

IBM Tivoli Asset Discovery for z/OS
Version 8 Release 1

*Collecting Data, Running Utilities, and
Configuring Language Support Guide*



Note

Before using this information and the product it supports, read the information in "Notices" on page 75.

Contents

Chapter 1. Collecting and importing data with IBM Tivoli Asset Discovery for z/OS 1

Updating the Global Knowledge Base	1
Collecting scanned libraries with the Inquisitor for z/OS	1
Running the Inquisitor program	1
PLX parameter of the Inquisitor program	2
Inquisitor program parameters and files	3
Inquisitor program command syntax	4
Inquisitor examples	10
Designing Inquisitor requests	12
Scanning migrated libraries	13
Scanning generation data sets	14
Collecting information about the I/O configuration	15
Collecting UNIX files with the Inquisitor for z/OS UNIX	15
Inquisitor for z/OS UNIX overview	15
Running the Inquisitor for z/OS UNIX program	15
Inquisitor for z/OS UNIX program parameters and files	16
Security considerations	17
Collecting usage data with the Usage Monitor	18
Setting up the Usage Monitor	18
Starting and stopping the Usage Monitor	21
Refresh processing for the Usage Monitor	22
Usage Monitor commands	23
Monitoring usage in CICS regions	39
Customizing a CICS region to provide usage data	40
Importing Inquisitor data	40
Running the Inquisitor import	41
Import filters and matching	41
TPARAM parameters	42
Importing usage data	43
Activating the Automation Server	43
Automation Server overview	43
Running the Automation Server	45
Creating the Automation Server control data set	45
Copying the started JCL task to a library	45
Designing request control statements	46
Starting and stopping the Automation Server	50
Excluding data sets	50
Automation Server control data set maintenance	51

Chapter 2. Running the utilities provided with Tivoli Asset Discovery for z/OS 53

Condensing usage data with the ZCAT utility	53
Summarizing usage data with the Usage Summary utility	55
Deleting usage data with the Usage Deletion utility	57
Deleting a specific system with the System Deletion utility	58
Listing high-level qualifiers for the Usage Monitor utility	58
Updating the TPARAM table	59
Tagging unidentified products with the Product Tagging utility	59
Product tagging process	59
Product tagging job and control statements	60
Product tagging examples	63
Importing subcapacity reporting data with the SCRT Import utility	64
Capturing historical SMF data with the SMF Scanner utility	65
Extracting data with the XML Export utility	65
Transferring output XML by FTP	66
Compressing and decompressing data sets with the HSIZIP utility	66
Text data processing with the HSIZIP utility	67
Binary data processing with the HSIZIP utility	67
HSIZIP program parameters	68
HSIZIP files	69
Dynamic invocation of the HSIZIP program by other programs	69
HSIZIP data set support	70
HSIZIP return codes	71

Chapter 3. Configuring language support 73

Configuring Japanese messages	73
Enabling the Analyzer utility for Japanese	73
Configuring the Japanese DB2 subsystem for use with Tivoli Asset Discovery for z/OS	74

Notices 75

Trademarks	76
----------------------	----

Chapter 1. Collecting and importing data with IBM Tivoli Asset Discovery for z/OS

Tivoli® Asset Discovery for z/OS® includes programs that collect system and usage data, import, filter and match this data, update the Repository tables, and make the data available for review and query.

Updating the Global Knowledge Base

IBM® provides monthly updates to the Global Knowledge Base (GKB) so that you can keep your product inventory definitions up-to-date. You can also submit items to IBM support for inclusion in GKB updates.

About this task

Updates to the GKB are available from the Fix Central website and you can register for notifications when updates are posted. Each update includes the following files:

- The TADZ81KB.XMI file contains a list of products that were added to the GKB since the last update. The file may also contain special processing instructions, such as running supplied SQL before doing any updates.
- The GKBLVELyymmdd.TXT file contains instructions for applying the update.

Procedure

1. From the Fix Central website, download the TADZ81KB.XMI file.
2. Upload the TADZ81KB.XMI file into a preallocated file on the mainframe with the attributes **FB 80**.
3. Issue the following TSO command to receive the file:
`RECEIVE INDATASET(TADZ81KB.XMI)`
4. Enter DA (*filename*) when you are prompted for additional information.
5. In the GKB load job, HSISGKBL, update the SET INDSN= value with the name of the file that you received, and then submit the job.

Collecting scanned libraries with the Inquisitor for z/OS

The Inquisitor is a program that scans and collects information about partitioned data set (PDS) and partitioned data set extended (PDSE) program libraries. The Inquisitor Import program takes the collected data as input to form the basis of your software inventory.

Related tasks:

“Importing Inquisitor data” on page 40

The Inquisitor Import reads data from Inquisitor scans, where the data is filtered and matched to products. The filtered, matched data is then copied to the Repository tables where it can be viewed and queried by the Analyzer reporting utility.

Running the Inquisitor program

The HSISINQZ job in the JCLLIB library performs the Inquisitor collection. This job is generated from the HSISCUST post-installation customization job.

About this task

Run-time for this job depends on the number of volumes and libraries to be scanned. Run this job during off-peak periods.

Procedure

1. In the HSIINQZ job, check the values for the following parameters and change if necessary:
 - The **ALLMSG** parameter requests both DSNMSG and PGMMMSG message logging.
 - The **PLX** parameter is set to Y (yes) if you plan to collect data for a sysplex and otherwise set to N (no).
Review information about the **PLX** parameter before you set this option.
 - The **LLQ** parameter is set to Z&SMF. You can change this value if you want to generate data sets with unique names without changing the JCL library.These values are set when the HSIINQZ job is created.
2. Optional: In the program parameter string, you can specify a report message level and an override to the system identifier. Use commas to separate the various settings specified within the program parameter string.
3. Run the HSIINQZ job.

PLX parameter of the Inquisitor program

The **PLX** parameter can reduce the time it takes to scan and process different SIDs that are completely shared and are, therefore, identical. When you set **PLX=Y**, the Inquisitor Import detects libraries that are mirrors or libraries that have not changed and quickly processes scans of these shared SIDs.

Plan your Repository to receive scans of system identifiers (SIDs) containing libraries that are unique in library name and volume, except when identically-named libraries are copies or are shared. If you have libraries that are identical in library name and volume name but are intended to have different content, place these libraries in different Repositories so that they can be processed separately.

If a library with the same library name and volume name is encountered in different SID scans, the Inquisitor Import considers the first instance that it encounters on the first SID to be the base. The Inquisitor Import treats any subsequent instances on different SIDs as mirrors.

The locations of all SIDs for a given library are recorded, but module discovery information is only calculated and stored when the library is encountered on its base SID. This approach ensures consistency in matching if the copies are not synchronized. The approach also saves processing time when SIDs are identical. If the Inquisitor Import encounters mirror libraries, it displays their names, current SIDs, and base SIDs in the log file, and reports their number at the end of the run.

If an SID is decommissioned and is no longer available for scanning, you can run the system deletion job to remove the SID and any libraries, modules, and products that are exclusively attached to the deleted SID. For shared libraries, only the record of the library that is attached to the specified SID is removed. The contents of the library are then attached to the subsequent SID in the list which then functions as the base SID.

If the **PLX=Y** option is specified during the Inquisitor run, the Inquisitor Import applies the results of the scan from the Inquisitor file to all SIDs that the were previously processed and share the same sysplex ID with the SID in the Inquisitor file. An existing library is processed on its base SID but is recorded as seen on all SIDs of the sysplex. A new library is processed on the current SID which becomes its base SID and is recorded as seen on all SIDs of the sysplex.

The Inquisitor Import requires that the Inquisitor file is more recent than any Inquisitor file that it previously processed for the same SID. If you specify the **PLX=Y** option during the Inquisitor scan, the Inquisitor file must be more recent than previously processed files of all SIDs of the sysplex.

The default value for the **PLX** parameter is N (no). If you intend to use the **PLX=Y** option to save scanning time, you must scan all SIDs at least once and present all the scans to the Inquisitor Import, so that it can determine how the different SIDs are shared.

When you specify the **PLX=Y** option, the Inquisitor Import processes the file in the following manner:

- Treats the content of all SIDs that share the sysplex ID with the currently-scanned SID as being identical to each other.
- Applies the scan results to all SIDs of the sysplex.

Because the Inquisitor Import process does not verify that all SIDs are identical, incorrect results can occur if the SIDs of the sysplex have different content. Use the **PLX=Y** option only if you are sure that all SIDs of the sysplex are identical in content at the time of the scan.

Inquisitor program parameters and files

The Inquisitor program has mandatory and optional parameters that affect how data is collected. The program uses some mandatory files as well as some optional files.

Table 1. Parameter settings for the Inquisitor

Parameter	Description
DSNMSG	Requests that messages relating to processed data sets, which might otherwise be suppressed, are to be logged in the SYSPRINT report.
PGMMSG	Requests that messages relating to processed programs, which might otherwise be suppressed, are to be logged in the SYSPRINT report.
ALLMSG	Requests both DSNMSG and PGMMSG message logging.
NOAPF	Specifies that the Inquisitor is to run in an environment which is not APF authorized.
SID=	The value is up to 4 characters long, and specifies the system identifier to be contained in the data output from the Inquisitor. If the SID identifier override is omitted, the system SMF identifier is used. The SID parameter setting is used when the SMF system identifier of a system is not unique. For example: SID=SYS2
PLX=	The parameter is used to identify if the Inquisitor data being collected is part of a SYSPLEX. The value is either Y or N. If the PLX parameter is not used, the default value of N is created in the Inquisitor header record.
LLQ=	This parameter is used to specify a suffix string made up of one or more data set name qualifiers to be appended to the data set name of the HSIPZIP and HSIPOUT data set. Its maximum length is 44 characters. It may contain both static and dynamic system symbols, and the user symbols &SMF. (SMF system identifier) and &SYSLPAR. (LPAR name) supplied by the Inquisitor. Use the LLQ setting when you need to create uniquely named data sets without changing the JCL.

Table 2. Files used by the Inquisitor

Filename	Description
SYSPRINT	A mandatory report file.
TAGREP	An optional report file that summarizes tag data collected by the Inquisitor.
SYSIN	A mandatory request input file. It processes fixed length, variable length, and undefined record formats. Records shorter than 72 bytes will be logically extended by the Inquisitor with blanks.
HSIPZIP	An optional output file that contains compressed Inquisitor data. It is written using a variable length record format. You must provide DCB information to ensure optimal use of DASD space.
HSIPOUT	<p>An optional output file that contains uncompressed Inquisitor data. It is not specified in the packaged sample, as the use of HSIPZIP is preferred, due to its reduced space requirements. HSIPOUT also contains variable length records. The program supplies the appropriate LRECL. By default, system determined block size is used.</p> <p>If you want to the direct the Inquisitor output to a compressible extended-format data set, then you should use the HSIPOUT file. The HSIPZIP file employs update-in-place processing, which prevents the use of DFSMS compression.</p>
MCDS	An optional file that allocates the DFHSM MCDS data set, and is required if any requests contain the REMIGRATE or NOML2 operands. Further, if supplied for other requests, you can use it to avoid recalling data sets which are not load libraries. If the DFHSM MCDS is spread over more than one data set, use the DD names MCDS2, MCDS3, and MCDS4 consecutively. This allocates all the MCDS data sets in key range order.
ABRIN	An optional SYSIN file belonging to the FDRABRP utility program that is required if any requests contain the ABRMIG or ABRARC operands. It is primed by the Inquisitor during execution. For this reason, a single track VIO file is an ideal allocation.
ABRPRINT	An optional SYSPRINT file belonging to the FDRABRP utility program that is required if any requests contain the ABRMIG or ABRARC operands. It is an output-only file, and is not processed by the Inquisitor.

Inquisitor program command syntax

The Inquisitor program includes SYSIN commands and optional command operands.

SYSIN commands

The Inquisitor program uses the SCANCMD, SCANDIR, and SCANPGM SYSIN commands that are described in the following table.

