running authorized.

| **LGDRABND**

| Abend occurred while processing the request.

| **LGDRRECS**

| Record type(s) are being suppressed.

Chapter 3. JES Client/Server Print Interface

JES provides an interface for a job to function as a server and make SYSOUT requests on behalf of a client.

There are several ways in which a data set created by a server differs from a data set created by an ordinary SYSOUT DD or dynamic allocation.

1. Data sets created by a server use the DALRTCTK dynamic allocation text unit, which causes JES to create a unique **Client Token**(CTOKEN) associated with the data set from that point on.
2. The server can use the Extended Status or SYSOUT Application Programming Interface (SAPI) Subsystem Interface (SSI) calls to access a data set, specifying a CTOKEN in the selection criteria in order to request a particular data set, without needing to know any other information about the data set.
3. When a data set has a CTOKEN, JES informs the application, through the use of ENF signal 58, of events relating to the data set. Among these events are selection by a writer, deselection by a writer, and data set purge. JES also issues signals for important events related to any job that has created at least one data set with a CTOKEN, such as job purge.

Creating a CTOKEN

The server creates an 80-byte CTOKEN using the Dynamic Allocation text unit of DALRTCTK. The DALRTCTK text unit appears as follows:

+0	+2	+4	+6
DALRTCTK	00 01	00 50	Returned data

Upon return from SVC 99, the data starting at position 6 in the CTOKEN text unit contains the CTOKEN returned by JES, provided the allocation was successful. When a CTOKEN is returned to you, add it to your list of CTOKENs for later use.

CTOKENs contain internal information that JES uses to locate the data set when the server issues SSI requests. Once you have received a CTOKEN from JES, do not change its contents except in one special case that will be discussed later.

CTOKENs contain ordering information. This allows you to store CTOKENs in a data structure of your choice that can make use of the order and result in faster searches. CTOKENs are not ordered according to a creation timestamp; they are ordered internally by JES.

Refer to SSI 54 in *z/OS MVS Using the Subsystem Interface* for a detailed description of the Subsystem Version Information Call.

Comparing CTOKENs

At various times during your processing, you will need to compare CTOKENs. Typically, you will do this when JES signals an event for a CTOKEN and you need to find this token in your list so that you can take some kind of action based on the event.

To compare one CTOKEN to another, you must not simply compare the entire 80 byte values. This is because under certain JES processing, CTOKEN equality is based on a subset of the information in the CTOKENs matching while other information in the CTOKENs could be different. IBM provides a macro IAZXCTKN which you must use to compare CTOKENs. This macro determines which information in two CTOKENs is significant and compares just this information. The macro works in such a way that you never need to interpret any information inside the CTOKEN.

Depending on the return code from the IAZXCTKN macro, you can determine whether the two CTOKENs are the same, whether the first CTOKEN is less than the second one, or whether the second CTOKEN is less than the first one.

IAZXCTKN also provides a special comparison function. At certain times, JES signals events for an entire job. When this happens, the signal includes a **job level CTOKEN**. Using IAZXCTKN, you can determine whether a CTOKEN for a data set in which you are interested is covered by the job level CTOKEN that JES provides.

Job level CTOKENs contain no ordering information; therefore a job level CTOKEN can be considered by IAZXCTKN to be "equal" or "not equal" to another CTOKEN but never "greater" or "less" than another CTOKEN.

Refer to the book *z/OS MVS Programming: Authorized Assembler Services Guide* for information about using the IAZXCTKN macro.

Obtaining Status for a Data Set

You can obtain status for a data set using the Extended Status subsystem interface (SSI 80) code. To do this, you supply as STATCTKN the address of the CTOKEN for the data set you are interested in and set the selection flag STATSCTK. When you use the STATSCTK selection flag, you cannot use the STATSJBI selection flag, and vice versa.

Refer to SSI 80 in *z/OS MVS Using the Subsystem Interface* for a detailed description of the Extended Status call.

Accessing a Data Set

You can access a data set using the SYSOUT Application Programming Interface, SSI 79. You would do this in order to:

- Show the contents of the data set to the requesting client.
- Allow the client to delete a data set.
- Allow the client to release a data set from hold to print.

To request JES to perform a SAPI operation on a client data set, you supply as SSS2CTKN the address of the CTOKEN for the data set you are interested in and set the selection flag SSS2SCTK. When you use the SSS2SCTK selection flag, you cannot use the SSS2SJBI selection flag, and vice versa.

When a data set is processed by a program written using SAPI, there is a distinction between a data set that is selected for processing and a data set that is selected for browsing. In the former case, the intention is to select the data set in much the same way as it would be selected for a writer (such as an external writer), which may or may not cause its state to be changed. In the latter case, the

intention is to not change its state at all. The main purpose for this distinction is to prevent "noise" caused by unnecessary ENF signals.

You can set the flag SSS2SBRO when you know that the intention of a SAPI access is to browse a data set. When this flag is on, JES will not issue any signals for the SAPI access to this data set. When this flag is off JES will issue signals whenever a data set with a CTOKEN is selected or deselected by a SAPI Put/Get operation. Do not set this flag if you need to be informed of selects and deselects.

This flag controls signals for selects and deselects only. If a data set is purged by a SAPI Put operation (for example, by turning off flag SSS2DKPE), a signal will be issued even if SSS2SBRO is set.

You must use SAPI in order to suppress signals when accessing a data set for browse. When a Process Sysout (PSO) application selects or deselects a data set with a CTOKEN, a signal is always issued.

The SSS2SBRO flag is valid only for Put/Get requests.

Refer to SSI 79 in *z/OS MVS Using the Subsystem Interface* for a detailed description of SAPI.

Security

Since all SYSOUT allocations and SAPI calls are being done by you as the server, preventing a client from having unauthorized access to another client's data set is your responsibility.

One way you can do this is by performing the dynamic allocation to create the SYSOUT file under a security environment with the client's identity. This is accomplished by using the RACROUTE macro with REQUEST=VERIFY. Then, when a client makes a request requiring you to make a SAPI SSI call, you would use RACROUTE REQUEST=VERIFY with the requesting client's user id to establish a security environment for the requestor. As part of the SAPI processing, JES makes authorization checks using the JESSPOOL security class.

Refer to the book *z/OS Security Server RACROUTE Macro Reference* for information on using the RACROUTE macro.

This method requires your clients to be defined as users in your security product, even if they never directly log on to your system. If this is not possible, you must design your own security protocol.

Identifying a Requestor on a Header Page

JES typically has printers defined to print with a header page identifying the job creating a SYSOUT data set.

However, the job information that prints on the header page is associated with the job that created the data set. This ordinarily identifies the job that runs your server, not the client that requested the printout. You would probably prefer that the client's identification print rather than the server's in order to be able to tell one client's output apart from another's.

In order to do this, you can use the IAZXJSAB macro. You could do something like the following:

Print Interface

```
IAZXJSAB CREATE,JOBNAME=client_jobname,  
          USERID=client_userid,TYPE=SUBTASK
```

The CREATE call must be made prior to allocating the dataset in both the DYNALLOC and ALLOC cases.

To make sure that the job identification does not persist beyond the requested data set, you can delete the JSAB after the first OPEN for the dataset by making the following call:

```
IAZXJSAB DELETE,TYPE=SUBTASK
```

or you can update it with different user identification after the first OPEN by making the following call:

```
IAZXJSAB UPDATE,JOBNAME=new_client_jobname,  
          USERID=new_client_userid
```

In the CREATE and DELETE cases, you must use the parameter TYPE=SUBTASK, otherwise JES will not recognize the requesting user identification correctly.

See the information on the IAZXJSAB macro in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for additional information about using the IAZXJSAB macro.

Listening for Events

During the course of JES operations, data sets and jobs are subject to changes for various reasons. When such events occur for a client data set (for example, a data set that was allocated with the DALRTCTK text unit) or a job containing at least one client data set, JES issues an Event Notification (ENF) signal. The ENF number of the signal is 58. The signal is issued only for data sets that have been allocated using the DALRTCTK text unit.

To listen for this signal, you could do something like the following:

```
ENFREQ ACTION=LISTEN,CODE=58,EXIT=exit_address,XSYS=YES,  
       PARM=parameter_address,DTOKEN=end_token_address
```

Note: The XSYS=YES parameter is used because JES could be issuing signals on a different processor from the one where your server runs.

To stop listening for this signal, you could do something like the following:

```
ENFREQ ACTION=DELETE,CODE=58,DTOKEN=end_token_address
```

The data area received by your listen exit from ENF is mapped by the IAZENF58 macro. You must include this macro in your program in order to use the data supplied by the ENF signal. This data area contains the following information:

ENF58_LENGTH

Length of parameter list

ENF58_QUALIFIER

Qualifier code — defined below:

ENF58_Q_PURGE

Data set was purged

ENF58_Q_SELECT

Data set was selected

ENF58_Q_DESELECT_PROCESSED

Data set was processed

ENF58_Q_DESELECT_NOT_PROCESSED

Data set is no longer selected, disposition was not changed

ENF58_Q_DESELECT_NOT_PROCESSED_HELD

Data set is no longer selected, disposition was not changed and data set is held

ENF58_Q_DESELECT_ERROR

An error resulting in a system level hold occurred

ENF58_Q_EOD_OK

End of data set notification occurred — successful

ENF58_Q_EOD_ERROR

End of data set notification occurred — unsuccessful

ENF58_Q_JOB_CHANGE

Job-status change occurred

ENF58_Q_TOKEN_CHANGE

Client token has changed

ENF58_Q_CHECKPOINT

A checkpoint has occurred on the printer on which the data set is printing.

ENF58_SYS_HOLD

System hold reason — refer to IAZOHL D for possible values

ENF58_JES_NAME

JES2 Member Name / JES3 MAIN name

ENF58_REASON

Reason text

ENF58_CTOKEN

Data Set Client Token

ENF58_NEW_CTOKEN

New client token that should replace the CTOKEN for a TOKEN_CHANGE ENF type

You should determine what action you need to take based on this event. For example, if you receive a signal with ENF58_Q_PURGE it usually means that you should delete from your list all information pertaining to the dataset with the CTOKEN of ENF58_CTOKEN.

To take action on the CTOKEN, you must first go through your CTOKEN list and issue IAZXCTKN macros, comparing ENF58_CTOKEN to CTOKENs from your list until you find the CTOKEN specified in the signal in your list. If ENF58_QUALIFIER is ENF58_Q_JOB_CHANGE, it means that ENF58_CTOKEN is a job level CTOKEN and you must go through your entire list of CTOKENs until you have identified, and taken action on, all data set level tokens covered by the job level CTOKEN.

Note:

1. When ENF58_QUALIFIER is ENF58_Q_JOB_CHANGE, the CTOKEN in ENF58_CTOKEN is a job level CTOKEN. At all other times it is a data set level CTOKEN.