Table 3. SYSIN commands used by the Inquisitor

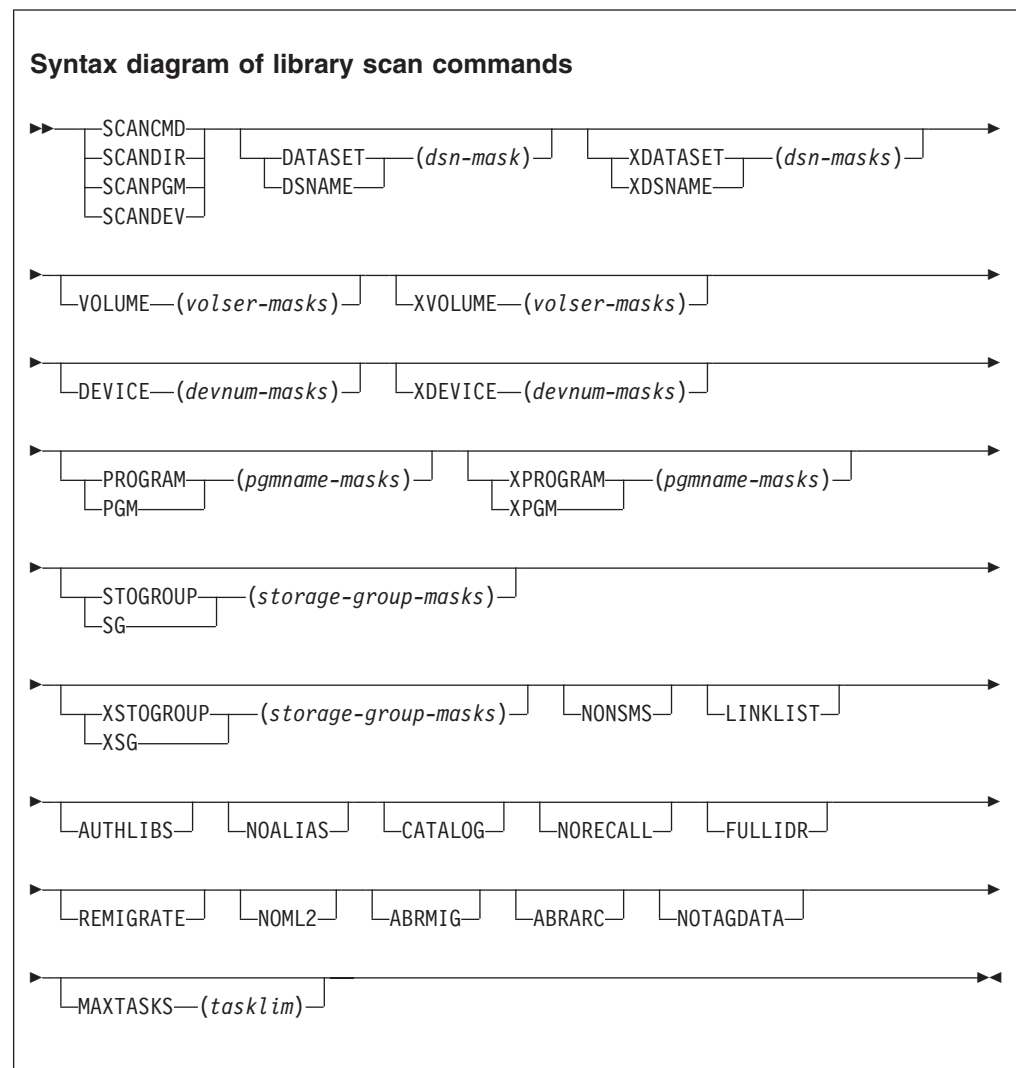
Command	Description
SCANCMD	<p>Allows command syntax and operand consistency to be checked by the Inquisitor without initiating an actual scan for program libraries. It performs a parse only operation, although output files are opened.</p> <p>Error messages relating to syntax and operand errors are produced as usual. This verb is useful if you are formulating the best request combination when implementing on any given system.</p>
SCANDIR	<p>Collects data from program library directory entries. Contents of program members are not accessed.</p> <p>Compared to SCANPGM, its reduced data collection allows it to run faster. Although all syntactically correct operands are allowed, some operands relating to data from member contents are ignored during processing. SCANDIR collects all of the information needed for automated software identification, and is the command of choice for a production environment.</p>

Table 3. SYSIN commands used by the Inquisitor (continued)

Command	Description
SCANPGM	Collects all data collected by SCANDIR, and information from member contents. Such information relates to program structure and history. Your IBM representative might request SCANPGM output data to assist with problem diagnosis and resolution.
SCANDEV	Collects information about the input and output (I/O) configuration of the z/OS system including online I/O devices, control units, and related channel path connectivity. The SCANDEV command has no operands.

The Inquisitor can process multiple requests in a single program run. The output of these requests is contained in the same file.

This syntax diagram shows the SYSIN commands and their operands.



Operand defaults are:

DSNAME(*) VOLUME(*) DEVICE(*) PROGRAM(*)

All operands are optional. They are:

DATASET Alias: DSNNAME

This operand specifies one or more 1 to 44 byte data set name masks. Only data sets with names matching any masks specified here are processed. Data sets with names not matching any masks specified here are not processed. Multiple masks must be separated by one or more delimiters. This operand can be specified more than once in a request, whereupon all masks specified in all occurrences of this operand are checked for selection matching. The precise treatment of asterisks in these masks is altered by the presence of the CATALOG keyword in the request. When CATALOG is specified, mask matching becomes qualifier aware and a single asterisk represents one, or part of, one qualifier only. When CATALOG is specified, use a double asterisk to specify any number of qualifiers. The data set name selection mask is the only mask affected by the CATALOG keyword. When the CATALOG keyword is present, exactly one DSNNAME mask must be specified.

XDATASET Alias: XDSNAME

This operand specifies one or more 1 to 44 byte data set name masks. Data sets with names matching any mask specified here are not processed. Multiple masks must be separated by one or more delimiters. This operand can be specified more than once in a request, whereupon all masks specified in all occurrences of this operand are checked for exclusion matching. If this operand is used, each mask must specify a subset of a DATASET mask.

VOLUME

This operand specifies one or more 1 to 6 byte volume serial number masks. Only volumes with serial numbers matching any mask specified here are processed. Volumes with serial numbers not matching any mask specified here, are not processed. Multiple masks must be separated by one or more delimiters. This operand can be specified more than once in a request, whereupon all masks specified in all occurrences of this operand are checked for selection matching. A volume serial number mask of six asterisks specifies the current IPL volume, which is ascertained during execution.

XVOLUME

This operand specifies one or more 1 to 6 byte volume serial number masks. Volumes with serial numbers matching any mask specified here are not processed. Multiple masks must be separated by one or more delimiters. This operand can be specified more than once in a request, whereupon all masks specified in all occurrences of this operand are checked for exclusion matching. If this operand is used, each mask must specify a subset of a VOLUME mask. A volume serial number mask of six asterisks specifies the current IPL volume, which is ascertained during execution.

DEVICE

This operand specifies one or more 1 to 4 byte device number masks. Only volumes with device numbers matching any mask specified here are processed. Volumes with device numbers not matching any mask specified here, are not processed. Multiple masks must be separated by one or more delimiters. This operand can be specified more than once in a request, whereupon all masks specified in all occurrences of this operand are

checked for selection matching. Standard character string mask matching is used. The use of characters which are not hexadecimal digits will not be detected by the program.

XDEVICE

This operand specifies one or more 1 to 4 byte device number masks. Volumes with device numbers matching any mask specified here are not processed. Multiple masks must be separated by one or more delimiters. This operand can be specified more than once in a request, whereupon all masks specified in all occurrences of this operand are checked for exclusion matching. If this operand is used, each mask must specify a subset of a DEVICE mask. Standard character string mask matching is used. The use of characters which are not hexadecimal digits will not be detected by the program.

PROGRAM Alias: PGM

This operand specifies one or more 1 to 8 byte program name masks. Only programs with names matching any mask specified here are processed. Programs with names not matching any mask specified here, are not processed. Multiple masks must be separated by one or more delimiters. This operand can be specified more than once in a request, whereupon all masks specified in all occurrences of this operand are checked for selection matching.

XPROGRAM Alias: XPGM

This operand specifies one or more 1 to 8 byte program name masks. Programs with names matching any mask specified here are not processed. Multiple masks must be separated by one or more delimiters. This operand can be specified more than once in a request, whereupon all masks specified in all occurrences of this operand are checked for exclusion matching. If this operand is used, each mask must specify a subset of a PROGRAM mask.

STOGROUP Alias: SG

This operand specifies one or more 1 to 8 byte storage group name masks. SMS-managed volumes in a storage group with a name matching any mask specified here are processed. SMS-managed volumes in a storage group with a name that does not match any mask specified here, are not processed. Multiple masks must be separated by one or more delimiters. This operand can be specified more than once in a request, whereupon all masks specified in all occurrences of this operand are checked for selection matching. Volumes which are not SMS-managed are not processed unless the NONSMS keyword operand is specified.

XSTOGROUP Alias: XSG

This operand specifies one or more 1 to 8 byte storage group name masks. SMS-managed volumes in a storage group with a name matching any mask specified here are not processed. Multiple masks must be separated by one or more delimiters. This operand can be specified more than once in a request, whereupon all masks specified in all occurrences of this operand are checked for exclusion matching. If both this mask and a STOGROUP mask are used, then each mask must specify a subset of a STOGROUP mask.

NONSMS

This keyword operand specifies that volumes which are not SMS-managed are eligible for processing. The presence of this operand means that SMS-managed volumes are not processed unless the STOGROUP operand was used to supply a storage group name mask.

LINKLIST

This keyword operand specifies that all link list data sets are to be unconditionally included for processing.

AUTHLIBS

This keyword operand specifies that all APF authorized data sets are to be unconditionally included for processing.

NOALIAS

This keyword operand specifies that any program member marked as an alias is to be excluded from processing.

CATALOG

This keyword operand specifies that data sets to be processed are located from a catalog search rather than VTOC searches. Data set alias names are not processed. The Inquisitor triggers and waits for a RECALL operation for each migrated data set which passes data set name mask processing, unless NORECALL is also specified.

NORECALL

This keyword specifies that migrated data sets are not to be recalled and are excluded from processing. This operand only has effect when the CATALOG operand is also specified. Data sets with a catalog entry indicating a volume serial number of MIGRAT, or ARCIVE, are deemed to be migrated.

FULLIDR

This keyword operand specifies that a full scan of CESD and IDR records is to be performed, even when a module would not have been selected for such processing. Depending upon the exact nature of the request being run, this operand can significantly elongate the elapsed time of Inquisitor runtime.

This operand is ignored for a SCANDIR request.

REMIGRATE

This keyword operand specifies that when a data set which had to be recalled has been processed, DFHSM is requested to migrate the data set again asynchronously. Migrated data sets can only be processed when the CATALOG operand is also specified. Only data sets with a catalog entry indicating a volume of MIGRAT are remigrated.

The presence of this operand requires that the MCDS file is allocated to the DFHSM MCDS. Access to the MCDS allows the Inquisitor to avoid recalls for data sets which are not partitioned, do not have an undefined record format, and do not have a block size of at least 1024.

NOML2

This keyword operand specifies that data sets migrated to level two are not to be recalled and are excluded from processing. Migrated data sets can only be processed when the CATALOG operand is also specified. Only data sets with a catalog entry indicating a volume of MIGRAT are checked for level two status.

The presence of this operand requires that the MCDS file is allocated to the DFHSM MCDS. Access to the MCDS allows the Inquisitor to avoid recalls for data sets which are not partitioned, do not have an undefined record format, and do not have a block size of at least 1024.

ABRMIG

This keyword operand indicates that when a catalog entry with a volume

of MIGRAT is encountered, the FDRABR product is to be invoked to determine whether a recallable archived copy of the data sets is available or not. If it is, then the data set is processed. If not, then the data set is not processed.

The NORECALL operand takes precedence over this operand.

The effect of ABRMIG is not affected by the ABRARC operand.

The presence of this operand requires that the ABRIN and ABRPRINT files are allocated.

ABRARC

This keyword indicates that, when a cataloged data set cannot be found on the volume, the FDRABR product is to be invoked in order to determine whether a recallable archived copy of the data set is available. If it is, then the data set is processed. If not, the data set is not processed.

The NORECALL operand takes precedence over this operand.

The effect of ABRARC is not affected by the ABRMIG operand.

The presence of this operand requires that the ABRIN and ABRPRINT files are allocated.

NOTAGDATA

This keyword indicates that data written to program libraries by the Product Tagger is not to be collected and written to the Inquisitor output file. Use this operand only when you do not want to update the Local Knowledge Base during the import process with the latest Tagger data that could be found by the Inquisitor.

MAXTASKS

This operand specifies the maximum number of VTOC-scanning subtasks to be activated by the Inquisitor. These subtasks reduce the elapsed time of an Inquisitor scan by enabling the concurrent processing of multiple volumes. Reducing the number of subtasks reduces the demand on storage in the region and does not impact performance unless the main task has to wait longer for VTOC scan results. The operand value is a single decimal number in the 1 to 200 range. The default value is 200. The actual number of subtasks used does not exceed the number of volumes to be scanned. VTOC-scanning subtasks are not used for CATALOG requests.

SYSIN syntax rules for the Inquisitor

Syntax rules are as follows:

- Only the first 72 bytes of an input record are ever scanned.
- Short records are extended to 72 bytes with blanks.
- Blanks and commas are equivalent.
- Subparameters of value operands are specified in parentheses.
- A continuation to the next record is requested by a plus or a hyphen when it follows a delimiter, or is at the start of a record.
- A continuation cannot be requested in the middle of a word or value.
- The part of the record following a continuation character is ignored and can be used for comments.
- Records beginning with an asterisk are comment records.
- Records containing only blanks or commas are comment records.