Print Interface

2. When ENF58_QUALIFIER is ENF58_Q_TOKEN_CHANGE, the ENF58 parameter list contains a new CTOKEN and ENF58_LENGTH reflects the existence of this new CTOKEN.
3. When an event with ENF58_Q_TOKEN_CHANGE is received, the CTOKEN in your list should be replaced with the contents of ENF58_NEW_CTOKEN. This is the only time that you should change the contents of a CTOKEN. Replacing this CTOKEN does not change the ordering of the CTOKEN you previously had in your list for this data set.
4. ENF58_NEW_CTOKEN is present only when ENF58_QUALIFIER is ENF58_Q_TOKEN_CHANGE. ENF58_LENGTH is larger for this qualifier type than it is for other types.
5. When an event with ENF58_Q_CHECKPOINT is received, the below fields are also present and contain the status of the print at the time that the checkpoint is taken.

ENF58_COPY

Checkpointed copy count

ENF58_RECORD

Checkpointed current record

ENF58_PAGE

Checkpointed current page

These fields are present only when ENF58_QUALIFIER is ENF58_Q_CHECKPOINT. ENF58_LENGTH is larger for this qualifier type than it is for other types except ENF58_NEW_CTOKEN.

If checkpoints occur frequently, ENF58 checkpoint signals may be generated at a fast rate.

If a restart of JES or the printer occurs after a checkpoint, the next checkpoint could be for a page or record count that represents reprocessed records or pages. This is because after a restart a writer will continue at the last checkpointed record or page, not the last one that completed printing.

6. Trace ID 43 traces ENF58 events that are sent and trace ID 44 traces ENF58 events that are received.

See *z/OS MVS Programming: Authorized Assembler Services Guide* and *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for information about using the ENFREQ macro and coding the listen exit.

Chapter 4. JES Symbol Service (IAZSYMBL)

The JES Symbol Service (IAZSYMBL) provides a single view of JCL and JES symbols. The JES Symbol Service consists of the **IAZSYMBL** invocation macro and the **IAZSY MDF** data definition macro.

The JES Symbol Service manages JES Symbols, which can be used in the following ways:

1. Several special purpose JES Symbols can be used to pass information between applications and JES: refer to “JES system symbols” on page 34.
2. JES symbols can be used internally by an application. As a method of communication, a JES symbol that is created by one program can be consumed by another program within the same job step.
3. JES symbols can be passed on to a submitted job by including the symbol names in the **SYMLIST=** keyword of an internal reader (INTRDR) allocation. JES symbols passed in this way must be valid JCL symbol names, which consist of 1-8 characters from the subset A-Z (capitals only), 0-9 (numerics), @ (at character), # (number sign character) and \$ (dollar sign character).
4. JES symbols can be used for in-stream symbol substitution in the same job step. JES in-stream substitution is performed when in-stream data set is read. During this substitution, the following symbols can be used:
 - JCL symbols EXPORTed by converter
 - JES symbols dynamically created by the JES Symbol Service
 - System symbols

The specific symbols to use for substitution are defined using the **SYMBOLS** keyword parameter on the DD statement that defines the in-stream data set. Refer to *z/OS MVS JCL Reference*.

5. JES Symbols can be used to communicate information between applications and JES.

The JES Symbol Service manages two classes of symbols, JCL Symbols and JES Symbols:

JCL symbols

JCL symbols are defined by the **EXPORT JCL** statement and made available by the converter at job execution time. JCL symbols can be read but not updated by the JES Symbol Service. However, the JES Symbol Service can be used to create a JES symbol with the same name as JCL symbol, which overrides the JCL symbol of the same name. Rules for JCL Symbol Service (**IEFSJSYM**) names and values are documented in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*.

JES symbols

JES symbols are created and managed by the JES Symbol Service at the job step level or current task level. Rules for JES symbol names and values are the same as those for JCL symbols, with the following exceptions:

1. The underscore character (**_**) can be used in a JES symbol name.
2. A JES symbol name can be 1-16 characters long.
3. A JES symbol value can be 0-4096 bytes long.

Note:

1. See the following special rules for the **SYS_CORR_USRDATA** symbol, which must be printable.
2. JES Symbols that are exported using the **INTRDR SYMLIST** feature must conform to JCL rules, or a JCL error will result.

JES system symbols

JES symbols with the prefix **SYS** are system symbols, which communicate information between applications and the system. JES **SYS** symbols are read-only and applications are not allowed to manage them, with two exceptions. The **SYS_CORR_USRDATA** and **SYS_JOB_NOTIFY** symbols, which communicate information from applications to JES, are user-defined and subject to the following rules:

SYS_CORR_USRDATA

The **SYS_CORR_USRDATA** symbol defines the user portion of the job correlator. The job correlator is an attribute that is associated with every job on a system, and can be used to uniquely identify a job. The job correlator is a 64-character printable value, consisting of two strings separated by a colon character (:). The first string is a 31-character system value, which applications must always treat as a single value. The second string is an optional 32-character user-defined value.

To define the user portion of the job correlator, use the JES Symbol Service to assign a value to the **SYS_CORR_USRDATA** symbol before the job is submitted. The **SYS_CORR_USRDATA** symbol value must comply with the following rules:

- The value must be 0-32 characters in length (an empty value is valid, and equivalent to a value of all blank characters).
- The first character must be from the subset A-Z (capitals only), @ (at character), # (number character), or \$ (dollar sign character).
- Subsequent characters must be from the subset A-Z (capitals only), 0-9 (numerics), @ (at character), # (number character), \$ (dollar sign character) and _ (underscore character).
- Embedded blank characters are not supported.

SYS_JOB_NOTIFY

Assigning a value to the JES **SYS_JOB_NOTIFY** symbol, before submitting a job, prompts JES to provide notification when the job is no longer eligible for execution. Notification is provided by ENF 78 upon successful completion of the job, or upon an error, cancellation, or purge that prevents the job from being executed. The only restriction on the **SYS_JOB_NOTIFY** value is a maximum length of 4096 bytes.

The following JES System Symbols are system-defined, read-only, and provide a flow of information from JES to applications:

SYS_CORR_CURRJOB

The **SYS_CORR_CURRJOB** value is the job correlator of the current job.

SYS_CORR_LASTJOB

The **SYS_CORR_LASTJOB** value is the job correlator of the most recent job that was submitted successfully by the current task through the internal reader (INTRDR). If a job submission through the internal reader fails, this symbol will have an empty value. If a job submission through

the internal reader succeeds, this symbol is set to the job correlator of the submitted job, including any user portion that is defined by the **SYS_CORR_USRDATA** symbol.

SYS_LASTJOBID

The value of the **SYS_LASTJOBID** symbol is the job identifier of the most recent job that was successfully submitted by the current task through the internal reader (INTRDR). If a job submission through the internal reader fails, this symbol will have an empty value.

SYSUID

The value of the **SYSUID** symbol represents the user ID under whose authority the job will run. A detailed description of the **SYSUID** symbol can be found in the section, Using the **SYSUID** system symbol in Chapter 5, Procedures and symbols in *z/OS MVS JCL Reference*.

SYSUID is a special type of JCL symbol which is set and maintained by JES. Unlike other JCL symbols that are only made available at job execution time by the **EXPORT** JCL statement, the value for the **SYSUID** symbol can be extracted even if it was not explicitly **EXPORTed** in the job's JCL stream.

The value returned by the JES Symbol Service is the same as the one used during conversion, except when the **SYSUID** symbol was modified by the **SET** JCL statement but not **EXPORTed**. In this case, the conversion will use the modified value of the symbol, whereas the JES Symbol Service will return the original value. If this is not the desired result, use the **EXPORT** JCL statement to ensure that the value returned by the JES Symbol Service is the same as that used by conversion.

Note: JES3 does not support Job Correlator functionality, nor these JES symbols:

- **SYS_CORR_USERDATA**
- **SYS_CORR_CURRJOB**
- **SYS_CORR_LASTJOB**

JES Symbol (IAZSYMBL) macro

The interface to the JES Symbol application is the **IAZSYMBL** macro. The parameter structure of the JES Symbol Service is mapped by the **IAZSYMDF** macro. The **IAZSYMBL** service performs the following operations on JES symbols:

- Creates symbols at the task or job step level and assigns initial values
- Clears symbols
- Updates symbol values
- Deletes symbols
- Extracts symbol values

IAZSYMBL syntax

The **IAZSYMBL** macro uses the following syntax:

IAZSYMBL PARM=prmlst | (reg) | <null>

The **IAZSYMBL PARM=** keyword can be defined as a parameter list, register or null value (blank):

prmlst Specifies an RX-type address of the parameter list that is passed to the **IAZSYMBL** service. The parameter list structure and data which are returned by the service are mapped by the **IAZSYMDF** macro.

(*reg*) Specifies that a parameter list address was loaded by the caller to the register *reg*.

<null>

If the **PARM=** keyword is omitted, the default location for the parameter list address loaded by the caller is register 1.

The return code is located in register 15. If the invocation was successful (R15=0), then the return code from the **IAZSYMBL** service is listed in the **JSYMRETN** field in the parameter list.

Input register information

Before issuing the **IAZSYMBL** macro, the caller does not have to place any information into any register, unless it is either using the information in register notation for a **PARM** parameter, or as a base register.

Output register information

The following **IAZSYMBL** register usage values indicate the status of the register upon exiting the macro:

Register Contents

- | | |
|------|---------------------------------------|
| 0 | Unchanged |
| 1 | Used as work registers by the system |
| 2-13 | Unchanged |
| 14 | Used as a work register by the system |
| 15 | Contains the return code |

Return code information

The following **IAZSYMBL** return code information indicates the status of the register upon exiting the macro:

Return Code

Meaning and Action

- | | |
|---|--|
| 0 | The service was successfully called. To check the result of the call, refer to the return code in the JSYMRETN field and to the reason code in the JSYMREAS field of the parameter list. |
|---|--|

No action is required.

- | | |
|---|--|
| 8 | The parameter list is unusable because of one of the following errors: |
|---|--|

- No parameter was passed
- Eyecatcher was incorrect
- Parameter list version was incorrect
- Parameter list has incorrect length

Check the parameters that are being passed to the service and repeat the request.

- | | |
|----|---|
| 12 | There is not enough storage to invoke the JES Symbol Service. |
|----|---|

Increase the size of the main storage available to the application and repeat the request.

- | | |
|----|-------------------------------|
| 16 | The service is not available. |
|----|-------------------------------|

Report the problem to the system programmer for problem determination and correction.

Environment

The IAZSYMBL environment requires the CVT and IHAECVT macros to map CVT and ECVT, respectively:

Minimum authorization

Problem or Supervisor state, with any PSW key

Dispatchable unit mode

Task, unless JSYMLVNJ is used

Cross Memory Mode

PASN=HASN=SASN

AMODE

31-bit

ASC mode

Primary

Locks No locks held unless option JSYMLVNJ is used.