- Comment records are ignored by syntax parsing logic, and do not alter continuation status.
- TSO conventions apply to abbreviations. That is, operands can be abbreviated to the minimum unambiguous length. Verbs cannot be abbreviated.
- If the input record contains an ampersand, the system symbol substitution routine ASASYMBM is called to perform symbol substitution processing.
- All input requests are parsed and stored before the first request is processed.
- If a syntax error is encountered, no requests are processed. This is to reduce the instance of incorrect or unproductive requests triggering lengthy DASD subsystem scans. The error is in the last record echoed in SYSPRINT.
- Value masks are character strings which are compared to data found at run time. Comparison is performed one byte at a time, from left to right. For a match, the characters must compare equal, unless a generic mask character is found.
- System static symbols, system dynamic symbols, and &SMF (SMF system identifier) and &SYSLPAR (LPAR name), can be used to construct value masks. &SYSLPAR may resolve to a null string if z/OS is running in a virtual machine.
- Valid generic mask characters are a percent (%), to flag a match for any single character, and an asterisk (*), to flag a match for any character string segment of zero or greater length.

Inquisitor examples

These examples show some possible scenarios where you can customize the scope and type of processing when you run the Inquisitor program.

Example 1

These three statements are equivalent, and request data collection for all programs on all online DASD volumes.

```
SCANDIR
SCANDIR DA(*) PGM(*)
SCANDIR VOL(*) DS(*)
```

Example 2

To scan all SMS-managed volumes except volumes in storage group SGWORK use:

```
SCANDIR STOGROUP(*) XSTOGROUP(SGWORK)
```

Example 3

To scan all volumes except volumes in storage groups with names beginning with SGW use:

```
SCANDIR XSTOGROUP(SGW*) NONSMS
```

Example 4

To scan all volumes with serial numbers beginning with TSO and WRK, these two requests are used in a single program run:

```
SCANDIR VOLUME(TSO*)
SCANDIR VOLUME(WRK*)
```

Example 5

To scan all volumes except those with serial numbers beginning with TSO and WRK use:

```
SCANDIR XVOLUME(TSO* WRK*)
```

Example 6

To scan all volumes with serial numbers beginning with USR which are also in SMS storage groups with names beginning with SG for programs with names beginning with UTIL, use: .

```
SCANDIR VOLUME(USR*) STOGROUP(SG*) PROGRAM(UTIL*)
```

Example 7

To scan all data sets with high level qualifiers of SYS1, SYS2, SYS3, except z/OS distribution libraries, use:

```
SCANDIR DSNAME(SYS%.* ) XDSNAME(SYS1.A*)
```

Example 8

To restrict the data in the previous example to cataloged data sets, use:

```
SCANDIR DSNAME(SYS%.** ) XDSNAME(SYS1.A*) CATALOG
```

Note: Note the extra asterisk in the data set name selection mask. Without this, only data set names with two qualifiers are selected. Data set name exclusion processing is not changed by the CATALOG operand.

Example 9

To scan the current IPL volume, and any other link, list, and APF authorized libraries use:

```
SCANDIR VOLUME(***** ) LINKLIST AUTHLIBS
```

Example 10

To scan the single cataloged data set SYS1.PPLIB without a lengthy DASD subsystem scan use:

```
SCANDIR DATASET(SYS1.PPLIB) CATALOG
```

Example 11

To scan all cataloged SYS1 and SYS2 data sets use (a) two requests in a single program run, or (b) a single request. The two approaches exhibit similar resource consumption:

```
SCANDIR DA(SYS1.** ) CAT  
SCANDIR DA(SYS2.** ) CAT
```

```
SCANDIR DS(SYS%.** ) CAT XDSN(SYS3.*,SYS4.*,SYSA.*)
```

The XDSN values are coded as shown under the assumption that SYS1, SYS2, SYS3, SYS4 and SYSA are the only 4 character high-level qualifiers beginning with SYS on the system being scanned.

Note: SCANDIR DS(SYS1.** ,SYS2.**) CAT is not allowed.

Example 12

These examples are all equivalent. They scan the entire DASD subsystem for all data sets with a first qualifier of SYS1 or SYS2, excluding those with a second qualifier beginning with A.

(a)

```
SCANDIR DA(SYS1.*,SYS2.*) XDA(SYS1.A*,SYS2.A*)
```

(b)

```
SCANDIR DA(SYS1.*      +  
SYS2.*)      +  
XDA(SYS1.A*    +  
SYS2.A*)
```

(c)

```
SCANDIR DA(SYS1.*)      +  
DA(SYS2.*)      +  
XDA(SYS1.A*)      +  
XDA(SYS2.A*)
```

(d)

```
SCANDIR DA(SYS1.*) XDA(SYS1.A*) +  
DA(SYS2.*) XDA(SYS2.A*)
```

(e)

```
SCANDIR DA(SYS1.*) XDSN(SYS1.A* SYS2.A*) DS(SYS2.*)
```

Designing Inquisitor requests

When constructing statements for the Inquisitor SYSIN file, try to combine all selection and exclusion criteria to form a single SCANDIR request. A single Inquisitor request will not scan a VTOC or a library more than once.

It can be difficult to formulate a system scan into a single CATALOG request, meaning that when the CATALOG operand is used, multiple requests are coded. Ensure that no data set will be scanned by more than one SCANDIR CATALOG request by excluding as many data set name patterns from each request as necessary. Data set name exclusions may not be necessary if all CATALOG search selection masks represent disjoint parts of the name space.

The example shown here uses the XDA operand to prevent SYS1.LINKLIB from being scanned more than once:

```
SCANDIR DA(SYS1.***) CATALOG  
SCANDIR DA(SYS%.LINKLIB) XDA(SYS1.LINKLIB) CATALOG
```

As well as using the selection and exclusion facilities to ensure completeness, they can also be used to improve performance and efficiency by excluding DASD volumes which do not contain program libraries. Although a volume with no program libraries can be scanned quickly, processing duration might be reduced if such volumes can be excluded from an Inquisitor scan.

For example, volumes that only contain databases, or temporary data sets, do not have any files suitable for Inquisitor processing, but the VTOCs of those volumes are still read unless excluded by the appropriate selection criteria.

To illustrate this further, consider a system with these DASD subsystem usage elements:

System platform

Non-SMS and storage group SYSTEM.

Work pool

Storage group TEMP containing temporary and short-lived (two days) permanent files.

TSO Storage groups TSOONE and TSOTWO.

Non-DB application

Non-SMS and storage groups BATCH1 and BATCH2.

Databases

Non-SMS volumes DBA001 to DBA099 and SMS storage groups DB01, DB02, and DB03.

The scanning of this configuration is to be carried out with the following assumptions:

- No need for data from libraries that do not exist for more than two days.
- No program libraries on database volumes.
- Applications combine their program libraries and non-database files.
- TSO users can have program libraries.
- Management requires information regarding all potentially permanent executable software.

To acquire Inquisitor data from all useful sources without processing volumes more than once, and without processing irrelevant volumes, you can specify multiple requests in a single Inquisitor run. For example:

```
SCANDIR SG(SYSTEM)
SCANDIR SG(TSO*)
SCANDIR SG(BATCH*)
SCANDIR NONSMS XVOL(DB*)
```

This can be consolidated into a single request giving the same result. For example:

```
SCANDIR SG(SYSTEM TSO* BATCH*) NONSMS XVOL (DB*)
```

Scanning migrated libraries

The Inquisitor locates load libraries by either scanning the VTOC of online volumes, or by searching the system catalog (CATALOG) for relevant data sets. When you use the Catalog Search Interface, you can return data sets for migrated libraries. VTOC scans do not find migrated data sets.

When the keyword CATALOG is specified in a request statement, the Inquisitor passes the data set name selection mask to the Catalog Search Interface (CSI) to search for the catalog entries. It is possible that one or more of the catalog entries returned by the CSI are for a data set that has been migrated. In contrast, VTOC scans do not find migrated data sets.

Inquisitor processing of migrated data sets found by the CSI involves dynamic allocation which then triggers the recall of the data set. Recalls increase Inquisitor processing time. The processing leaves the data set in a recalled status.

The Inquisitor looks at the volume serial number in the catalog entry to determine if a data set is migrated or not. A data set is considered to have been migrated if its catalog entry indicates a volume serial number of either MIGRAT or ARCIVE.

To suppress the processing of all migrated data sets, specify the NORECALL keyword on each Inquisitor request.

Integration with DFHSM

If you are using the MCDS file allocation, and a data set cataloged on volume MIGRAT is encountered, the Inquisitor can read the data set record from the DFHSM Migration Control Data Set (MCDS) to verify that the data has the attributes of a program library. If the MCDS record is not found, the data set is ignored and processing is bypassed, avoiding a DFHSM error condition. If the data set does not have partitioned organization, an undefined record format, and a block size of at least 1024, the Inquisitor ignores the data set, avoiding the recall of many data sets which are not program libraries.

For systems with DFHSM space management functions, you can use the request keywords NOML2 and REMIG. The MCDS file allocation is a prerequisite for using the following keywords:

NOML2

Specifies that data sets migrated to level 2 are excluded from the scan.

REMIG

Specifies that after a recalled data set is processed by the Inquisitor, the Inquisitor requests DFHSM to remigrate the data set. The Inquisitor does not wait for the migration to complete, but begins to process the next data set immediately after making the request to DFHSM. Migration level 2 is never specified by the Inquisitor for the migration, even if the data set was recalled from ML2. (However, it might be selected by DFHSM as a result of SMS management class settings.)

Note:

Any combination of REMIGRATE, NOML2, and NORECALL is valid. Specifying NORECALL means NOML2 and REMIGRATE have no effect.

In the case where you want to scan all relevant migrated program libraries and do not want any such libraries explicitly remigrated afterward, you would not code any of the NORECALL, NOML2 and REMIGRATE keywords. In this instance, the MCDS file allocation, though optional, can still be used to great advantage.

Scanning generation data sets

Inquisitor CSI requests are limited to NONVSAM type A catalog entries. Generation data sets (which are members of a generation data group) are not scanned by Inquisitor CATALOG requests but can be processed by Inquisitor VTOC scans. Consider excluding generation data sets if you back up program libraries using generation data sets.

To exclude generation data sets from a VTOC scan request, specify a suitable data set exclusion mask, for example:

```
XDA(*.G%%V00)
```

Collecting information about the I/O configuration

The Inquisitor can scan the input and output (I/O) configuration of a z/OS system to collect information about the use of hardware assets such as storage devices. The SCANDEV command is used to request such a scan.

When you run the SCANDEV command, two additional types of records are generated which describe device groups and channel paths. Consideration of I/O devices is limited to those devices which are online at the time of the scan.

A device group is a contiguous block of device numbers, not including offline devices, where the device type, control unit type and serial number, and online channel path connectivity is the same. The channel path type is collected for each channel path used to connect to an online I/O device.

A channel that does not provide an online path to any online device is not reported even if the channel is configured online to the z/OS system.

Collecting UNIX files with the Inquisitor for z/OS UNIX

The Inquisitor for z/OS UNIX is a program that collects information about executable software existing in HFS and zFS data sets currently mounted and accessible to z/OS UNIX. The Inquisitor Import program takes the collected data as input to form the basis of your z/OS UNIX software inventory.

Inquisitor for z/OS UNIX overview

The Inquisitor for z/OS UNIX produces a set of record types which are different from those produced by the Inquisitor for z/OS. However, both programs collect the same types of information about installed software.

The Inquisitor for z/OS UNIX processes the hierarchical file system (HFS) root directory, as well as all subdirectories. For this reason, the program must run with a UID that allows access to all directories and programs to be examined. If the Inquisitor for z/OS UNIX does not have permission to access a directory, then no information is collected from that directory, or any of its subdirectories.

The HSIXROOT file is used to nominate one or more directories to be considered root directories. When specified, only the nominated directories and their subdirectories are processed. This facility is useful when only a subset of the file hierarchy needs to be scanned.

The HSIXOMIT file is used to nominate one or more directories which are to be omitted or excluded from the scan, together with all of their subdirectories. This facility can be used to reduce resource consumption by preventing parts of the UNIX file hierarchy known not to have any executable software from being scanned.

Running the Inquisitor for z/OS UNIX program

The HSIINQU job in the JCLLIB library performs the Inquisitor for z/OS UNIX collection. This job is generated from the HSISCUST post-installation customization job.

About this task

Run-time for this job depends on the size and complexity of the UNIX directory structure to be scanned. Run this job during off-peak periods.