Restrictions

Access to JCL symbols requires that the caller does not hold any locks and is in task mode: refer to “DELETE and EXTRACT symbols” on page 41. Specifying the JSYMLVNJ option removes these restrictions, but also prevents access to the JCL symbols.

JES Symbol Service data definition (IAZSY MDF) macro

The JES Symbol Service data definition (**IAZSY MDF**) macro is used to map the parameter structure that is passed to the JES Symbol Service and the data structures that are returned by the service.

Access to the **IAZSY MDF** macro is defined by the following syntax:

IAZSY MDF DSECT=YES | NO

Controls access to the **IAZSY MDF** macro:

DSECT=YES

Generates a DSECT statement for the parameter list structure.

DSECT=NO

Does not generate a DSECT statement for the parameter list structure. Setting **DSECT=NO** can be used to reserve space for the parameter list in the current CSECT or DSECT.

Return codes (JSYMRETN)

The following JES Data Definition macro return codes (JSYMRETN) are provided:

Field Name

Description

JSYMOK

Request successful.

JSYMERRW

Request completed with possible errors; “Reason codes (JSYMREAS)” on page 38 contains the reason code.

JSYMERRU

Request not completed due to user error; “Reason codes (JSYMREAS)” on page 38 contains the reason code.

JSYMERRJ

Request not completed due to an internal (JES) error; "Reason codes (JSYMREAS)" contains the internal JES reason code.

Reason codes (JSYMREAS)

The following JES Data Definition macro reason codes (JSYMREAS) apply to non-zero return codes:

Field Name**Description****JSYMNOTF**

Some or all of the symbols were not found.

JSYMSTRE

Not enough storage provided by the caller (refer to the JSYMSRCM field).

JSYMOPER

Invalid operation requested.

JSYMSLEV

Invalid symbol level or scope.

JSYMINTB

Invalid input symbol table.

JSYMTRNC

Some values were truncated.

JSYMDUP

Duplicate symbols.

JSYMAUTH

Caller not authorized to perform this operation or caller not authorized to manage system symbols.

JSYMNMER

Invalid symbol name.

JSYMNSTG

Not enough storage for symbol table.

JSYMSSVE

Invalid value for a system symbol.

JSYMISTG

Extract not complete due to storage shortage for internal processing.

JSYMISSNE

Incorrect length of symbol name in the input symbol list.

JSYMSPSY

Special symbols can be extracted by name but cannot be managed as symbols.

JSYMNENVE

Environment error - function not available in SRB mode.

Parameter list (JSYMPARM)

The JSYMPARM data structure is a JES Symbol Service parameter. A pointer to this parameter list is passed in register 1 when calling the service (refer to the Chapter 4, "JES Symbol Service (IAZSYMBL)," on page 33 macro).

Field Name	Description
JSYMEYE	Eyecatcher.
JSYMLNG	Length of parameter list.
JSYMVRM	Parameter version and modification.
JSYMVER	Parameter version.

Field Name	Description
-------------------	--------------------

JSYMVRM1	Original version and modification.
-----------------	------------------------------------

JSYMVRMC	Latest version and modification.
-----------------	----------------------------------

JSYMMOD	Parameter modification.
----------------	-------------------------

JSYMSVER	Service version and modification.
-----------------	-----------------------------------

JSYMSVRM	Service version.
-----------------	------------------

Field Name	Description
-------------------	--------------------

JSYMSVM1	Original service version and modification.
-----------------	--

JSYMSVMC	Latest service version and modification.
-----------------	--

JSYMSMOD	Service modification
-----------------	----------------------

Requested operations (JSYMRQOP)

The following JES Data Definition macro operations (JSYMRQOP) can be requested:

Field Name	Description
-------------------	--------------------

JSYMRQOP	Specifies the requested operation:
-----------------	------------------------------------

CREATE (JSYMCRT)	Given the input symbol table provided by the caller in JSYMISYT , the service will create the specified symbols with the specified values.
-------------------------	---

CLEAR (JSYMCLR)	Deletes all defined symbols at the specified levels. This operation is only available for authorized callers.
------------------------	---

UPDATE (JSYMUPDT)

Given the input symbol table provided by the caller in **JSYMISYT**, the service will update the specified symbols with the new specified values.

DELETE (JSYMDELE)

Given the symbol filter specified by the caller in **JSYMSNMA/JSYMSN#**, the service will delete the specified symbols. Refer to “DELETE and EXTRACT symbols” on page 41.

EXTRACT (JSYMEXTR)

Given the symbol filter specified by the caller in **JSYMSNMA/JSYMSN#**, the service will return the output symbol table with the names and values of the requested symbols. The EXTRACT subfunction provides access to both JES and JCL symbols, unless the **JSYMLVNJ** option is specified. Refer to “DELETE and EXTRACT symbols” on page 41.

Defining JES symbols

JES Symbol Service (IAZSYMBL) symbols are defined at either the task level or the job step level. JES symbols that are defined at the task level are only visible to the code that is running in the same task (TCB). Symbols that are defined at the job step level are visible to the code that is running in all tasks in the same job step. A JES symbol at the task level overrides a JES symbol with the same name at the job step level.

The JES Symbol Service manages JES symbols by using the **CREATE**, **CLEAR**, **UPDATE**, **DELETE** operations. In addition, the **EXTRACT** operation provides access to JCL Symbols, unless the **JSYMLVNJ** option is specified. When access to JCL symbols is required, the caller must be in the task mode and cannot hold any locks. When access to a JCL symbol is requested, the JCL symbol is only returned if no JES symbol with the same name was found.

The **CREATE** operation requires the symbol definition level to be selected by setting either the **JSYMLVLT** or **JSYMLVLJ** option, but not both.

The **CLEAR** operation requires the symbol definition level to be selected by setting the **JSYMLVLT** or **JSYMLVLJ** option, or both.

The **UPDATE**, **DELETE** and **EXTRACT** operations ignore the symbol definition level selection. Processing for these operations always starts at the task level and only moves to the job step level if the requested symbol was not found at the task level. If the EXTRACT function does not find a JES symbol with a particular name, it will search for a JCL symbol with the same name.

The **JSYMLVUD** option modifies the behavior of a **CREATE** operation request when duplicate symbols are encountered. If the **JSYMLVUD** option is specified, duplicate symbols are updated. If the **JSYMLVUD** option is not specified, duplicate symbols are not processed and a duplicate symbol warning is returned.

The **JSYMLVJC** option applies JCL constraints during **EXTRACT** operation processing. If the **JSYMLVJC** option is specified, only symbols that can be used for JCL substitution are returned. The JCL constraints are a maximum symbol name length of 8 characters, and a maximum symbol value length of 255 characters. Rules for JCL Symbol Service (**IEFSJSYM**) names and values are documented in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*.

JES symbol options (JSYMLVL)

The JSYMLVL field specifies the following JES symbol options:

Field Name	Description
------------	-------------

JSYMLVLT	Access symbols at the task level.
----------	-----------------------------------

JSYMLVLJ	Access symbols at the job step level.
----------	---------------------------------------

JSYMLVNJ	Do not access JCL symbols. This option only applies to an EXTRACT request. If this option is not selected, an EXTRACT request will continue looking for the requested symbol or symbols among the exported JCL symbols. When accessing JCL symbols, the caller must be in task mode and cannot be holding any locks.
----------	--

JSYMLVUD	Allow a CREATE request to update a symbol value if the symbol already exists at the specified level. If this option is not specified, the duplicate symbol will not be changed and a duplicate symbol warning will be returned.
----------	---

JSYMLVJC	Check JCL constraints. This option only applies to an EXTRACT request with wildcard symbol selection. If this option is selected, an EXTRACT request will not return symbols that match the wildcard selection but cannot be used as valid JCL symbols.
----------	---

Note: This option will also suppress symbols defined by JES, such as the SYSUID symbol, unless they were explicitly exported using the EXPORT JCL statement.

Input symbol table (JSYMISYT)

The JSYMISYT option points to an input symbol table. The caller passes this table to CREATE and UPDATE operations to define names and values of the symbols to create or update. The layout of the symbol table is defined by the JSYTABLE structure.

Field Name	Description
------------	-------------

JSYMISYT	Pointer to an input symbol table.
----------	-----------------------------------

DELETE and EXTRACT symbols

The JES Symbol Service uses the symbol selection list to specify symbols for the DELETE and EXTRACT operations. Each element in the selection list specifies the name of a symbol to be selected. Symbol names must be left-justified and padded by blank spaces to the specified length. Wildcard characters can be used in any element in the selection list. If the EXTRACT function does not find a JES symbol with a particular name, it will search for a JCL symbol with the same name.

The selection list is defined by the following fields:

Field Name	Description
------------	-------------

JSYMSNMA

Pointer to a symbol selection list.

JSYMSNM#

Number of elements in the symbol selection list.

JSYMSNML

Length of each element in the selection list. Valid values for this field are 0-16 characters. A value of 0 defaults to 16 characters.

The EXTRACT operation returns a symbol table with the names and values of the symbols that were found. The output table is created in the output area provided by the caller. The layout of the symbol table is defined by the **JSYTABLE** data structure.

The total size of the output symbol table is returned in the **JSYTLEN** field in the table header. However, the value of the **JSYTLEN** field does not necessarily represent the minimum size required for the table, because the output symbol table created by the service can have unused space inside. The real size of meaningful data inside the output table is returned in the **JSYMSRCM** field. If the size of the output area provided by the caller is not sufficient for the output symbol table, the reason code field **JSYMREAS** will be set to **JSYMSTRE** and the **JSYMSRCM** field will contain the recommended size of the output area.

Field Name**Description****JSYMOUTA**

Pointer to the caller-provided output area.

JSYMOUTS

Size of the caller-provided output area.

JSYMRETN

Service return code.

JSYMREAS

Service reason code.

JSYMSRCM

Recommended size of the output area.

JSYMERAD

If the service returns an error, this field is a pointer to the approximate location in the input data where the error was detected.

JSYMSZE1

Length of version 1 of the parameter list (**JSYMPRM**).

JSYMSIZE

Length of the current version of the parameter list (**JSYMPRM**).

Symbol table (JSYTABLE)

The JES symbol table (**JSYTABLE**) contains information about JES symbols and their names. The symbol table consists of a table header that is mapped by the **JSYTABLE** data structure, and zero or more symbol entries that are mapped by the **JSYENTRY** data structure.

Field Name**Description**

JSYTEYE

Eyecatcher - JSYT.

JSYTLEN

The total length of the table, which includes the table header, symbol entries and space for symbol values. The **JSYTLEN** field indicates the distance between the first byte of table header and the first byte which follows the table. Note that if the table has unused space (between symbol values, for example), the unused space is accounted for by the **JSYTLEN** field.