Procedure

1. In the HSIINQU job, check the values for the following parameters and change if necessary:
 - The **PLX** parameter is set to Y (yes) if you plan to collect data for a sysplex and otherwise set to N (no).
 - The **LLQ** parameter is set to Z&SMF. You can change this value if you want to generate data sets with unique names without changing the JCL library.These values are set when the HSIINQU job is created.
2. Optional: In the program parameter string, you can specify a report message level, an override to the system identifier, and whether you want compressed or uncompressed output. Use commas to separate the various settings specified within the program parameter string.
3. Run the HSIINQU job.

Inquisitor for z/OS UNIX program parameters and files

The Inquisitor for z/OS UNIX program has mandatory and optional parameters that affect how data is collected. The program uses some mandatory files as well as some optional files.

Table 4. Parameter settings for Inquisitor for z/OS UNIX

Parameter	Description
PTHMSG	Requests that a message is written to HSIXMSG each time a directory is opened or closed.
PGMMSG	Requests that a message is written to HSIXMSG each time an executable file is processed.
ALLMSG	Requests both PTHMSG and PGMMSG message logging.
SID=	The value is up to 4 characters long and specifies the system identifier to be contained in the data output from the Inquisitor. If the SID identifier override is omitted, the system SMF identifier is used. The SID parameter setting is used when the SMF system identifier of a system is not unique. For example: SID=SYS2
PLX=	The setting is used to identify if the Inquisitor data being collected is a SYSPLEX. The value is either Y or N. If the PLX parameter is not used, the field remains blank in the Inquisitor header record.
OUT=	Specifies output file usage. The default value is Z. <ul style="list-style-type: none">• A value of Z requests zipped output to HSIXZIP.• A value of T requests text output to HSIXOUT.• A value of B requests output to both HSIXZIP and HSIXOUT files.
LLQ=	This parameter is used to specify a suffix string made up of one or more data set name qualifiers to be appended to the data set name of the HSIXZIP and HSIXOUT data set. Its maximum length is 44 characters. It may contain both static and dynamic system symbols, and the user symbols &SMF. (SMF system identifier) and &SYSLPAR. (LPAR name) supplied by the Inquisitor. Use the LLQ setting when you need to create uniquely named data sets without changing the JCL.

Table 5. Files used by the Inquisitor for z/OS UNIX

Filename	Description
HSIXMSG	Report file used by HSIXINQ.

Table 5. Files used by the Inquisitor for z/OS UNIX (continued)

Filename	Description
SYSPRINT	Used by Language Environment® (LE), which is required to be in the standard module search path, and by IDCAMS when LLQ= is specified.
SYSOUT	Used by Language Environment (LE), which is required to be in the standard module search path.
HSIXZIP	An optional output file that contains compressed Inquisitor for z/OS data. It is written using a variable length record format. You need to provide DCB information to ensure optimal use of DASD space. The HSIXZIP file must never undergo any translation when being transferred, whatever the architecture of the target system. That is, only BINARY transfers are to be used to transport the file.
HSIXOUT	An optional output file that contains uncompressed Inquisitor for z/OS UNIX data. It is not specified in the packaged sample, as the use of HSIXZIP is preferred, due to its reduced space requirements. HSIXOUT also contains variable length records. The program supplies the appropriate LRECL. By default, system determined block size is used. If you want to direct the Inquisitor for z/OS UNIX output to a compressible extended-format data set, then you should use the HSIXOUT file. The HSIXOUT file employs update-in-place processing, which prevents the use of DFSMS compression.
HSIXROOT	An optional file which can contain one or more records; each of which specifies a directory path to be considered as a root directory to be processed. If HSIXROOT is not allocated or empty, then a forward slash (/) is considered to be the only root directory to be processed.
HSIXOMIT	An optional file which can contain one or more records; each of which specifies a directory path which is to be omitted from the scan. Root directories cannot be omitted.

The HSIXROOT and HSIXOMIT files have the following characteristics and attributes in common:

- There is no requirement for the file to be allocated.
- The file might be empty or allocated to DUMMY.
- The file might contain fixed length or variable length records.
- Records must not contain more than 1024 bytes of data.
- Blank records are deemed to be comments and discarded.
- Leading and trailing blanks are discarded when the directory name is extracted.
- Records with an asterisk as the first nonblank are deemed to be comments and discarded.
- If the directory path does not end in a slash, then one is appended.

Security considerations

If you want to collect all relevant z/OS UNIX data, you must have access to all UNIX directories, including the root directory. This access ensures that all z/OS UNIX data is collected.

To allow the Inquisitor unrestricted read access to all z/OS UNIX files, consider using the UNIXPRIV RACF® Resource Class, which alleviates the need for UID(0).

The following sample definition can be used by your Security Administrator to define, permit, activate, and RACLIST the RACF UNIXPRIV Class:

```
RDEL UNIXPRIV SUPERUSER.FILESYS.**
RDEF UNIXPRIV SUPERUSER.FILESYS.** UACC(NONE) OWNER(IBMUSER)
PE SUPERUSER.FILESYS.** CLASS(UNIXPRIV) RESET
```

```
PE SUPERUSER.FILESYS.** CLASS(UNIXPRIV) ID(USERONE) ACCESS(READ)
SETR CLASSACT(UNIXPRIV)
SETR RACLIST(UNIXPRIV)
SETR RACLIST(UNIXPRIV) REFR
```

Collecting usage data with the Usage Monitor

The Usage Monitor is a server address space that runs as a started task. Work is queued to the Usage Monitor from all address spaces where programs are used. The Usage Monitor can move captured data into the data space repository and can also write accumulated program usage to a sequential file. The Usage Monitor runs APF authorized and is nonswappable.

Related concepts:

“Importing usage data” on page 43

The Usage import job imports data generated by the Usage Monitor and aggregates usage data for discovered or identified modules in the Repository tables in the database.

Setting up the Usage Monitor

The Usage Monitor uses the HSIJMON job in the JCLLIB library. This job is generated from the HSISCUST post-installation customization job. This job will call the HSIJMON procedure from the JCLLIB library.

Parameters for the Usage Monitor job are HSISMNPM member in the PARMLIB library.

Files used by the Usage Monitor

The Usage Monitor has three product-specific files. They are:

HSIZIN

A sequential file consisting of fixed length 80 byte records. It contains initial commands which are run before data collection becomes active. It must contain the data set prefix to be used for dynamically created output files. The prefix can be changed later by an operator MODIFY command.

HSIZIN is opened, read, and closed during initialization processing. Do not specify FREE=CLOSE in the JCL for HSIZIN, or refresh processing is not possible.

Dataset &HSIINST..&DB..UM.HLQIDS contains the high-level qualifier listing for products and is populated by the IQ Import job (HSISIQIM). To minimize the number of records to be created by the Usage Monitor, only usage events that match the list of products in this dataset are generated. To activate this facility, in //HSIZIN, uncomment dataset &HSIINST..&DB..UM.HLQIDS as described in PROC HSIJMON.

HSIZMSG

A log file which contains the initial commands issued, and which indicates their degree of success. It also contains regular status reports, refresh reports (when appropriate), and a termination report. It consists of fixed length 121 byte records.

SYSOUT

A report file used by the SORT program.

Output files containing program usage data are dynamically allocated by the Usage Monitor. The data set name prefix, the allocation unit, and the primary and

secondary space allocation quantities (in tracks), need to be customized for the target system. This is done in the PARMLIB member HSISMNPM.

Using exclusion masks to reduce data

The data from a significant number of program usage events does not contribute meaningfully to the task of managing the software inventory. To reduce the processing of this unnecessary data, two mechanisms which allow some data to be excluded from collection have been provided. They are exclusion masking based on program name, and exclusion masking based on data set name.

Filtering by program name

A program name exclusion table exists which contains program name masks. When a program usage event is detected by the Usage Monitor, the program name is checked against entries in the program name exclusion table. When a match is found, the usage event data is discarded. Program name exclusion filtering occurs before the data set name of the program library is determined by the Usage Monitor which makes it more efficient than data set name filtering.

Each table entry contains a program name comparison string up to 8 bytes long. The string is either an 8 byte program name, or a shorter program name prefix. When entering these strings with the EXC command, a prefix is denoted by using an asterisk as the last character.

The program name exclusion table resides in key zero common storage, and its size is always a multiple of 4,096 bytes. The minimum table size can house up to 253 entries, and the table size increases dynamically, as required. The default program name exclusion table contains entries to exclude data pertaining to the usage of many programs which are part of the operating system.

In order to add, reset, remove, or display the entries to the table, use these commands:

- EXC** To add entries to the program name exclusion table, or to reset the table to its default contents.
- DEL** To remove some, or all, entries from the table.
- D-X** To display the current contents of the table.

Unlike masks added by the EXC command, default program name exclusion masks do not exclude job step program usage events. So, for example, the IEF* default exclusion mask excludes dynamic calls and loads of program IEFBR14, but usages where IEFBR14 is invoked by JCL are not excluded by this mask.

Filter by data set name

After the Usage Monitor has ascertained the name of the data set from which a used program is fetched, it is used to decide if the usage data is retained for collection or discarded. To perform this process, three lists of data set name masks are scanned; the first is the default data set name exclusion list, the second is the dynamic data set name inclusion list, and the third is the dynamic data set name exclusion list.

The default data set name exclusion list is built during Usage Monitor initialization, and consists of the SCEERUN library, the SCEERUN2 library,

SYS1.CMDLIB (containing TSO commands) and SYS1.CSSLIB (containing callable services modules). The data set names of the SCEERUN and SCEERUN2 Language Environment libraries are determined by searching the link list for specific LE modules. You can use the XDD command to deactivate any of these default exclusion entries. You can use the XDS(*DFLT*) command to reactivate the default data set exclude list without affecting the status of masks in other lists.

The other two lists are constructed from commands you specify either in the HSIZEIN file or dynamically via the system MODIFY command.

To avoid excessive storage and processor resource consumption, it is preferable to keep the number of elements in each list to a minimum. This is achieved by using generic masks to cover many data set names. The inclusion mask list is provided so that specific exceptions to broad exclusion rules can be specified. If you do not supply any data set name exclusion masks, the inclusion list does not affect data collection, but can still be used as a convenient way to collect relative usage statistics from the regular Usage Monitor status reports.

Data set name filtering occurs in the following sequence:

1. Excludes usage if the data set name matches a default exclusion mask, otherwise proceeds to step 2.
2. Include usage if the data set name matches a mask supplied by an IDS command, otherwise proceeds to step 3.
3. Excludes usage if the data set name matches a mask supplied by an XDS command, otherwise proceeds to step 4.
4. Includes usage if the data set name does not match any of the masks.

Data set mask elements reside in key zero common storage. Each element occupies 56 bytes, and contains a data set name mask up to 44 bytes in length. You can use the percent sign as a wildcard to match a single character. You can use a trailing asterisk to match the rest of the data set name.

In order to add, reset, remove, or display the entries to the tables, use these commands:

- XDS** To add a data set name mask to the exclusion list.
- IDS** To add a data set name mask to the inclusion list.
- XDD** To deactivate a data set name exclusion mask.
- IDD** To deactivate a data set name inclusion mask.
- D-D** To display all active data set name masks.

Both of the non-default lists have no elements until an XDS or IDS command is processed. Storage is dynamically acquired for each element as required. To ensure system integrity, XDD and IDD commands do not cause the storage of a deactivated element to be freed, but mark the element as inactive. When a deactivated mask is reactivated, the existing entry is marked as active without the further acquisition of storage.

When the Usage Monitor address space first initializes, all elements of lists that remain in storage from a previous run are freed before the processing of initial commands and the commencement of data collection.

There is no requirement to use either data set name mask list at any stage.

Filtering by UNIX path name

If the mask value specified in an IDS, XDS, IDD or XDD command contains at least one slash, the value is deemed to be a UNIX path name mask and not a data set name mask. During processing, multiple consecutive slashes are reduced to a single slash.

UNIX path name masks entered via IDS and XDS commands are compared to the path names specified by applications at run time and may not correspond to the path names against which usage is attributed. The main cause for this difference in path names is the use of symbolic links. The Usage Monitor writer task converts path names with symbolic links to real path names in order to match inventory discovered by the Inquisitor.

Do not use an IDD or XDD command that specifies a UNIX path name mask because the only use for such a command is to dynamically delete a UNIX path name mask. Most UNIX path name masks contain lower case alphabets. The system MODIFY command interface usually changes lower case characters to upper case which prevents the mask matching the relevant active mask. To delete a UNIX path name mask you must either recycle the Usage Monitor or use the REF command to refresh the settings from the HSIZIN file. In either case, all UNIX path name masks are deactivated and the necessary change is to remove the IDS or XDS command that you want to deactivate from the HSIZIN file.

Similarly, because of the prevalence of lower case alphabets in UNIX path names, you only specify IDS and XDS commands with path name masks as HSIZIN file input rather than via the MODIFY system command interface.

The length limit of 44 characters also applies to UNIX path name masks.