JSYTVER

Version of the table.

Field Name**Description****JSYTVER1**

Version 1 of the table.

JSYTENT1

Offset from the beginning of the table to the first entry.

JSYTENT#

Number of entries in the table.

JSYTENTS

Size of each entry.

Symbol entry (JSYENTRY)

A symbol entry in the symbol table is mapped by the **JSYENTRY** data structure:

Field Name**Description****JSYENAME**

Symbol name.

JSYEVALO

Offset from the beginning of the table header (**JSYTABLE**) to the symbol value.

JSYEVALS

Size of the symbol value.

Chapter 5. Internal reader facility

The internal reader facility is a logical device similar to a card reader that allows you to submit jobs to JES. You can also read job streams from tape, disk or any QSAM-supported device through the internal reader to JES by using the procedure named RDR. Using the internal reader facility, you can submit jobs from time-sharing logons, started tasks, or other jobs. The ability to submit jobs from currently running jobs or tasks is especially powerful. This ability gives the programmer the flexibility to have a job that reaches a point successfully to submit another job for execution.

Defining the internal reader facility

In JES2, define the attributes of the internal reader facility with the INTRDR statement.

There are three types of internal readers:

- TSO logons are submitted by use of TSOINRDR. TSUINRDR and TSOINRDR are used interchangeably.
- Started tasks are submitted by use of STCINRDR.
- Batch jobs are submitted by use of INTRDR.

In JES3, internal readers are dynamically managed by the JES3 global and are always available for use.

If BATCH=NO is specified, you cannot use internal readers for batch jobs. However, you can still submit batch jobs through real (local) card readers, RJE, NJE, or spool offload.

Using the internal reader facility

There are four methods of using the internal reader facility. These methods are:

- Using a special external writer called INTRDR to submit a job from input in a batch job stream.
- Dynamically allocating the internal reader from your program.
- Using the IBM-supplied RDR procedure from either a batch job stream or the operator's console to read the job from a QSAM-supported device.
- Using the TSO/E SUBMIT command to pass a job stream to the internal reader facility. For more details about the TSO/E SUBMIT command, see *z/OS TSO/E Command Reference*.

Note: The user portion of the job correlator can be set using the UJOB CORR JCL keyword on the JOB card. For more information on the UJOB CORR keyword, refer to *z/OS MVS JCL Reference*. A user portion that is set by the UJOB CORR keyword will be overridden by the value that is set in the SYS_CORR_USRDATA symbol. The user portion can also be set in installation exits 2 and 52 for JOB JCL statement scan, and in exits 20 and 50 for end of job input. A user portion that is set in an installation exit will override any value that is specified on the UJOB CORR keyword or in the SYS_CORR_USRDATA symbol. For more information on installation exits, refer to *z/OS JES2 Installation Exits*.

Submitting to the internal reader from jobs or tasks

Figure 1 shows a step from a job (or task) that submits a job to the internal reader.

```
⋮
//STEP9 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=Z
//SYSUT2 DD SYSOUT=(A,INTRDR)
//SYSIN DD DUMMY
//SYSUT1 DD DATA,DLM=XX
//MYJOB1 JOB ACCT,VAZQUEZ,CLASS=A
//STEP1 EXEC PGM=CRUSHER
//ERRORS DD SYSOUT=A
//INPUT DD DSN=JES2.INIT.TUNE,DISP=SHR
//OUTPUT DD DSN=SMALL.BOOK,DISP=SHR
XX
//STEP10 EXEC ...
⋮
```

Figure 1. Submitting a Job to the Internal Reader

Step 9 writes the JCL that follows the STEP9 SYSUT1 card (up to the XX which acts as a delimiter) to a SYSOUT data set used as input to the INTRDR program.

If the ACB interface was used to open the internal reader, you can use the ENDREQ macro to complete the submission of jobs. For more information about coding the ENDREQ macro, see *z/OS DFSMS Macro Instructions for Data Sets* and *z/OS V2R1.0 Communications Server: SNA Programming*. For more information about JES control statement processing, see “JES control statements that affect the internal reader” on page 50.

Dynamically allocating the internal reader

You can allocate SYSOUT data sets to the special external writer, INTRDR, just as you would any other external writer. For example, your program can issue an SVC 99 (for details on SVC 99, see *z/OS MVS Programming: Assembler Services Guide*) and write JCL-images directly to the internal reader.

The following text units are required on the dynamic allocation request to allocate an internal reader:

- DALSYSOU - to indicate that this is a SYSOUT data set and the default MSGCLASS for jobs that are submitted through this internal reader. If '*' is specified, the MSGCLASS is the same as the MSGCLASS, job or TSO/E logon that allocated the internal reader.
- DALSPGNM - you must specify "INTRDR" to indicate that an internal reader is being allocated.
- DALDDNAM or DALRTDDN - specifies the DD name to associate with the internal reader or to request that the system assign a DD name.
- DALSSREQ or DALUASSR - optionally specify the name of the subsystem that the internal reader should be associated with. The name must be that of an active JES2 subsystem on this member. To use DALSSREQ, the caller must be APF authorized.

Note: If DALSSREQ or DALUASSR is specified, the address space that allocates the internal reader does not have to be associated with the JES2 that is specified. The allocating address space can be associated with the master address space (such as a started task running SUB=MSTR) or running under another JES subsystem (such as a job associated with the primary subsystem allocating an internal reader

on a secondary subsystem). However, having an internal reader allocated will prevent the owning JES2 from shutting down. IBM recommends that applications using DALSSREQ or DALUASSR not keep the internal reader allocated for an extended period of time or have a mechanism to request that the internal reader be unallocated. This prevents the internal reader allocation from impacting the starting and stopping of JES2 subsystems.

Passing JCL symbols to the submitted job

When using the internal reader facility, the submitting job can be used to pass JCL symbols to the submitted job. These JCL symbols can be used in the JCL of the submitted job in the same way that JCL symbols created by a SET JCL statement are used. To pass JCL symbols to the submitted job using the internal reader, the list of symbols to pass must be defined for this internal reader using two methods:

1. Use the SYMBOLS= keyword on the DD statement that defines the internal reader (for static allocation). Refer to *z/OS MVS JCL Reference* for details.
2. Use the DALSYML text unit during dynamic allocation of the internal reader. Refer to *z/OS MVS Programming: Authorized Assembler Services Guide* for details.

The following symbols can be passed as JCL symbols to the submitted job:

- JCL symbols that were previously made available to the job by an EXPORT JCL statement.
- JES symbols that were dynamically created by the JES Symbol Service (IAZSYMBL). Refer to Chapter 4, “JES Symbol Service (IAZSYMBL),” on page 33 for details.

If dynamically-created JES symbols are passed by the submitting job, they must conform to the limitations of JCL symbols; refer to *z/OS MVS JCL Reference* for details. The special value SYMBOLS=* passes all symbols that are available to the current task and that conform to JCL limitations to the submitted job, which includes all JCL symbols and all usable JES symbols.

The list of JCL symbols to be passed by the internal reader specifies symbol names, but not their values. Symbol values are captured by the internal reader when the job is submitted. Applications can set or change symbol values before submitting a job so that different jobs submitted through the same internal reader will have the same set of symbols but different values.

Requesting job notification

The internal reader facility can be used to request job notification for a job that is being submitted:

- Job notification is requested by defining the SYS_JOB_NOTIFY symbol before submitting a job using the internal reader.
- When the job is no longer eligible for execution, JES sends job completion notification by ENF 78. Refer to *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for ENF 78 information.
- ENF 78 includes job identification information and the value of the SYS_JOB_NOTIFY symbol.
- Applications can use job notification to track submitted jobs.

Assigning the user portion of the job correlator

The internal reader facility can be used to assign the user portion of the job correlator:

- The job correlator (JOB CORR parameter) can be used to limit the volume of processing that is required to control all batch jobs, STCs and TSUs.
- The job correlator value consists of a system portion and a user portion.
- The user portion of a job correlator can be set by assigning a value to the **SYS_CORR_USRDATA** symbol before submitting the job through the internal reader. Refer to “JES system symbols” on page 34 for details on using the **SYS_CORR_USRDATA** symbol. The user portion can also be set using the UJOB CORR parameter (see *z/OS MVS JCL Reference*) or using JES2 installation exits 2 and 52 for JOB JCL statement scan, and exits 20 and 50 for end of job input (see *z/OS JES2 Installation Exits*).

For details on using the JOB CORR parameter, refer to *z/OS JES2 Commands*.

Getting feedback

You can get feedback from the internal reader:

- The internal reader signals the result of the job submission by using special JES symbols.
- If the job was successfully submitted, the following special JES symbols are set:

SYS_CORR_LASTJOB

This value is set to the job correlator of the job that was just submitted, including the user portion if provided (refer to “Getting feedback”).

SYS_LASTJOBID

This value is set to the job identifier of the job that was just submitted.

- If job submission failed, these **SYS** symbols are set to empty values.

Time-sharing logon (TSO/E) and started task (STC) flow

Time-sharing logons and started system tasks appear to JES as two special forms of jobs that are received from designated internal readers. In JES2, these jobs are queued in special job classes (TSU and STC) and are assigned a MSGCLASS that is set during JES2 initialization (MSGCLASS parameter on the JOBCLASS(TSU) and JOBCLASS(STC) initialization statement). In JES3, the MSGCLASS for these jobs defaults to the MSGCLASS parameter specified on the CIPARM initialization statement. The two byte of PARMID= parameter on CIPARM statement is referenced by the INTPMID= parameter on the STANDARDS initialization statement.

The time-sharing message class (MSGCLASS parameter on the JOBCLASS(TSU) or CIPARM statement) becomes the output class for all dynamically allocated SYSOUT data sets for which a class is not specified, and becomes the MSGCLASS for all submitted jobs with no MSGCLASS parameter on the JOB statement. It is, therefore, not advisable to set MSGCLASS= to a SYSOUT class that specifies OUTDISP=PURGE. See the information on output disposition for SYSOUT data sets in *z/OS JES2 Initialization and Tuning Guide* for further information.

Time-sharing users can dynamically allocate data sets, dynamically deallocate them (spinoff), and print them at the time-sharing terminal (OUTPUT command). JES treats a file submitted by the TSO/Extensions interactive data transmission facility as an output data set.

Using the RDR procedure

Figure 2 shows the JCL procedure IBM supplies for you to use the internal reader to read jobs from tape, disk, or any QSAM-supported device.

```
//IEFPROC EXEC PGM=IEBEDIT
//SYSUT1 DD DDNAME=IEFRDER
//IEFRDER DD DSN=NULLFILE,DISP=OLD
//SYSUT2 DD SYSOUT=(A,INTRDR)
//SYSPRINT DD SYSOUT=A
//SYSIN DD DUMMY
```

Figure 2. The RDR Procedure

Examples of using the RDR procedure

The operator can invoke the RDR procedure to read:

- A job stream from the second file of a tape named JOBTAP on device 180:
S RDR,180,JOBTAP,LABEL=2,DSN=JOBS
- A job stream from a cataloged library of jobs:
S RDR,3330,DSN=PRODUCTN(PAYROLL)
- A job stream starting with a specific job on a tape named JOBTAP, the operator must submit a job to JES2 similar to:

```
//READJOBx JOB ...
// EXEC RDR
//IEFRDER DD DSN=JOBS,VOL=SER=JOBTAP,
// UNIT=3400,DISP=OLD
//SYSIN DD *
EDIT START=JOBx
/*
```

By using conditional JCL, you can cause internal readers to start only under specific conditions. You can then form a dependent job or set of jobs that execute (without operator intervention) only when a master job executes in a manner you want.

For example, to submit BADNEWS only if GOODNEWS does not complete successfully, specify the following:

```
//STEPHEN IF (RC = 0)THEN
/*
//GOODNEWS EXEC PGM=IEBGENER
//SYSPRINT DD DUMMY
//SYSIN DD DUMMY
//SYSUT1 DD JOBS(JOBA)
//SYSUT2 DD SYSOUT=(A,INTRDR)
/*
//STEPELSE ELSE
//BADNEWS EXEC PGM=IEBGENER
//SYSPRINT DD DUMMY
//SYSIN DD DUMMY
//SYSUT1 DD JOBS(JOBB)
//SYSUT2 DD SYSOUT=(A,INTRDR)
/*
//STEPEND ENDIF
```

User-written procedures and programs can further exploit the internal reader facility to select particular jobs, to generate special job streams, and to allow operator submission of production job streams.

JES control statements that affect the internal reader

The following JES control statements affect the way in which the internal reader handles the input stream it receives:

- /*EOF - ends the current job in the data set and makes it eligible for immediate processing.
- /*DEL - deletes the job in the data set and schedules it for immediate SYSOUT processing. This statement deletes the current job in the job stream. If there is no job in the data set, this statement has no effect. The SYSOUT consists of any JCL submitted, followed by a message indicating that the job was deleted before execution.
- /*SCAN - causes the job to be scanned for JCL errors, but not executed. (The same processing occurs if TYPRUN=SCAN appears on the JOB statement.)
- /*PURGE - deletes the job in the data set and schedules it for purge processing. If no job is in the data set, this statement deletes the previous job in the job stream. No output is produced for this job. This is for JES2 only because JES3 does not recognize /*PURGE as a control statement.

Performance considerations for JES internal reader

The following performance considerations affect the performance of internal reader in a JES subsystem.

Use of unblocked records for SYSIN and SYSOUT data sets

You should not block SYSIN and SYSOUT data sets because the SAM (sequential access method) compatibility interface will increase overhead by unnecessarily deblocking and blocking data sets.

Held internal readers in JES2

JES2 treats all internal readers as a single facility, therefore holding one internal reader places all internal readers in hold. This is particularly troublesome when the central operator holds the internal readers and TSO/E users want to submit jobs. You can avoid this problem by:

1. Assigning a specific job class for all jobs submitted through a particular internal reader. Instead of holding the internal reader, you can hold the class by using either a JES2 initialization statement or a JES2 \$T JOBCLASS(x),QHELD=YES command.
2. Use the TYPRUN=HOLD parameter or TYPRUN=JCLHOLD parameter on the JOB statement.
3. Submitting the job through an internal reader and individually hold it with the JES2 \$H J command.

Record length of SYSIN data sets

Jobs can include input data in SYSIN data sets. In JES2, the maximum length of a record written to the internal reader is 32760 bytes. In JES3, the maximum length is the installation defined buffer size. These can be processed locally or sent to other nodes through NJE. Some NJE nodes do not support SYSIN records that are greater than 254 bytes (in JES2) or 80 characters (in JES3) in length. When data is sent to one of these nodes, the SYSIN records will be truncated to 254 bytes (in JES2) or 80 characters (in JES3). Before attempting to send long SYSIN records to a node, ensure that the node and any intermediate node support long SYSIN records (for example, by sending a test).

SYSIN record formats

JES sets the record format for SYSIN data sets based on the data written to them. If all records that are written are of the same length (before any blank truncation), the record format (RECFM) will be set to fixed (F). If the records vary in length, the record format will be set to V. If carriage control is detected in the SYSIN stream, the record format will be updated to FM, FA, VB or VA depending on whether the records vary in length or not and whether the carriage control is ASA or Machine. If both ASA and Machine carriage control are detected, the record format will be set in the RECFM.

Chapter 6. The External Writer

This chapter documents Intended Programming Interface and Associated Guidance Information.

An external writer allows an installation to perform output processing for data sets not eligible for processing by the primary job entry subsystem (JES2 or JES3). For example, an external writer processes data sets going to printers, plotters, diskettes or data sets that are to be stored on a device not supported by JES.

There are two major parts of the IBM-supplied external writer that can be modified or replaced by an installation: the output writing routine and the output separator routine.

This chapter describes:

- Overview of the IBM-supplied external writer
- How to set up and start the IBM-supplied external writer
- How the IBM-supplied output writing routine works
- How to add your own output writing routine
- How the IBM-supplied output separator routine works
- How to add your own output separator routine

See *z/OS MVS Using the Functional Subsystem Interface* for information in subsystem interface (SSI) function code 1 on how to write your own external writer.

Overview of the IBM-Supplied External Writer

IBM supplies an external writer, which contains two routines that can be modified or replaced by the installation:

- The output writing routine. This routine processes data sets.
- The output separator routine. This routine writes separator records within a continuous listing or deck to help the operator separate one job's output from another's.

Characteristics of the IBM-Supplied External Writer

The external writer has the following features:

- It runs as a started task, in its own address space, in 24-bit addressing mode.
- It processes only those data sets that meet its selection criteria. You set some of these in a cataloged procedure (see “The External Writer Cataloged Procedure” on page 54) but you can override them with the START and/or the MODIFY operator commands (see *z/OS MVS System Commands* for the general description of these commands).

If you do not specify data set selection criteria, the external writer will be used to process **all** the output in the system.

- It removes data sets from the JES spool. That is, it dynamically allocates the data sets, reads them, writes them to output devices, and then dynamically deallocates them. A large format spool data set can be shared.
- It supports basic format, extended format and large format sequential data sets on DASD.

- It supports checkpoint and spool on 3390 Model 9 and Enterprise Storage Server (ESS).
- It accesses data sets according to one or more of the following criteria:
 - Output class
 - Job ID
 - Forms specification
 - Destination (LOCAL, or remote workstation name)
 - The name of your output writing routine

The usual technique however, for setting data set selection criteria, is to build a list of eligible SYSOUT classes for the devices that will use the external writer.

If no class is specified, then we will not start selecting any output and will wait for a MODIFY command to be issued. For example, if you have the following in process:

```
//IEFPROC EXEC PGM=IASXWR00,PARM='P',REGION=20K
```

No class is specified. An S XWTR will cause the external writer to be started, but idle. Then, a F XWTR,F=ABC may be specified to have the external writer select by form type. If another F XWTR,C= is issued, then everything on the output queue will be selected because a null class is specified.

How to Set Up and Start the External Writer

For an external writer to work in a JES environment, it must be defined to the system in a cataloged procedure residing in SYS1.PROCLIB; and it must be started by a START command, either from the system console or from within a program.

There are two ways to set up the criteria by which the external writer selects output. The PARM= field (described below) allows the specification of SYSOUT classes to select. Other criteria (such as forms, destination, writer name, jobid and also SYSOUT class) must be specified using the MVS MODIFY command. See *z/OS MVS System Commands* for a complete description of the modify command as it applies to the external writer.

The External Writer Cataloged Procedure

The IBM-supplied external writer is defined and invoked by the cataloged procedure named XWTR, which can serve as the base or a model for a procedure you would write for your own output writer.

XWTR contains one step and consists of two JCL statements:

- The EXEC statement specifies the name of the external writer program to be executed.
- The DD statement, which **must** be named IEFRDER as shown below, defines the output data set.

Following is the actual XWTR procedure:

```
//IEFPROC EXEC PGM=IASXWR00,REGION=20K, X
// PARM='PA,, ' X
//IEFRDER DD UNIT=TAPE,VOLUME=(,,35), X
// DSNNAME=SYSOUT,DISP=(NEW,KEEP), X
// DCB=(BLKSIZE=133,LRECL=133,BUFL=133, X
// BUFNO=2,RECFM=FM) X
```

The EXEC Statement

The generalized format for the EXEC statement is:

```
//IEFPROC EXEC PGM=IASXWR00[,REGION=nnnnnK,]  
//          [PARAM='cxxxxxxxx[,seprname]]'  
//          [PARAM='cxxxxxxxx[,seprname][,CERTIFY=Y|N]]'
```

The stepname must be IEFPROC, as shown. The parameter requirements are as follows:

PGM=IASXWR00

The name of the external writer load module. It must be IASXWR00, as shown.

REGION=nnnnnK

This parameter specifies the region size for the external writer program. The value nnnnn is a 1- to 5-digit number that is multiplied by 1K (1024 bytes) to designate the region size. The region size can vary according to the size of buffers and the size of your output writing routine. Insufficient region size will cause the external writer to abend.

```
[PARAM='cxxxxxxxx[,seprname][,CERTIFY=Y|N]]'  
[PARAM='cxxxxxxxx[,seprname]]'
```

Specifies output writer routine characteristics.

- c** An alphabetic character, either **P** (for printer) or **C** (for card punch), that specifies the control characters for the class of output the output writing routine will process.

xxxxxxxx

From one to eight (no padding required) single-character SYSOUT class names. These characters specify the classes the output writing routine will process and establish the priority for those classes, with the highest priority at the high-order (leftmost) end of the character string.

Note: If the START command includes class name parameters, they override all of the class names coded here. If you do not code class names on the procedure EXEC statement or the START command, then the external writer will wait for a MODIFY command from the operator before processing any output.

seprname

This is the name of the output separator routine to run with the output writing routine. If you omit this subparameter, no output separator pages are produced.

IEFSD094 is the IBM-supplied output separator routine. If you write your own output separator routine, and name it here, put it in SYS1.LINKLIB (or a library concatenated to LINKLIB through a LNKLSTxx member of parmlib).

Note: Do not use *CERTIFY* as the name of your output separator routine. For your separator routine to be invoked, you must either code its name on this subparameter or name it in the parameters on the START command.

CERTIFY=Y|N

Indicates whether (Y) or not (N) you want to ensure that all data is safely written to the output device. CERTIFY=Y indicates that data will **not** be lost even if the external writer abnormally terminates, such as would occur if the output data set becomes full.