Starting and stopping the Usage Monitor

A Usage Monitor member named HSIJMON is provided in SHSIPROC. If you want to start HSIJMON as a started task, copy the customized member from the JCLLIB to an authorized PROCLIB.

Procedure

1. To start the Usage Monitor in normal mode, enter the following command:
S HSIJMON

2. To fully stop the Usage Monitor, enter one of the following commands:

```
P HSIJMON
F HSIJMON,STOP
F HSIJMON,END
```

These commands cause the Usage Monitor to stop data collection, attach a writer task to process the existing data in the data space, wait for the writer task to sort and output the data, and then terminate.

The writer task can take longer to process collected data than if you stop library lookaside (LLA) before the shutdown of the Usage Monitor is complete.

3. To perform a quick stop of the Usage Monitor, enter the following command:
F HSIJMON,QUICK

This command causes the server address space to stop collecting data, attaches a writer task to process the existing data in the data space, waits for the writer task to complete, and then terminates without sorting the data.

This command is not appropriate for general use and is provided only to increase the available options during an operational crisis.

4. To perform an immediate termination, enter the following command:

```
F HSIJMON,CAN
```

This command causes the server address space to stop data collection, detaches any running writer task which renders the output data set unusable, deletes the current data space without writing out its contents, and terminates. If you use the z/OS system command CANCEL to stop the Usage Monitor, its data space remains in storage. To clear the data space from storage, you must restart the Usage Monitor.

Refresh processing for the Usage Monitor

The Usage Monitor includes commands that you can issue dynamically to alter processing but that are active only for the duration of the current Usage Monitor session. To implement a change to both the running Usage Monitor and to the initialization commands for starting subsequent Usage Monitor sessions, you can use the refresh facility.

Refresh processing involves the execution of the command stream placed in the HSIJIN file, without the requirement of stopping and restarting the Usage Monitor. As a result, refresh processing can verify the validity of the initialization command stream so that changes are made and tested dynamically. This ensures that future Usage Monitor sessions do not encounter initialization command stream errors.

Some commands set a switch for logic control, or set a numeric value to be used during processing. These commands specify the values to be used in the future. Other commands pertaining to inclusion and exclusion masking add a mask to, or remove a mask from, the active mask list, so are part of an accumulation of commands which specify future processing.

Consider the example where several exclusion masks are active, and a change to deactivate one of the masks is required. A command to deactivate the mask might be issued dynamically, but if this change is to be made permanent, then the HSIJIN file needs to be updated. The alternative is to remove the command setting the exclusion from the HSIJIN file, and to then issue the Usage Monitor REF command to initiate a refresh.

Before the first HSIJIN command is run during refresh processing, the program mask exclusion list is set to the default list. Further, all data set name exclusion masks are deactivated, and all data set name inclusion masks are deactivated. This order of deactivation ensures that there is no loss of data that would otherwise be collected. However, there is the possibility that data which would have been excluded is collected during the short window between the reset of the mask lists and the processing of the HSIJIN commands.

The response to each command in the HSIJIN file is written to the HSIJMSG file. A summary WTO message, indicating whether any errors are found or not, is issued after refresh processing has finished.

Stopping the Usage Monitor and restarting it, produces the same active exclusion masks as a refresh. It also produces a data collection outage. For more information, see the REF command in the next topic for a list of the processes performed during a refresh operation.

Usage Monitor commands

The Usage Monitor commands are passed to the Usage Monitor from the HSIZIN input file, or by an operator MODIFY command.

The syntax rules are as follows:

- All commands are three characters long.
- Operands or subparameters are specified in parentheses.
- Multiple subparameters are separated by commas.
- The command must not contain any embedded blanks.
- Commands must start in column one.

To record the settings the Usage Monitor is using, place the display commands at the end of the HSIZIN file.

Details of each command follow.

CAP - Set hardware capacity collection status

CAP is used to specify if the Usage Monitor is to produce records containing information about the hardware capacity of the system. Collecting this information is important when hardware capacity changes dynamically.

A change to this setting does not take effect until the next data space repository switch.

►►CAP(☐Y☐N)◄◄

Y Specifies that hardware capacity data is collected and written out.

N Specifies that hardware capacity is not collected or written out.

If no CAP command is issued after IPL, the default is CAP(Y).

Table 6. Examples of using the CAP command

Command purpose	Example code
Collect hardware capacity data.	F HSIJMON,CAP(Y)
Do not collect hardware capacity data.	F HSIJMON,CAP(N)

CIC – Allow or disable program usage data from CICS regions

The CIC command provides a system-wide control mechanism to allow or disallow program usage data to be collected by the Usage Monitor CICS global user exit (GLUE) program.

A change to this setting does not take effect until the next data space repository switch.

►►CIC(☐Y☐N)◄◄

If no CIC command is issued after IPL, the default is CIC(Y).

CSA - Set the ECSA queuing storage limit

CSA is used to specify a limit to the quantity of ECSA storage used to queue work. If the Usage Monitor address space is not dispatched in a timely fashion, then many work elements can exist concurrently before being processed. Such work is queued in ECSA storage until it is transferred to the Usage Monitor repository.

If ECSA storage is exhausted and the system provides storage from CSA, the provided storage is freed and the program usage event data is lost. Avoid using storage with less than 16 MB available, to prevent the exhaustion of all common storage.

Data from program usage events occurring while this limit has been reached might not be collected.

An active CSA limit setting stays in force unless overridden, even if the Usage Monitor is stopped and restarted.

►►—CSA(*limit*)—◄◄

limit Specifies a number of kilobytes from 0 to 200,000.

If no CSA command is issued after IPL, the default is CSA(0). CSA(0) specifies that the Usage Monitor does not attempt to limit the ECSA storage used by work elements awaiting processing.

Table 7. Examples of using the CSA command

Command purpose	Example code
Limit queuing in ECSA to 50,000 KB (almost 50 MB).	F HSIJMON,CSA(50000)
Do not enable explicit ECSA limit for storing queued data.	F HSIJMON,CSA(0)

D-A - Display output allocation parameters

D-A is used to display dynamic allocation details to be used in the creation of output data files. The data set name, DCB attributes, primary and secondary space quantities, and unit and optional volume serial number are shown.

►►—D-A—◄◄

The following code example displays the current dynamic allocation values.

F HSIJMON,D-A

D-C - Display the counters and statistics

D-C is used to display the Usage Monitor activity and status indicators. The purpose of this command is to assist IBM technical support in problem diagnosis. The meaning of the output generated by this command is not published.

►►—D-C—◄◄

The following code example displays the current value of internal Usage Monitor counters.

```
F HSIJMON,D-C
```

D-D - Display the data set name inclusion and exclusion lists

D-D is used to display the data set name masks in the inclusion list, followed by the data set name masks in the exclusion list.

The inclusion and exclusion lists do not need to be populated in order to collect data. The absence of any entries in the exclusion list means that data collection is not filtered by program library data set names.

►► D-D ◀◀

The following code example displays the current data set name inclusion and exclusion lists.

```
F HSIJMON,D-D
```

D-I - Display the system identifier

D-I is used to display the system identifier, which is written in the output header record. It can be altered by the **SID** command.

►► D-I ◀◀

The following example code displays the current system identifier used by the Usage Monitor.

```
F HSIJMON,D-I
```

D-S - Display the status settings

D-S is used to display several miscellaneous settings. Other commands are used to alter the individual settings, but this command provides a convenient way to list the current values.

►► D-S ◀◀

Place at the end of the HSIJIN file to confirm monitoring settings.

The following example code displays the current values of settings.

```
F HSIJMON,D-S
```

D-T - Display the automatic switch-and-write time setting

D-T is used to display the time-of-day specified for automatic data space switching and consequent writer task creation. When data from after this time-of-day is detected, data collection is automatically switched to a new data space, and write-out of data in the old data space is started.

The UTC or GMT switch time is calculated using local time current at data space creation time. The time when a data space is terminated is set when it is created.

Changes to the system local time offset, such as those caused by a change to daylight saving time, do not alter the UTC or GMT that the current data space is closed. The time of the switch after the next switch is calculated using the new local time.

►► D-T ◀◀

The following example code displays the current automatic switch-and-write time setting.

```
F HSIJMON,D-T
```

D-X - Display the active exclude list

D-X is used to display the active program name mask exclude list. Data is not collected for programs with names that match the mask in any active entry in the exclude list.

►► D-X ◀◀

The following example code displays the current exclude list entries.

```
F HSIJMON,D-X
```

DCB - Set output DCB attributes

DCB is used to set DCB attributes, which are optimal for a specific device type.

►► DCB (

3390
3380
UNKN

) ◀◀

If no DCB command is issued, the default is DCB(3390).

DCB(3390)

Sets the output DCB to

RECFM=VB,LRECL=27994,BLKSIZE=27998

Use when the output device has 3390 compatible geometry.

DCB(3380)

Sets the output DCB to

RECFM=VB,LRECL=23472,BLKSIZE=23476

Use when the output device has 3380 compatible geometry.

DCB(UNKN)

Sets the output DCB to

RECFM=VBS,LRECL=32756,BLKSIZE=0

The system determines the optimal block size for the device used by dynamic allocation. Use when the output device type is not known until allocation time.

Some FTP products do not process a file with RECFM=VBS correctly, even when no records are actually spanned.

DEL - Deleting program mask entries

DEL is used to remove program name masks from filter tables. Both default and user-added entries can be removed. The required operand specifies one or more program name masks.

DEL (*mask*
 , *mask*
 , *mask* ...
 ALL)

mask Specifies a 1 - 8 character program name mask. Any wildcard characters in the mask are treated as literals for the purposes of finding the mask to delete.

ALL Specifies every currently active mask. This mask cannot be specified with any other mask.

Except for short test periods, it is expected that default exclusion masks such as IGG* remain active.

Table 8. Examples of using the DEL command

Command purpose	Example code
Remove all entries, so that all possible programs are monitored.	F HSIJMON,DEL(*ALL*)
Remove exclusion masks to monitor LE and REXX modules.	F HSIJMON,DEL(CEE*,IRX*)
Remove an exclusion mask to monitor the program called CEE.	F HSIJMON,DEL(CEE)

DSN - Setting the data set name prefix

DSN is used to specify the first part of the data set names used for the output files. The prefix is specified in the required operand. The HSIZIN file must contain a DSN command.

You can use symbols in the construction of the data set name prefix. Available symbols include all z/OS static symbols, &SMF, the SMF identifier for the system, and &SYSLPAR, the logical partition name for the system.

►►—DSN(*dsnpref*)—◄◄

dsnpref Specifies a 1 - 26 character data set name prefix. It can contain one or more data set qualifiers, and must not end in a period after any symbol substitution.

Usage Monitor needs RACF ALTER access to the data sets to be able to create them.

The following example code shows how to get output files with names of the form SYS3.HSI.HSIJMON.Dyyyyddd.Thhmsst:

```
F HSIJMON,DSN(SYS3.HSI.HSIJMON)
```

DUR - Set execution duration

DUR is used to specify a fixed short-term execution duration of the Usage Monitor started task. When the specified time has elapsed the Usage Monitor will terminate automatically. The Usage Monitor stop time is calculated by adding the specified duration to the current time when the command is processed.

Any subsequent WRT commands are ignored.

The DUR command is not normally used in standard operations where the Usage Monitor is to remain active until system shutdown. When it is used, it is normally placed in the HSIJIN file to specify a predetermined length of execution for sampling or testing purposes.

►►—DUR(*hmm*)—►►

hmm Specifies a time duration in hour and minute notation. The value must be four decimal digits. The minimum value is 0001 and the maximum value is 2400. The last two digits (mm) must be in the 00 - 59 range.

The following example code instructs the Usage Monitor to stop after 150 minutes.

F HSIJMON,DUR(0230)

EXC - Adding program mask exclusion entries

EXC is used to add program name masks to the exclusion table. The required operand specifies one or more program name masks.

►►—EXC(*mask*
 ,*mask*
 ,*mask*...
 DFLT)—►►

mask Specifies a 1 - 8 character program name mask. If the mask ends in an asterisk only, characters before the asterisk are compared. Otherwise, an exact program name is deemed to have been specified.

DFLT

Specifies every supplied default entry in the exclusion table is to be made active, and all user-added entries are to be removed from the exclusion table. This mask cannot be specified with any other mask.

Except for short test periods, it is expected that default exclusion masks such as IGG* would remain active.

Table 9. Examples of using the EXC command

Command purpose	Example code
Reset the exclusion table to its default status.	F HSIJMON,EXC(*DFLT*)
Exclude the collection of data for Language Environment modules and REXX modules.	F HSIJMON,EXC(CEE*,IRX*)
Exclude the collection of data for the program CEE.	F HSIJMON,EXC(CEE)

IDD - Deleting data set name inclusion entries

IDD is used to remove data set name masks previously added by the **IDS** command.