Note:

1. The use of the certify function increases processing overhead and decreases efficient space utilization by the output data set.
2. If you write your own output writing routine and require the CERTIFY function, you need to modify your code to be similar to that supplied by IBM. Refer to “Using the CERTIFY Function” within section “Programming Considerations for the Output Writing Routine” on page 61 for further details.
3. IBM cannot guarantee data set integrity when writing multiple data sets to a PDS/PDSE member even if you specify CERTIFY=Y. See “Functions of the output writer routine” on page 58 for further details.

The DD Statement

In this statement, you must define the output data set that the external writer will use. The generalized format for the DD statement is:

```
//IEFRDER DD UNIT=device,LABEL=(,type), X
//          VOLUME=(,,volcount),DSNAME=anyname, X
//          DISP=(NEW,KEEP),DCB=(list of attributes), X
//          UCS=(code[,FOLD][,VERIFY]), X
//          FCB=(image-id[,ALIGN][,VERIFY])
```

The ddname must be IEFRDER, as shown. The parameter requirements are as follows:

UNIT=device

This specifies the printer, tape, card punch, or DASD device on which the output data set is to be written.

LABEL=type

This describes a data set label, if one is needed (for tape data sets only). If this parameter is omitted, a standard tape label is used.

VOLUME=(,,volcount)

Needed for tape data sets only, this parameter limits the number of tape volumes that this external writer can use during its entire operation.

DSNAME=anyname

This specifies a name for the output data set, so later steps in the procedure can refer to it. The data set name is required for the disposition of KEEP.

DISP=(NEW,KEEP)

The disposition of KEEP prevents deletion of the data set (tape and DASD only) at the end of the job step.

DCB=(list of attributes)

The DCB parameter specifies the characteristics of the output data set and the buffers. The BLKSIZE and LRECL subparameters are always required. The BUFL value, if you do not code it, is calculated from the BLKSIZE value. Other subparameter fields may be coded as needed; if they are not, the defaults are the QSAM default attributes. These are:

BUFNO—

Three buffers for the 2540 punch; two buffers for all other devices.

RECFM—

U-format, with no control characters.

TRTCH—

Odd parity, no data conversion, and no translation.

DEN—
Lowest density.

OPTCD—
Printer data checks are suppressed, and “select translate table” characters are printed as data. The IBM external writer does not support OPTCD=J, a printer dependent specification.

UCS=(code[,FOLD][,VERIFY])

This specifies the code for a universal character set (UCS) image to be loaded into the UCS buffer.

FOLD causes bits 0 and 1 to be ignored when comparing characters between the UCS and print line buffers, thereby allowing lowercase alphabetic characters to be printed (in uppercase) by an uppercase print chain or train.

VERIFY causes the specified UCS image to be printed for verification by the operator.

The UCS parameter is optional, and is valid only when the output device is a 1403, a 3211, or a 3203-5 printer.

FCB=(image-id[,ALIGN][,VERIFY])

This causes the specified forms control buffer (FCB) image to be loaded into the FCB. ALIGN and VERIFY are optional subparameters that allow the operator to align forms. In addition, VERIFY causes the specified FCB image to be printed for visual verification. The FCB parameter is valid only for a 3203-5, 3211, or 3800 printer; otherwise, it is ignored.

See *z/OS MVS JCL Reference* for more information on the parameters mentioned here.

Special Printer Output Considerations: To process output jobs that require special chains for printing, you should have specific classes for each different print chain. You can specify the desired chain in your output writer procedure, and when that output writer is started, the chain will be loaded automatically. (Printers used with special chains should be named with esoteric group names as defined at system installation.) See *z/OS HCD Planning* for information on the eligible device table.

Following is an example of the JCL needed to define a special print chain in a cataloged procedure for an external writer.

```
//IEFPROC EXEC PGM=IASXWRO0,REGION=20K,PARM='PDEG,IEFSD094'  
//IEFRDER DD UNIT=SYSPR,DSNAME=SYSOUT,FCB=(STD2,ALIGN), X  
// UCS=P11,DISP=(,KEEP), X  
// DCB=(BLKSIZE=133,LRECL=133,BUFL=133, X  
// BUFNO=2,RECFM=FM)
```

In this example, the UCS DD parameter requests the print chain alias for data sets in the SYSOUT classes D, E, and G.

If the output device is a 3211 or a 3203-5, a UCS or FCB image can be loaded dynamically between the printing of data sets. Therefore, you can specify a mixture of data sets using different images in a single output class for this device. This will probably require mounting trains and changing forms, however, so it might not be desirable.

When the output device is a 1403 or 3800, the UCS image or 3800 attributes are specified at START XWTR time; they cannot be changed until the writer is stopped. Therefore, all data sets within an output class must be printed using the same train.

The FCB image is ignored when the 1403 printer is the output device.

External writer output to an IBM 3800 Printing Subsystem can also make use of the CHARS, COPIES, FLASH, and MODIFY JCL parameters on the DD statement. For information about using these parameters, see *3800 Printing Subsystem Programmer's Guide*. The coding rules and defaults are documented in *z/OS MVS JCL Reference*.

How the IBM-Supplied Output Writer Routine Works

As stated previously, there are two main pieces to the external writer that can be modified or replaced by the installation: The output writer routine and the output separator routine. When the external writer is called, it gets control in module IASXWR00. The main function of IASXWR00 is to initialize a 7-word parameter list for the use of the other external writer routines.

The parameter list contains information about the output device and DCB addresses for each data set. The input data set to the external writer was a job's output data set which met the criteria for processing by the external writer. This input data set is not yet open.

The format of the external writer parameter list is as follows:

Table 2. External Writer Parameter List

Byte	Meaning										
0	Used to describe the type of output device <table border="1"> <thead> <tr> <th>Bit</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>011.</td> <td>2540 punch unit</td> </tr> <tr> <td>001.</td> <td>1403, 3203-5, 3211, or 3800 printer device</td> </tr> <tr> <td>010.</td> <td>tape device with punch-destined output</td> </tr> <tr> <td>000.</td> <td>tape device with printer-destined output</td> </tr> </tbody> </table>	Bit	Meaning	011.	2540 punch unit	001.	1403, 3203-5, 3211, or 3800 printer device	010.	tape device with punch-destined output	000.	tape device with printer-destined output
Bit	Meaning										
011.	2540 punch unit										
001.	1403, 3203-5, 3211, or 3800 printer device										
010.	tape device with punch-destined output										
000.	tape device with printer-destined output										
1-3	Reserved for IBM use										
4-7	Address of the DCB for the opened output data set where the external writer will put the input records										
8-11	Address of the DCB for the input data set, from which the external writer will obtain logical records.										
12-15	Address of the cancel ECB for writer routine subtask (IEFSD087 or user-written writer routine).										

The switches indicated by the three high-order bit settings in byte 0 can be used in translating control character information from the input records to the form required by the output device.

Functions of the output writer routine

When you specify IEFSD087 as the output writer on the SYSOUT= parameter of the DD statement, or when you leave this parameter blank, control will pass to the IBM-supplied output writing routine in module IEFSD087, and it:

- Issues an OPEN (J type) for the input data set previously taken from the JES spool.
- Provides its own SYNAD error-handling routine on behalf of both the input and output data sets. See *z/OS DFSMSdfp Advanced Services* for more information about OPEN-J and SYNAD.
- Reads the input data set, using the locate mode of the GET macro.
- Calls a subroutine to handle ANSI and machine control character differences and to handle conversions between the input records and the output data set.
- Calls a routine to write records to the output data set, using the locate mode of the PUT macro.
- Closes the input data set after it has been read.
- Provides accounting support by updating fields in the SMF type 6 record for the input data set.
- Frees the buffer associated with the input data set (by issuing the FREEPOOL macro).
- Returns control to the main logic control module of the external writer, IASXWR00, using the RETURN macro and setting a return code.

Note: When writing multiple data sets to a PDS/PDSE member, if the 'end-of-output-data-set' condition arises, the output external writer cannot guarantee (even if you set CERTIFY=Y as a startup parameter) that all records will be saved. The current data set is saved, but all records written up to that point cannot be recovered in the case of writing to a PDS/PDSE member.

The following diagram shows the general flow of the IBM-supplied output writing routine:

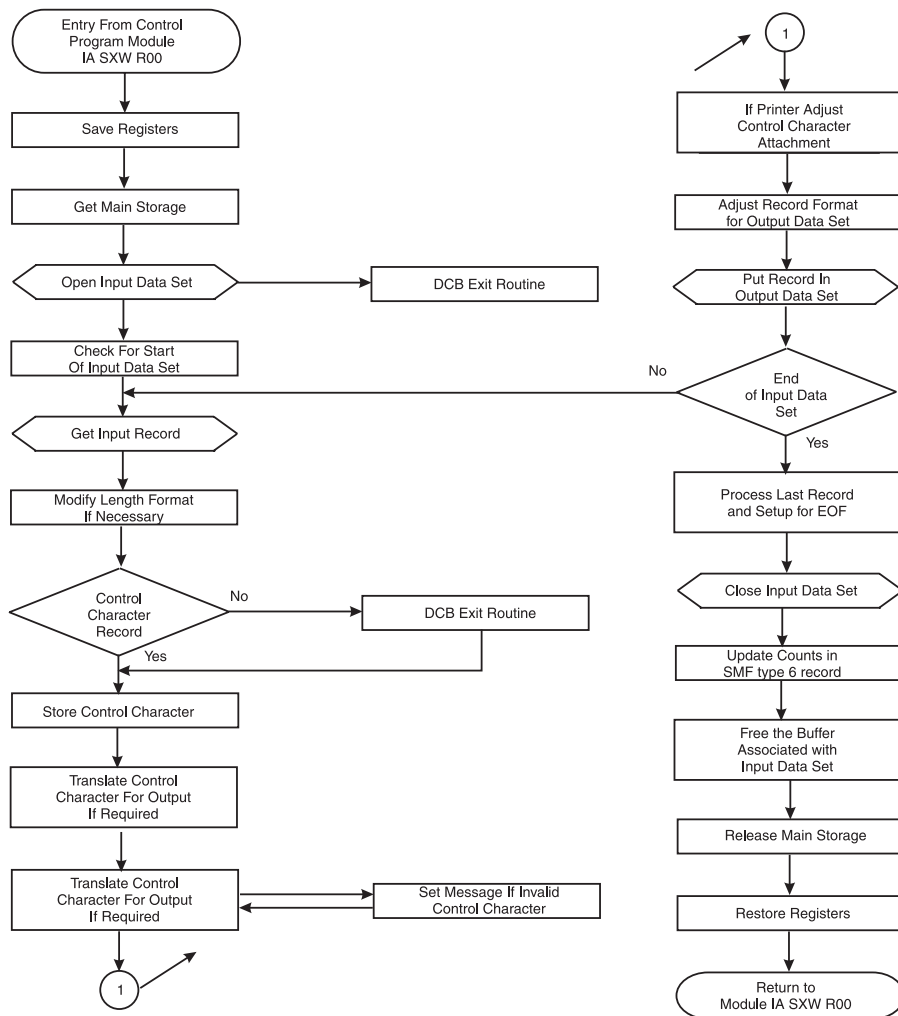


Figure 3. General flow of IBM's output writing routine

How to Add Your Own Output Writing Routine

If you want to add your own output writing routine to the external writer, consider whether your routine will be needed more often than the IBM-supplied routine. If your routine will be invoked to write **most** of the external writer's output, you might want to replace the IBM-supplied with your own routine, so that your routine will be called by default. You can retain the IBM-supplied routine by renaming it to an alias.