►►—IDD(*mask*)—◄◄

mask Specifies a 1 - 44 character data set name mask. Any wildcard characters in the mask are treated as literals for the purposes of finding the mask to delete.


The following example code deactivates the SYS3.LINKLIB inclusion mask.

```
F HSIJMON,IDD(SYS3.LINKLIB)
```

IDL - Control idle work element usage

IDL is used to control whether the Usage Monitor uses idle work elements. When the data in a work element has been processed, the element is normally freed in order to return the storage to the system. Enabling idle elements means that processed elements are retained on the idle chain. This chain is used before acquiring more storage when a new work element is needed.

Enabling idle elements reduces system storage management cost. The storage used by idle elements is included in the storage limit set by the **CSA** command.

►►—IDL()—◄◄

Y Specifies the Usage Monitor retains processed elements for reuse, subject to the CSA limit setting.

N Specifies that all processed work elements are to be freed.

If no IDL command is issued after IPL, then the idle chain is used. IDL(Y) is the default setting.

IDS - Adding data set name inclusion entries

IDS is used to supply data set name masks, which specify data set names to be excluded from exclusion processing. Program usage data fetched from data sets with names matching inclusion masks, is collected without reference to the data set name mask exclusion list.

Inclusion masks only affect data collection if there are active exclusion masks. An inclusion mask is normally expected to match a subset of data set names, which would match an exclusion mask.

►►—IDS(*mask*)—◄◄

mask Specifies a 1 - 44 character data set name mask. If the mask ends in an asterisk only characters before the asterisk are compared. Percent signs in the mask indicate that any character in that location is considered a match. If the mask contains a slash character (/), the value is considered to be a UNIX path name mask rather than a data set name mask.

You can use the following example code if your intention is to not collect program usage data for data sets with a high-level qualifier of SYS3, except for SYS3.LINKLIB. SYS3.LINKLIB is the only data set with a high-level qualifier of SYS3 for which program usage data is to be collected.

```
XDS(SYS3.*)  
IDS(SYS3.LINKLIB)
```

JAC - Set job account collection status

JAC is used to specify if the Usage Monitor is to consider the account code of jobs significant when aggregating data. The Usage Monitor normally aggregates data based on the program name, the job name, and the user ID. This setting is used to add the job account, truncated after 20 characters, to the aggregation key.

Do not instruct the Usage Monitor to collect and preserve all job account codes if they are not important to the administration of your system. Collecting and preserving job accounts can significantly increase data volumes.

A change to this setting does not take effect until the next data space repository switch.

►► JAC(

Y
N

) ◀◀

Y Specifies that job account codes are used.

N Specifies that job account codes are ignored.

If no JAC command is issued after IPL, then job accounts are not used. The default is JAC(N).

JID - Control the preservation of batch job identifiers

JID is used to control whether all batch job identifiers are to be preserved or not. Normally usage data for each program is aggregated by job name and user ID with only the most recent job identifier being retained. JID provides the option of keeping all batch job identifiers so that the number of jobs using a program can be counted, and usage can be attributed to specific individual jobs. Job identifier aggregation for started tasks and TSO user sessions is always equivalent to JID(N) and is not affected by this setting.

►► JID(

Y
N

) ◀◀

Y Specifies that batch job identifiers should not be overlaid and that different batch job identifiers should prevent data aggregation.

N Specifies that normal aggregation by job name and user ID is to proceed without considering job identifier differences.


The default setting of JID(N) applies each time the Usage Monitor is started.

JNM - Control the collection of job names

JNM is used to specify whether the Usage Monitor collects the names of jobs which use programs or not. If the names of jobs which use the various programs

are not considered to be important, you can dispense with the collection of these names. The advantage of not collecting individual job names is the reduction in processing times and data volumes caused by the aggregation of data into fewer records. When individual job names are not collected, usage is summed over broad address space categories, such as JOB, STC, TSO, and SYS. The total usage counts collected by the Usage Monitor for each program are not affected by this setting.

A change to this setting takes effect at the next data space repository switch.

►►JNM()►►

Y Specifies that the name of each job running a program is to be collected.

N Specifies that only a broad address space category of each job running a program is to be collected, instead of the individual job name.

If no JNM command has been issued since IPL, then job names are collected. JNM(Y) is the default.

LLC - Link list correction

LLC is used where sites make a number of dynamic link list changes. This command updates the HSIJMON data to point to the correct load library. Use this command only if you enable dynamic link list updates, which alter the relative concatenation numbers of persisting libraries.

►►LLC()►►

Y A BLDL is performed at write time by the writer task and, if found, the data set name is overlaid. To avoid performance problems, especially during system shutdown, ensure that LLA remains active until the Usage Monitor has terminated.

N Do not check for dynamic list updates.

If no LLC command is issued after IPL, then the default setting of LLC(N) is current.

LPA - Set link pack area program monitoring status

LPA is used to specify whether the monitoring of programs in the Link Pack Area (LPA) is to occur or not. All types of LPA are included in this category.

►►LPA()►►

Y Specifies that LPA program usage is to be monitored.

N Specifies that LPA program usage is not to be monitored.

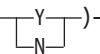
If no LPA command is issued after IPL, then LPA program usage data is collected. LPA(Y) is the default setting.

PRE - Collect usage for long running programs

PRE is used to specify if the Usage Monitor is to collect usage for programs which started before the current collection cycle. Without this data collection a Usage Monitor collection cycle will have no usage data for programs which started running before the cycle started and remain running when the cycle ends. If a job or task runs for more than two days, most days will not have any usage recorded for the main program unless this additional data collection is enabled.

When the additional data collection is enabled, previously fetched programs resident in the regions of started task and batch job address spaces where SMF interval recording is active have usage recorded in each collection cycle which encompassed the end of at least one SMF interval.

This setting can affect usage figures. For example, the main program of a constantly running task can accrue a usage count of around 30 over a month even though it was really only used once for an extended period.

►►PRE()►►

Y Specifies that usage for previously running programs is to be collected.

N Specifies that usage for previously running programs is not to be collected.

The default setting of PRE (Y) applies each time the Usage Monitor is started.

PRI - Set the data set space primary allocation

PRI is used to specify the primary space allocation quantity in tracks. It is used for output data set allocations.

►►PRI(*trks*)►►

trks Specifies a number of tracks from 0 to 150,000.

If no PRI command is issued, the primary space allocation is 750 tracks. The Usage Monitor uses the RLSE space allocation attribute.

The following example code sets the primary space allocation to 900 tracks.

```
F HSIJMON,PRI(900)
```

PRS - Set registered software activity data collection status

PRS is used to specify if the Usage Monitor is to output records containing information about the activity of registered software. Registered software uses the system Register service. The data contains information about the usage of registered software, and information about software registration settings from the PARMLIB member IFAPRDxx.

A change to this setting does not take effect until the next data space repository switch.

►► PRS ($\begin{array}{c} \text{Y} \\ \text{N} \end{array}$) ◀◀

Y Specifies that registered software information is collected and output.

N Specifies that registered software information is neither collected or output.

If no PRS command is issued after IPL, then registered software data is collected.
PRS(Y) is the default.

REF - Refresh Usage Monitor settings

REF is used at any time to reset Usage Monitor settings according to commands in the HSIZIN file, without stopping and starting the Usage Monitor. The detailed results of the refresh operation are written to the HSIZMSG file.

The processes of a refresh operation include:

- Verify that HSIZIN is still allocated.
- Open HSIZIN.
- Set the program exclusion list to the default list.
- Deactivate all data set exclusion list elements.
- Deactivate all data set inclusion list elements.
- Process the commands in HSIZIN.
- Close HSIZIN.
- Issue either HSIZ059I or HSIZ060I, as appropriate.

►► REF ◀◀

The following example code changes Usage Monitor settings to updated values from HSIZIN.

```
F HSIJMON,REF
```

SEC - Set the data set space secondary allocation

SEC is used to specify the secondary space allocation quantity in tracks. It is used for output data set allocations.

►► SEC(*trks*) ◀◀

trks Specifies a number of tracks from 0 to 150,000.

If no SEC command is issued, the secondary space allocation is 300 tracks. The Usage Monitor uses the RLSE space allocation attribute.

The following example code sets the secondary space allocation to 600 tracks.

```
F HSIJMON,SEC(600)
```

SID - Set the Usage Monitor system identifier

SID is used to override the system identifier contained in the output header record. The SMF system identifier is used as a norm, but an override enables the data from separate systems to be differentiated in all instances where duplicate SMF

identifiers are in use. Symbols can be employed in the construction of the system identifier. Available symbols include all z/OS system symbols, &SMF, the SMF identifier for the system, and &SYSLPAR, the logical partition name for the system.

►►—SID(*sid*)—►►

sid Specifies a string which is to be resolved to an identifier 1-4 bytes in length.

Table 10. Examples of using the EXC command

Command purpose	Example code
Set the output system identifier to PROD.	F HSIJMON,SID(PROD)
Set the header record system identifier to the current LPAR name. The LPAR name must not exceed four characters in length.	F HSIJMON,SID(&SYSLPAR)

SIZ - Set the data space repository size

SIZ is used to specify the maximum number of entries that the data space repository can hold.

►►—SIZ(*entries*)—►►

entries Specifies a number of entries from 100 to 6,000,000.

If no SIZ command is issued, a data space capacity of 200,000 entries is used. Each entry occupies 272 bytes. As each data space page has data placed in it for the first time, that page must be backed physically by the system. When a data space is full, a repository switch is triggered automatically. A repository switch also occurs when data stamped after the switch time is detected, or when a manual switch is registered by the SWI command.

The following example code sets the size of future data spaces to 1,000,000 entries.
F HSIJMON,SIZ(1000000)

SJS - Controlling spawned job suffix preservation

When a spawned address space is created by a unit of work with a job name that is shorter than eight characters, the system appends a sequence digit in the 1 to 9 range to the job name, and this becomes the job name of the spawned address space. This approach means that the usage of programs generated by jobs with a specific name can be logged under as many as ten different job names. The system-generated job names usually do not assist in identifying the source of the work because there is often no other reconciliation data which also uses these generated names.

The SJS setting can be used to remove the spawned sequence number suffix so that all usage events for programs are logged under the original job name, resulting in fewer Usage Monitor records and reduced processing time. If the spawning job name is eight characters long and ends in a digit in the 1 to 9 range, then activity in spawned address spaces (but not the original address space) can be reported under a job name which is only the first seven characters of the original job name. If this is likely to present a problem then use SJS(N).

►► SJS ($\begin{array}{|c|} \hline Y \\ \hline \text{---} \\ \hline N \\ \hline \end{array}$) ◀◀

Y Spawned job name suffix digit is truncated.

N No editing is performed of spawned address space job names.

SJS(Y) is the default if no SJS command has been issued since the Usage Monitor started.

SWI - Switch to a new data space repository

SWI causes a new data space repository to be created and used for subsequent data collection. A writer task processes the data contents of the data space that is being used at the time that the SWI command is issued.

The SWI command has no operands. It is invalid in the HSIZIN initial command file. As well as the switch caused by an explicit SWI command, automatic switches occur when a repository becomes full, and when data stamped after the switch time is detected. The SWI command might be rejected if the writer task is busy.

►► SWI ◀◀

The following example code manually switches to a new repository.

```
F HSIJMON,SWI
```

UID - Control the collection of user details

UID is used to specify whether the Usage Monitor collects the identifiers and names of users who use programs or not. If the details of users who use the various programs are not considered to be important, then you can dispense with the collection of this information. The advantage of not collecting user information is the reduction in processing times and data volumes.

When user information is not collected, the user ID data item remains blank, and user names are not output, regardless which UNM setting is current. The total usage counts collected by the Usage Monitor for each program are not affected by this setting.

If you want program usage attributed to individual users but do not want the names of users to be retained, use UID(Y) and UNM(N).

A change to this setting does not take effect until the next data space repository switch.

►► UID ($\begin{array}{|c|} \hline Y \\ \hline \text{---} \\ \hline N \\ \hline \end{array}$) ◀◀

Y Specifies that details of each user using a program are to be collected.

N Specifies that details of each user using a program are not to be collected.

If no UID command is issued after IPL, user details are collected. UID(Y) is the default.

UNK - Set the unknown event collection switch

UNK is used to specify whether events with incomplete data are to be collected or not. The database content is not affected. Collecting extra data is useful in determining why some usage events are not captured. It must be set only when requested by IBM support.

►►—UNK(☐N☐Y)—►►

Y Specifies that the "unknown" events are to be collected.

N Specifies that the "unknown" events are not to be collected.

If no UNK command is issued after IPL, the unknown events are not collected. UNK(N) is the default setting.

UNM - Set user name collection status

Software security packages, such as RACF, have a name field for each user ID defined to the system. The Usage Monitor collects the user ID (up to eight characters long), and the contents of the name field (up to 20 characters long), as part of the data collection performed when programs are used. UNM is used to specify whether the names of users collected from the security package are output. The output of the user ID is controlled by the **UID** setting. This setting is checked by the writer task when the data in a data space repository is being processed for output.

►►—UNM(☐Y☐N)—►►

Y Specifies that collected user names are written to the output file.

N Specifies that collected user names are discarded.

If no UNM command is issued since after IPL, then user names are collected. UNM(Y) is the default.

UNT - Set the data set allocation unit

UNT is used to specify the allocation unit to be used for output data set allocations.

►►—UNT(*unitname*)—►►

Unitname

Specifies a 1 - 8 character long unit name.

If no UNT command is issued, SYSALLDA is used.

The following example code sets the allocation unit to WORKDA.

```
F HSIJMON,UNT(WORKDA)
```


USS - Set UNIX program monitoring status

USS is used to determine if the programs retrieved from Hierarchical File System (HFS) files are to be monitored.

►►—USS($\begin{array}{c} \text{N} \\ \text{Y} \end{array}$)—►►

Y Programs fetched from HFS files are to be monitored.

N Programs fetched from HFS files are not to be monitored.

If no USS command is issued after IPL, the programs retrieved from HFS files are not monitored. USS(N) is the default setting.

VOL - Set the data set allocation volume

VOL is used to specify the allocation volume to be used for output data set allocations. The explicit nomination of a specific volume is necessary when there are no PUBLIC or STORAGE volumes in the allocation unit pool.

►►—VOL(*volume*)—►►

volume specifies a 1 - 6 character long volume serial number.

If no VOL command is issued, a specific volume is not explicitly requested. You must then have PUBLIC or STORAGE volumes in the public allocation pool, unless the data sets are managed by SMS.

The following example code sets the allocation volume to SCR001.

```
F HSIJMON,VOL(SCR001)
```

WRT - Set the automatic switch-and-write time of day

WRT is used to specify a time-of-day to end data collection for the current data space, and automatically switch to a new data space. The data write-out for the closed data space is also initiated at the same time. These events are triggered when data from after the specified time is detected.

The UTC or GMT switch time is calculated using the local time when the data space is created. The time that a data space is terminated is set when it is created. Changes to the system local time offset, such as those caused by a change to daylight saving time status, do not alter the UTC or GMT time that the current data space is closed. The time of the switch, after the next switch, is calculated using the new local time.

►►—WRT(*hhmm*)—►►

hhmm Specifies a 24-hour time-of-day in hour and minute notation. The value must be four decimal digits. The first two digits (hh) must be in the 00 - 23 range. The last two digits (mm) must be in the 00 - 59 range.

If no WRT command is issued, the automatic switch time of midnight is used. That is, WRT(0000) is the default.

The following example code sets the automatic switch-and-write time to 10 minutes before midnight.

```
F HSIJMON,WRT(2350)
```

XDD - Deleting data set name exclusion entries

XDD is used to remove data set name masks which were added by the XDS command. XDD can also deactivate entries from the default exclusion list that was automatically created by the Usage Monitor.

►► XDD(*mask*) ◀◀

mask Specifies a 1 - 44 character data set name mask. Any wildcard characters in the mask are treated as literals for the purposes of finding the mask to delete.

The following example code deactivates the SYS3.* exclusion mask.

```
F HSIJMON,XDD(SYS3.*)
```

XDS - Adding data set name exclusion entries

XDS is used to supply data set name masks which specify data set names to be excluded from data collection. Program usage data for programs fetched from data sets with names matching exclusion masks is discarded. When the captured data set name has been matched to an inclusion mask set by the IDS command, the data is collected without reference to the exclusion mask list.

►► XDS(*mask*
 ,*mask*
 ,*mask*...
 DFLT) ◀◀

mask Specifies a 1 - 44 character data set name mask. If the mask ends in an asterisk, only characters before the asterisk are compared. Percent signs in the mask indicate that any character in that location is considered a match. If the mask contains a slash character (/), the value is considered to be a UNIX path name mask rather than a data set name mask.

DFLT

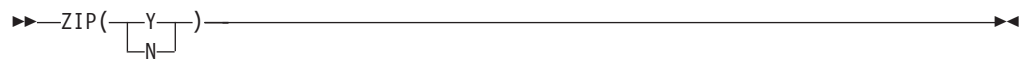
Specifies that all entries in the default data set name exclusion list automatically created by the Usage Monitor which have been deactivated by XDD commands are to be reactivated. The status of masks added by IDS and XDS commands is not altered.

The following example code excludes program usage data from collection for programs fetched from data sets with a high-level qualifier of SYS3.

```
F HSIJMON,XDS(SYS3.*)
```

ZIP - Set the compressed output data switch

ZIP is used to control whether the writer task is to compress output data or not. Compressing the output data reduces data volume, in turn reducing data transfer time and storage space requirements.



Y Specifies that output data is to be compressed.

N Specifies that output data is not to be compressed.

If no ZIP command is issued, then compressed data is output. ZIP(Y) is the default setting.

Monitoring usage in CICS regions

The CICS Transaction Server for z/OS performs much of its program management outside of the contents supervisor framework that most applications use. For the Usage Monitor to accurately detect and record the use of programs in a CICS region, you must customize each CICS region where you require detailed program usage monitoring.

To prepare a CICS region to enable detailed monitoring, you must install the following components:

- A CICS global user exit (GLUE) program
- An enabling program to activate this user exit program
- An entry in the program list table (PLT) that triggers the enabling program

The customized JCLLIB library contains the following sample jobs that you can copy and use in your customization:

- The HSIENAX member contains a sample job to translate, assemble and bind the enabling program.
- The HSISPLTX member contains a sample job to create a PLT with the required entry to trigger the enabling program. If you use this sample job, verify the name of the enabling program and the PLT suffix before you submit the job.

The CICS program monitoring facility does not support releases earlier than CICS Transaction Server version 3, release 2. Different releases of the CICS Transaction Server require different versions of the GLUE program. You must ensure that the correct version of this program is used for each CICS region. You must also take care when upgrading regions to a later release of CICS so that the correct version of this module will be used with the newer software. The following table lists the required GLUE programs for different versions of the CICS transaction server.

CICS Transaction Server release	Exit program
Version 3, release 2	HSIZFTC0
Version 4, release 1	HSIZFTC1
Version 4, release 2	HSIZFTC2

When you implement this CICS Transaction Server customization, the Usage Monitor can collect and record data related to program name and data set name. The collected data is subject to the Usage Monitor program name and data set name selection and exclusion filters. You can stop data collections from all HSIZFTCx GLUE programs with the CIC(N) Usage Monitor setting. CIC(Y) is the default setting if you do not issue a CIC Usage Monitor command. If you want to access more detailed CICS data such as particulars of transactions and the end users involved, a specialized CICS monitor such as IBM Tivoli OMEGAMON XE for CICS on z/OS is required.

Customizing a CICS region to provide usage data

Procedure

1. Copy the appropriate HSIZFTCx global user exit (GLUE) program from the SHSIMOD1 library to a DFHRPL library of the CICS region.
2. Customize and submit the HSIENAX job to create a program that enables the HSIZFTCx exit program:
 - a. Customize the sample job for translating, assembling, and binding the enabling program that is provided in the HSIENAX member in the customized JCLLIB library. For convenience, you can name this program HSIENAx, where *x* is the same suffix character as the suffix of the HSIZFTCx program that it enables.
 - b. Check that the name specified in the PROGRAM operand of EXEC CICS ENABLE statement is the name of the enabling program.
 - c. Check that the name specified in EXEC CICS ENABLE PROGRAM statement is the name of the of the GLUE program
 - d. Link the HSIENAx enabling program into the same DFHRPL library where you copied the .HSIZFTCx GLUE program.
 - e. Submit the HSIENAx job.
3. Add an entry in the following format to the active program library table (PLT) of the CICS Transaction Server to install the HSIENAx module:
DFHPLT TYPE=ENTRY,PROGRAM=HSIENAx
Place the entry before the DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM entry so that it loads early during CICS initialization to minimize the need for program resource definitions.
4. Ensure that the PLTPI setting for the CICS region specifies your newly updated PLT.
5. Optional: Use the HSITAGP tagger program to tag non-vendor application programs that you want to be identified in usage reports.

Results

When you complete this task, the use of programs that are given control by various mechanisms in the CICS Transaction Server are attributed to the CICS region address spaces that invoke them.

What to do next

You can stop data collection from all HSIZFTCx GLUE programs with the CIC(N) Usage Monitor setting. The CIC(Y) option is the default if you do not issue a CIC Usage Monitor command.

Importing Inquisitor data

The Inquisitor Import reads data from Inquisitor scans, where the data is filtered and matched to products. The filtered, matched data is then copied to the Repository tables where it can be viewed and queried by the Analyzer reporting utility.

Related concepts:

“PLX parameter of the Inquisitor program” on page 2

The **PLX** parameter can reduce the time it takes to scan and process different SIDs that are completely shared and are, therefore, identical. When you set **PLX=Y**, the Inquisitor Import detects libraries that are mirrors or libraries that have not changed and quickly processes scans of these shared SIDs.

Related tasks:

“Collecting scanned libraries with the Inquisitor for z/OS” on page 1

The Inquisitor is a program that scans and collects information about partitioned data set (PDS) and partitioned data set extended (PDSE) program libraries. The Inquisitor Import program takes the collected data as input to form the basis of your software inventory.

Running the Inquisitor import

The HSIQIM job in the JCLLIB library performs the Inquisitor import. This job is generated from the HSISCUST post-installation customization job.

About this task

Run-time for the HSIQIM job depends on the number of modules to be imported into the database Inquisitor tables. Because the processing is memory-intensive, run the HSIQIM job during off-peak periods.

If the HSIQIM job processes the same data that is generated by the Inquisitor scan for a specified system, the job terminates with an error to indicate that the input file is a duplicate input file.

Procedure

1. In the HSIQIM job, update the following parameters, according to your requirements:

- **FULLREMATCH**: Set to N (no) to skip import from scanned libraries where no member directories have changed since a previous Inquisitor import to the same Repository. Set to Y (yes) to import and match process all libraries.

When **FULLREMATCH** is set to Y, an extra check is performed on libraries in the repository. For a given system ID (SID), existing libraries in the repository are marked as deleted unless the libraries are found in the scanned Inquisitor file. For shared libraries (where **PLX=Y**), the scanned Inquisitor file can be from any SID that belongs to the sysplex. For non-shared libraries (where **PLX=N**), the scanned Inquisitor file must be from the same SID. The deletions include products, libraries and modules.

- **PRODUCTONLY**: Set to N (no) to import all modules, including unidentified modules. Set to Y (yes) to import only matched modules.

The settings for both of these parameters influence the duration of the import process.

2. Optional: Review and modify other parameters, as required, in the PARMLIB member.
3. Submit the HSIQIM job.

Import filters and matching

When you import data collected by the Inquisitor, the import procedure reads the data and filters and matches the data before copying the data to the Repository tables.

The Inquisitor Import loads data and performs the following tasks:

1. Reads Inquisitor data generated from Inquisitor scans. To exclude importing specific libraries, the Inquisitor data is filtered against a set of supplied Inquisitor filter tables. These Inquisitor filter tables are updated monthly, together with the knowledge databases. The filtering excludes, for example, the ISV distribution libraries.
2. Matches load modules to best fitting products at the version, release, and modification (VRM) level. Best matches for modules are found based on module names and sizes, and information in the Global Knowledge Base (GKB) and Local Knowledge Base (LKB). Temporary scorecard tables are used to hold all the possible scorecards for modules in a given library while they are matched.
3. Loads matched load modules, including matching information, into the Repository tables. Data from the Repository tables are now ready for viewing or reporting using the Analyzer reporting facility.
4. Aggregates usage data for rediscovered modules in the Repository tables.

The Inquisitor Import uses memory intensively in order to efficiently match many Knowledge Base scorecards to library modules. The maximum memory requirements depend on the number of modules in a library of an Inquisitor import file, and the number of scorecards in the GKB and LKB that affect the processed libraries. To estimate a requirement, allow 5M +1.5k per module. For example, for an Inquisitor file containing a maximum library size of 30000 modules, the requirement is approximately $5M + (0.0015 \times 30000) = 50M$.

TPARAM parameters

The TPARAM parameters that you specify for Inquisitor import define what data is included in the import.

COMMIT=

Default is 1000. Number of records stored before issuing a COMMIT.

DSN= DB2® location. Value assigned, as defined in job HSISCUST

FILTERSCHEMA=

Inquisitor Import filter qualifier. Name of qualifier is &GKBZSCHM_IQF7

FULLREMATCH=

Default is N, which means import and match processing will be skipped for scanned libraries that have had no member directory changes since a previous Inquisitor Import and into the same Repository. Y means that all libraries will be imported and matched.