Replacing the IBM-Supplied Routine

You can replace the IBM-supplied routine with your routine by:

- Renaming the IBM-supplied routine (IEFSD087) to an alias.
- Naming your routine IEFSD087
- Installing your routine in SYS1.LINKLIB

The external writer will call your routine by default. You can request the IBM-supplied routine by coding its alias on the SYSOUT= parameter. For example:

```
//MYDATA DD SYSOUT=(H,IBMWRITR)
```

where IBMWRITR is the alias you've given to the IBM-supplied routine.

Do not code STDWTR as a writer name. STDWTR and INTRDR are and are reserved for JES used as a parameters in the MVS operator's MODIFY command.

In a JES3 system, do not code NJERDR as a writer name. NJERDR is reserved for JES3.

Using Your Routine for Only Certain Jobs

If you plan to have your routine invoked for only certain jobs, give it a unique name, and install it in SYS1.LINKLIB or an authorized library concatenated to LINKLIB through a LNKSTxx member of SYS1.PARMLIB. To have your routine invoked, the end-user must specify its name on the SYSOUT= parameter of the job's DD statement. When your routine is not specified, the system calls the IBM-supplied routine by default.

Coding Conventions for the Output Writing Routine

In order for your output writing routine to work in the external writer, it must observe the following conventions:

- If you put your routine in LPA so that you only use one copy for all external writers, then it must be reentrant. If you use a STEPLIB and each writer can have its own copy, then the code need not be reentrant.
- The routine must use standard entry and exit linkage, saving and restoring its caller's registers.

At entry, register 1 points to the parameter list and register 13 points to an 18-word save area.

- The routine must issue an OPEN-J macro to open the desired input data set. The data set to which the output writing routine will write records is opened before the routine is loaded.
- The routine must use the GETMAIN and FREEMAIN macros, or the STORAGE macro, to acquire and release any necessary storage.
- The routine must return control to its caller through the RETURN macro, in the same addressing mode it was called in. It must put a return code in register 15:
 - A return code of 0 indicates that the data set was processed successfully to the output device.
 - A return code of 8 indicates that the routine was unable to process the data set because of an output error.

Programming Considerations for the Output Writing Routine

In addition to the coding conventions, consider the following when writing your own output writing routine:

- **Obtaining Storage for Work Areas:** Using the GETMAIN or STORAGE macro, the output writing routine should obtain storage in which to set up switches and save record lengths and control characters.
- **Processing an Input Data Set:** To process a data set, the writing routine must get each record individually from the input data set, transform (if necessary) the record format and the control characters to conform to the output data set's requirements, and put the record in the output data set. Consider each of these tasks individually:
 1. Before the routine actually obtains a record from an input data set, the routine must provide a way to handle the two forms of record control character that are allowed in an output data set, if the output device is a printer.

Most printers are designed so that, if the output data set records contain machine control characters, a record (line) is printed before the effect of its control character is considered. However, if ANSI control characters are used in the output records, the control character's effect is considered before the printer prints the line. Thus, if the input data sets do not all have the same type of control characters, you will need to avoid overprinting the last line of one data set with the first line of the next.

When the input records have machine control characters and the output records are to have ANSI control characters, the standard (IBM-supplied) output writing routine produces a control character that indicates one line should be skipped before printing the first line of output data.

When the input records have ANSI control characters and the output records are to have machine control characters, the standard writing routine prints a line of blanks before printing the first actual output data set record. Following this line of blanks, the printer generates a one-line space before printing the first record.

Depending upon the characteristics of the printers in your installation, you will probably want your output writer to perform some kind of "printer initialization" like that outlined here.

2. After the output writing routine has properly opened the input data set and has completed any necessary printer initialization, it must obtain records from the input data set.

The standard output writing routine uses the locate mode of the GET macro. If you use this macro, you will need to check the MACRF field of the input data set's DCB to see if GET in locate mode is allowed. If not, you can override the MACRF parameter on the GET macro itself. See *z/OS DFSMSdfp Advanced Services* for information on coding and using all the QSAM macros.

In a JES2 system, the padding is done automatically.

In a JES3 system, if blank truncation is in effect (TRUNC=YES specified on the BUFFER initialization statement or the SYSOUT initialization statement), then fixed format records will not be padded with blanks. You need to do the padding manually.

3. Having obtained a record from the input data set, the output writing routine must now make sure that its format and control character are compatible with the requirements of the output data set.

Because the output data set is already opened when the output writing routine is entered, your routine will have to adhere to the established conventions.

The standard output writing routine uses the PUT macro in the locate mode to write records to the output data set. For fixed-length output, it obtains the record length for the output data set from the DCBLRECL field of the DCB.

If an input record is longer than the length specified for the output records, the standard output writing routine truncates the input record from the right.

If an input record is shorter than the length specified for the output records, the standard output writer left-justifies the input record and pads the field with blanks.

When the output record length is variable and the input record length is fixed, the standard output writer adds control character information (if necessary) and variable record control information to the output record. Control character information is one byte, and record control information is four bytes long. Both additions are at the high-order end of the record.

If the output record is not at least 18 bytes long, the standard output writer pads it on the right with blanks. The standard output writer also adjusts the length of the output record to match the length of the output buffer.

4. When the output writing routine has successfully adjusted the input and output records, it can read the input data set until end of data. At that point, you need to consider another aspect of input data set processing: what is going to happen to the last input record?

The standard output writer handles output to either card punch or printer, as required; your routine could also send output to an intermediate tape or DASD device. Depending upon the kind of device, the last few records obtained from the input data set will receive different treatment.

It might happen that all the records from a given data set are not available on the output device until the output of records from the next data set is started, or until the output data set is closed. However, if you specify `CERTIFY=Y` as a start-up parameter, the records can be made available on the output device after all the input records have been written by the output writing routine. When the output data set is closed, the standard output writer automatically puts out the last record of its last input data set.

For Punch Output: When a card punch is the output device, the last three output cards could still be in the machine when the input data set is closed.

To put out these three records with the rest of the data set, and with no breaks, the standard output writer provides for three blank records following the actual data set records.

For Printer Output: When a printer is the output device, the last record of the input data set is not normally put in the output data set at the time the input data set is closed.

To force out this last record, the standard output writer generates a blank record to follow the last record of the actual data set.

- **Using the CERTIFY Function:** The CERTIFY function can be specified as a parameter when starting the external writer. It guarantees that data will not be lost when an abend condition occurs. If the CERTIFY function is not enabled, it is possible that records written to the QSAM buffers, but not yet written to the output device, can be purged before they are recorded during an abend situation. This is possible, for example, when the output data set is not large enough to contain the data, resulting in an 'end-of-volume' (x37) abend.

The CERTIFY function flushes the buffers each time an input data set is processed, and makes that data available on the output device. However, while guaranteeing data integrity, this function might also increase processing overhead and decrease efficient output data set space utilization. Also, note that even with `CERTIFY=Y`, data integrity might be compromised in the case of writing multiple JES data sets to a partitioned dataset (PDS) or PDSE in an abend situation.

If you need to use the CERTIFY function, you need to modify your output writing routine to be similar to that supplied by IBM. Refer to module IEFSD087 for an example of how to enable the CERTIFY function. Basically, you must do the following:

1. In the external writer main task, force the number of buffers for the output data set to 1. That is, set `DCBBUFNO=1`.
2. In the output writing routine after all records are PUT to the data set, perform a second PUT to write a blank line (a 'transition record') after the data to separate the data sets.
3. In the output writing routine, commit the data to the output device.

- **Closing Input Data Set(s):** After the standard output writer finishes putting out the records of an input data set, it closes the data set before returning control to the calling module. All input data sets must be closed.

After closing the input data set, it is recommended that your output writer free the buffer pool associated with the input data set (by issuing the FREEPOOL macro).

- **Releasing Main Storage:** The output writing routine should release the storage it acquired, using the FREEMAIN or STORAGE macro, before returning to its caller.
- **Handling Errors:** The routine must put a return code into register 15 before returning to its caller using the RETURN macro.

The standard output writer sets a return code of 8 if it terminates because of an unrecoverable error on the output data set; otherwise, the return code is 0. The output writing routine must handle input errors itself.

How the IBM-Supplied Output Separator Routine Works

The second major part of the IBM-supplied external writer routine is the output separator routine. Any output processing to a punch or printer must include a means of separating one job from another within the continuous deck or listing. When you specify IEFSD094 on the PARM= parameter on the EXEC card of the external writer, the IBM-supplied output separator routine writes separation records to the output data set prior to the writing of each job's output.

You can modify this separator routine to suit your installation's needs, or you can create your own routine. IBM's version does the following:

- **For Punch-Destined Output:** The separator routine provides three specially-punched cards (deposited in stacker 1) prior to the punch card output of each job. Each of these cards is punched in the following format:

Columns 1 to 35

blanks

Columns 36 to 43

job name

Columns 44 to 45

blanks

Column 46

output classname

Columns 47 to 80

blanks

- **For Printer-Destined Output:** The IBM-supplied separator routine provides three specially-printed pages prior to printing the output of each job. Each of these separator pages is printed in the following format:

- Beginning at the channel 1 location (normally near the top of the page), the job name is printed in block characters over 12 consecutive lines. The first block character of the 8-character job name begins in column 11. Each block character is separated by 2 blank columns.
- The next two lines are blank.
- The output classname is printed in block characters covering the next 12 lines. This is a 1-character name, and the block character begins in column 35.
- The remaining lines, to the bottom of the page, are blank.
- In addition to the block characters, a full line of asterisks (*) is printed twice (that is, overprinted) across the folds of the paper. These lines are printed on

the fold *preceding* each of the three separator pages, and on the fold *following* the third page. This is to make it easy for the operator to separate the job output in a stack of printed pages.

To control the location of the lines of asterisks on the page, the IBM-supplied separator routine requires that a channel 9 punch be included (along with the channel 1 punch) on the carriage control tape or in the forms control buffer (FCB). The channel 9 punch should correspond to the bottom of the page. The printer registration should be offset to print the line of asterisks on the fold of the page.

The IBM-supplied separator routine makes no provision for the 3800 printing subsystem; if you use it on a 3800, the FCB must locate a channel 9 punch at least one-half inch from the paper perforation.

Separator Routine Parameter List: IBM's external writer provides its separator routine with a 7-word parameter list of necessary information. When the separator routine receives control, register 1 contains the address of the parameter list, which contains the following:

Table 3. Separator Routine Parameter List

Byte	Meaning										
0	Used to describe the type of output device										
	<table border="1"> <thead> <tr> <th>Bit</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>011.</td> <td>2540 punch unit</td> </tr> <tr> <td>001.</td> <td>1403, 3203-5, 3211, or 3800 printer device</td> </tr> <tr> <td>010.</td> <td>tape device with punch-destined output</td> </tr> <tr> <td>000.</td> <td>tape device with printer-destined output</td> </tr> </tbody> </table>	Bit	Meaning	011.	2540 punch unit	001.	1403, 3203-5, 3211, or 3800 printer device	010.	tape device with punch-destined output	000.	tape device with printer-destined output
Bit	Meaning										
011.	2540 punch unit										
001.	1403, 3203-5, 3211, or 3800 printer device										
010.	tape device with punch-destined output										
000.	tape device with printer-destined output										
1-3	Not used, but must be present										
4-7	Address of the DCB for the opened output data set where the external writer will put the input records										
8-11	Address of an 8-character field containing the job name										
12-15	Address of a 1-character field containing the output class name										
16-19	Address of the data set PROC name in the SSOB extension.										
20-23	Address of the data set STEP name in the SSOB extension.										
24-27	Address of the data set DD name in the SSOB extension.										

The parameter list points to a DCB; this DCB is established for the QSAM output data set, which is already open when the separator program receives control.

The address of the job name and the address of the output classname are provided in the parameter list so they can be used in the separation records the separator routine writes.

Output from the Separator Routine: A separator routine can write any kind of separation identification. IBM supplies a routine that constructs block characters. (See "Using the Block Character Routine" on page 66.) The separator routine can punch as many separator cards, or print as many separator pages, as necessary.

The output from the separator program must conform to the attributes of the output data set. To find out what these attributes are, examine the open output DCB pointed to by the parameter list. The attributes are:

- Record format (fixed, variable, or undefined length)

- Record length
- Type of carriage control characters (machine, ANSI, or none)

For printer-destined output, a separator routine can begin its separator records on the same page as the previous job output, or skip to any subsequent page. However, the separator routine should skip at least one line before writing any records because in some cases the printer is still positioned on the line last printed.

After completing the output of the separation records, the separator routine should write sufficient blank records to force out the last separation record. This also allows the error exit routine to obtain control if an uncorrectable output error occurs while writing the last record. One blank record, for printer-destined output, and three blank records, for punch-destined output, are sufficient to force out the last record.

How to Add Your Own Output Separator Routine: If you write your own separator routine, it must conform to the following requirements:

- The routine must have a unique name and this name must be specified in the PARM= parameter on the EXEC card for the external writer. If you do not specify the name of an output separator routine, no separator records are written.

Note: Do not use *CERTIFY* as the name of your output separator routine. Refer to “The EXEC Statement” on page 55 for details on specifying an output separator routine on the PARM= parameter.

- If you want to replace the standard IBM routine, then name your routine IEFSD094. This should reside in SYS1.LINKLIB or in a library concatenated to LINKLIB through a LNKLSxxx member of SYS1.PARMLIB.
- The routine must use standard entry and exit linkages, saving and restoring its caller's registers, and returning to its caller through the RETURN macro, with a return code in register 15.
- The routine must use the QSAM PUT macro in locate mode to write separation records to the output data set.
- The routine must use the GETMAIN and FREEMAIN macros, or the STORAGE macro, to obtain and release the storage required for work areas.
- The routine must establish its own synchronous error exit routine, and place the exit address in the DCBSYNAD field of the output DCB. The error routine will receive control during output writing in case of an uncorrectable I/O error; it must set a return code of 8 (binary) in register 15 to indicate an unrecoverable output error.

If the separator routine completes processing successfully, it must set a return code of 0 in register 15, before returning to its caller.

Note: The separator routine receives control in problem-program state, but with a protection key of 0. Therefore, the routine must ensure data protection during its execution.

Using the Block Character Routine: For printer-destined output, the separator routine can use an IBM-supplied routine to construct separation records in a block character format. This routine is a reentrant module named IEFSD095 that resides in the module library SYS1.AOSB0.

The block character routine constructs block letters (A to Z), block numbers (0 to 9), and a blank. The separator routine furnishes the desired character string and

the construction area. The block characters are constructed one line position at a time. Each complete character is contained in 12 lines and 12 columns; therefore, a block character area consists of 144 print positions. For each position, the routine provides either a space or the character itself.

The routine spaces 2 columns between each block character in the string. However, the routine does not enter blanks between or within the block characters. The separator routine must prepare the construction area with blanks or other desired background before entering the block character routine.

To invoke the IBM-supplied block character routine, the IBM-supplied separator routine executes the CALL macro with the entry point name of IEFSD095. Since the block characters are constructed one line position at a time, complete construction of a block character string requires 12 entries to the routine. Each time, the address of a 7-word parameter list is provided in register 1.

The parameter list contains the following:

Table 4. Block Character Routine Parameter List

Byte	Meaning
0-3	This fullword is the address of a field containing the desired character string in EBCDIC format.
4-7	This fullword is the address of a field containing the line count as a binary integer from 1 to 12. This represents the line position to be constructed on this call.
8-11	This word is the address of a construction area where the routine will build a line of the block character string. The required length in bytes of this construction area is $14n-2$, where n represents the number of characters in the string.
12-15	This word is the address of a fullword field containing, in binary, the number of characters in the string.
16-19	Address of the data set PROC name in the SSOB extension.
20-23	Address of the data set STEP name in the SSOB extension.
24-27	Address of the data set DD name in the SSOB extension.

Appendix. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (<http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome>).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact z/OS or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS V2R2 ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM® Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number

(for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element

The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element

The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In the example, if you include the FILE

keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

*** indicates an optional syntax element that is repeatable**

The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Policy for unsupported hardware

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

for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: (no navtitle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

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

Index

A

- accessibility 69
 - contact IBM 69
 - features 69
- assistive technologies 69

B

- block character routine 66

C

- Client/Server print interface 27
 - botain status 28
 - compare CTOKENs 27
 - create a CTOKEN 27
 - data set 28
 - access 28
 - security 29
 - event
 - listen for event 30
 - header page
 - identify a requestor 29
- configuration
 - configuration 45
 - defining 45
 - internal reader 45
- contact
 - z/OS 69

D

- Defining JES symbols 40
- DELETE and EXTRACT symbols 41

E

- external writer 53
 - block characters to separate jobs 66
 - characteristics 53
 - description 53
 - features of IBM-supplied version 53
 - for the 3800 Printing Subsystem 58
 - format and content 58
 - modifying for special print chains 57
 - overview 53
 - parameter list 58
 - selection criteria
 - data set 54
 - starting the external writer 54
 - using the 3800 Printing Subsystem 58
 - XWTR cataloged procedure 54, 57, 58

I

- IASXWR00 module 58
- IAZSYMBL 33
- IAZSYMBL environment 37
- IAZSYMBL input register
 - information 36
- IAZSYMBL macro 35
- IAZSYMBL output register
 - information 36
- IAZSYMBL return code information 36
- IAZSYMBL syntax 35
- IAZSYMDF macro 37
- IAZSYMDF parameter list (JSYMPARM) 38
- IAZSYMDF reason codes (JSYMREAS) 38
- IAZSYMDF requested operations (JSYMRQOP) 39
- IAZSYMDF restrictions 37
- IAZSYMDF return codes (JSYMRETN) 37
- IEFSD094 output separator routine 64
- IEFSD095 output separator routine 66
- input register information
 - IAZSYMBL 36
- Input symbol table (JSYMISYT) 41
- internal reader
 - assigning the user portion of the job correlator 47
 - example
 - allocation, dynamic 46
 - dynamic allocation 46
 - submitting from a task 46
 - submitting from job 46
 - getting feedback 48
 - maximum number of simultaneous job streams 45
 - passing JCL symbols 47
 - requesting job notification 47
 - using 45

J

- JES Symbol (IAZSYMBL) macro 35
- JES Symbol entry (JSYENTRY) 43
- JES symbol options (JSYMLVL) 41
- JES Symbol Service (IAZSYMBL) 33
- JES Symbol Service data definition (IAZSYMDF) macro 37
- JES Symbol table (JSYTABLE) 42
- JES symbols
 - defining 40
- JSYENTRY 43
- JSYMISYT 41
- JSYMLVL 41
- JSYMPARM 38
- JSYMREAS 38
- JSYMRQOP 39
- JSYTABLE 42

K

- keyboard
 - navigation 69
 - PF keys 69
 - shortcut keys 69

N

- navigation
 - keyboard 69
- non-JES output writing routine 53
- Notices 73

O

- output register information
 - IAZSYMBL 36
- output separator routine
 - writing 66
- output separator routine for an external writer
 - block character routine by IBM 66, 67
 - description 64, 65, 66, 67
 - functions 66
 - IBM-supplied output separator routine 64, 65, 66, 67
 - invoking 67
 - module name IEFSD095 66
 - note on protection key 66
 - output from the routine 65
 - parameter list 65, 66
 - requirements for writing 66
 - return codes 66
- output writing routine 58
 - coding conventions 61
 - description 58, 60
 - IBM-supplied routine 58
 - programming considerations 61

P

- performance considerations for JES2 Readers
 - held internal reader 50
 - Record length of SYSIN data sets 50
 - SYSIN record formats 51
 - use of unblocked records for SYSIN and SYSOUT data sets 50

S

- sending comments to IBM xi
- separator routine 64
 - parameter list 65
- shortcut keys 69
- spool data set browse 3
 - ACB/RPL interface 8
 - allocation 3

- spool data set browse (*continued*)
 - compatibility interface 8
 - end of processing 13
 - performance 13
 - secondary subsystem support 14
- started task 49
 - control statements that affect the
 - internal reader 50
 - discussion 48
 - example of conditional JCL submitted
 - from a cataloged data set 49
 - example of console command to read
 - job cataloged data set 49
 - example of console command to read
 - job from tape 49
 - example of RDR procedure 49
 - example to submit from a tape
 - through a batch job 49
- summary of changes xiii
- Summary of changes xiii

T

- time-sharing task
 - logon 48
- trademarks 75

U

- user interface
 - ISPF 69
 - TSO/E 69

X

- XWTR cataloged procedure 54



Product Number: 5650-ZOS

Printed in USA

SA32-0987-01