GKBSCHMA=

Global Knowledge Base qualifier for z/OS. Name of qualifier is &GKBZSCHM_GKB7

GKUSCHMA=

Global Knowledge Base qualifier for z/OS UNIX. Name of qualifier is &GKBZSCHM_GKU7

LKBSCHMA=

Local Knowledge Base qualifier for z/OS. Name of qualifier is &REPZSCHM_LKB7

LKUSCHMA=

Local Knowledge Base qualifier for z/OS UNIX. Name of qualifier is &REPZSCHM_LKU7

PRODUCTONLY=

Default is N, which means all modules, including unidentified modules, are loaded into the Repository. Y means only modules that have been matched to known products are loaded into the Repository, meaning, application modules are excluded.

REPSHEMA=

Repository qualifier. Name of qualifier is *&REPZSCHM*.

Importing usage data

The Usage import job imports data generated by the Usage Monitor and aggregates usage data for discovered or identified modules in the Repository tables in the database.

The HSIUIMP job in the JCLLIB library performs the Usage import. This job is generated from the HSISCUST post-installation customization job. Because run-time for this job depends on the volume of usage data to load, run this job during off-peak periods.

Usage data files can be either outputs from the Usage Monitor or condensed outputs from the ZCAT utility. These output files can be concatenated as a single input in the job. When you are priming a new repository, use a single small file as input when loading usage data for the first time.

If the HSIUIMP job processes the same data that is generated by the Usage Monitor for a specified system, the job terminates with an error to indicate that the input file is a duplicate input file.

TPARAM parameters**COMMIT=**

Default is 1000. Number of records stored before issuing a COMMIT.

DSN= Database location. Value assigned, as defined in job HSISCUST.

REPSHEMA=

Repository qualifier. Name of qualifier is *&REPZSCHM*.

Related tasks:

“Collecting usage data with the Usage Monitor” on page 18

The Usage Monitor is a server address space that runs as a started task. Work is queued to the Usage Monitor from all address spaces where programs are used. The Usage Monitor can move captured data into the data space repository and can also write accumulated program usage to a sequential file. The Usage Monitor runs APF authorized and is nonswappable.

Activating the Automation Server

The Automation Server discovers new data sets and processes them by starting a set of predefined actions that associate the data with data set name masks that form a catalog search. This search determines if any data set names matching the mask are to be processed.

Automation Server overview

The Automation Server provides the ability to select data sets if you have data set names that are variable, such as those created by the Usage Monitor, which have low-level qualifiers containing time stamps.

The Automation Server runs as a started task in its own address space.

The user ID for the Automation Server must have an OMVS segment and a UID, or there must be a default UID configured.

The Automation Server issues a system-wide ENQ to ensure that there is only one instance of it in a z/OS image. A single instance of the Automation Server continuously references all data sets, catalogs, and volumes that are accessible from all systems in a sysplex so it is unnecessary for the Automation Server to run on more than one system.

Input control statements define the processing to be performed by the Automation Server. There are two types of control statements, *action* statements and *DSN* statements:

action Action statements name the template member which forms the basic input for the action to be performed when a relevant data set is newly discovered by a catalog search. They have optional operands to specify time-of-day, day-of-week, day-of-month, and month-of-year restrictions.

DSN DSN statements provide a data set name mask to be associated with the preceding action statement. There can be many DSN statements after each action statement.

There are currently two types of action statement:

FTP Starts the FTP utility to perform a file transfer.

For the FTP action, the template member is read and, after symbol substitution processing, is written to the file defined by the INPUT DD statement. The INPUT file is allocated to a temporary data set. The FTP program is attached as a subtask and scans the INPUT file to process the FTP requests. The report messages it generates are written to the OUTPUT file.

Upon completion of the FTP subtask, the Automation Server examines the completion code. If the program ends normally with a zero return code, the Automation Server deems the action to have been successful and updates the action status in the HSIACDS file so the action is not repeated for this data set.

If the FTP program abends, the Automation Server deems the action to have failed. A failed transfer is tried again at a later time. A retry is subject to specified scheduling constraints. The OUTPUT FTP report file contains information to track the exact cause of a transfer failure.

JOB Submits a batch job.

For the JOB action, the template member is read and, after symbol substitution processing, is written to the file defined by the INTRDR DD statement. This file is directed to the internal reader used by the system, and the jobs submitted by the Automation Server become available for JCL conversion as soon as the INTRDR file is closed, or another JOB card image is found by the reader.

The Automation Server deems all JOB submissions successful, so there are no retries. Any failure should be investigated using the appropriate procedures used by your installation.

Note: The job stream in a JOB action template member may define more than one job.

The Automation Server does not check template member records for either FTP or JCL validity.

Before initiating an action for a detected data set, the Automation Server attempts to exclusively allocate the data set. If the data set is in use, the action is not performed and the awaiting retry status is stored in the control data set. The action is performed when the data set is next found to be available for use within any specified scheduling restrictions. Dynamic allocation failures due to other reasons do not inhibit the action.

Running the Automation Server

To run the Automation Server, you must create a control data set, configure the Automation Server as a started task, design request control statements, and exclude data sets from processing.

Creating the Automation Server control data set

To create the Automation Server control data set, use member HSIASALC in the JCLLIB. This member is generated from the HSISCUST post-installation customization job.

Procedure

1. Allocate sufficient space for the Automation Server to handle the workload required by the installation. One 96 byte record (including the 52 byte key) is required for each data set processed by the Automation Server.
2. Create the control data set by running the HSIASALC job that contains the IDCAMS JCL and control statements.
3. In the HSIACSD ddname, allocate the VSAM KSDS control data set to the Automation Server.

Copying the started JCL task to a library

The Automation Server is run by the HSIJAUTO job in the JCLLIB library that is supplied in the SHSIPROC data set. To start the HSIJAUTO job as a started task, you must customize the JCLLIB member and copy it to an authorized PROCLIB library.

Procedure

1. Set values for the following parameters in the HSIJAUTO task in the JCLLIB library:
 - **HSI:** Set high-level qualifiers for the installation target libraries months
 - **HSIINST:** Set high-level qualifiers for the &HSIINST..PARMLIB data set created by the HSISCUST job.
 - **ACDS:** Set the data set name of the Automation Control Data Set (ACDS).
2. Copy the HSIJAUTO member from the JCLLIB library to an authorized PROCLIB library.

Files used by the Automation Server:

Several files must be available for use by the Automation Server.

STEPLIB

Load library containing the product software. Not required if Tivoli Asset Discovery for z/OS is installed into the system link list.

HSIACNTL

Partitioned data set containing fixed length 80 byte records. Member HSIAPARM of this partitioned data set contains the Automation Server control statements that specify the actions to be performed. For each action in the HSIAPARM member, there is a corresponding member of the same name containing the template data for that action. The template data is made up of JCL or an FTP command stream containing symbolic references, to be resolved by the Automation Server when the action is performed.

HSIACDS

A VSAM KSDS control data set used by the Automation Server.

HSIAMSG

Specifies the message report file for the Automation Server. Initialization statements, error messages, and activity logging messages, are written to this file.

SYSPRINT

Specifies the message report file for Language Environment.

SYSOUT

Specifies the message report file for Language Environment.

OUTPUT

Specifies the message report file for the FTP program. The contents are determined by the FTP program installed in the system.

INPUT

Specifies a fixed length 120 byte record file containing FTP commands read by the FTP program. The FTP commands are written to this file before the Automation Server FTP action is performed.

INTRDR

Specifies a fixed length 80 byte record file to be directed to the internal reader used by the system. The Automation Server writes a job stream to this file whenever a JOB action is to be performed.

Designing request control statements

Automation Server action requests are specified in the HSIAPARM member of the SHSIPARM file.

Syntax rules

Syntax rules are as follows:

- Records with an asterisk in column 1 are comments.
- Blank records are comments.
- A parameter record has one or more parameters, each with a value specified within parentheses after the parameter name.
- The first parameter specifies the statement type.
- All parameters must begin before column 72.
- Blanks can be used before and after parameter names, parentheses, and parameter values.
- Continuations on to subsequent records are not possible.

Statement syntax

Action statement

Each statement requests that an action is performed for a data set when it matches an associated data set name mask, and is detected for the first time. An action is performed once for each match, but the presence of a data set triggers the action for each specified data set name mask it matches.

Action statements have several optional operands to provide control over when Automation Server processing is to occur.

These operands can specify:

- time-of-day window
- day-of-week control string
- day-of-month window
- month-of-year control string

When all these constraints have been satisfied, the Automation Server searches the catalog for data sets with names that match the masks associated with an action.

Data set name mask statement

Each data set name mask statement associates the specified data set name mask with the preceding action statement. It is invalid for the HSIAPARM member to begin with a data set name mask statement. When a data set with a name matching the specified mask is first located, the action specified in the preceding action statement is triggered.

The data set name mask of NULLFILE is an exception. When a data set name mask with this exact value is processed by the Automation Server, a catalog search is not performed, but the associated action is triggered as if a new cataloged data set matching the mask has been located. Automation Server symbols for the data set name, and for the first qualifier of the data set name have values of the 8 byte string NULLFILE. Use the data set name mask of NULLFILE to trigger scheduled actions which do not depend on the creation of a particular data set.

Action statement syntax



action FTP or JOB

template

Name of a member in the HSIACNTL file.

TIME This operand is optional, and the default is TIME(0000-2400), which specifies no time-of-day constraint.

hhmm-hhmm

Specifies a time-of-day range. Each *hhmm* value is four contiguous decimal digits that specify a time-of-day using the 24 hour clock. The minimum value is 0000 and the maximum value is 2400; the last two digits must not exceed 59. The two values are separated by a hyphen. Zero or more additional blanks are also permitted. The first *hhmm* specifies the time-of-day window start, while the second specifies the time-of-day

window end. The window includes times which are after the window start and before the window end. However, if the second *hlmm* value is lower than the first, the window includes times which are after the window start or before the window end.

WEEK This operand is optional. The default is WEEK(YYYYYYY), which specifies that the *action* can be run on every day of the week.

wkflags

Specifies a single contiguous 7 byte string, consisting of the uppercase characters Y or N. Each Y or N corresponds to a day of the week depending on its position in the string; the first corresponding to Sunday, the last to Saturday. If the character corresponding to a day of the week is N, the action is not processed on that day.

NOTB This operand is optional. The default NOTB(1) specifies that the monthly window starts on the first possible day of the month. NOTB means "not before".

d1 Specifies a one or two digit decimal number in the 1-31 range. This number denotes the first possible day of the month on which the action is permitted.

NOTA This operand is optional. The default NOTA(31) specifies that the monthly window extends to the last day of the month. NOTA means "not after".

d2 Specifies a one or two digit decimal number in the 1-31 range. This number denotes the last possible day of the month on which the action is permitted.

MNTH

This operand is optional. The default value enables processing in every month of the year.

monthflags

Specifies a single contiguous 12 byte string, consisting of the uppercase characters Y or N. Each Y or N corresponds to a month of the year depending on its position in the string; the first corresponding to January, the last to December. If the character corresponding to a month of the year is N, the action is not processed in that month.

DSN statement syntax

►►—DSN(data-set-name-mask)—►►

DSN Data set name.

data-set-name-mask

Specifies a data set name mask pattern which does not exceed 44 characters in length, and is used by the Catalog Search Interface. The generic match mask for a single character is the percent sign. The generic match mask variable number of characters is the asterisk. A double asterisk can be used to match a variable number of data set name qualifiers. The catalog search is restricted to entry type A non-VSAM data sets and entry type H generation data sets.

Control statement examples

Example 1:

Files created by the Usage Monitor undergo two independent processes, both within the 8:00 p.m. to 11:30 p.m. window. They are processed by a

job based on the JCL contained in member HSIJOB1, and are separately transferred to a z/OS system using the FTP commands in member HSIFTP1. All members are pointed to by the HSIACNTL ddname.

```
* TRANSFER USAGE MONITOR FILES TO Z/OS SYSTEM
JOB(HSISJOB1)  TIME(2000-2330)
DSN(USER.OMU*.D*.T*)
FTP(HSIFTP1)  TIME(2000-2330)
DSN(USER.OMU*.D*.T*)
```

Example 2:

Files created by the Usage Monitor are to be imported to the appropriate database.

```
* PERFORM USAGE MONITOR IMPORT
JOB(HSISUIMP)
DSN(USER.UMON.*.*)
```

In this example HSIUIMP contains the necessary JCL to run Usage Import on a z/OS system.

Note: The JCL can route the job to any connected NJE node, or specify an affinity to any system sharing the SPOOL. You do not need to run the job on the z/OS system where the Automation Server is running. The template name, HSIUIMP in this example, does not need to match the job name submitted by the Automation Server action.

Example 3:

A job stream stored in member WED2MNTH is to be submitted unconditionally on the second Wednesday of every month.

```
* RUN MONTHLY JOBSTREAM ON THE SECOND WEDNESDAY OF EVERY MONTH
JOB ( WED2MNTH )  WEEK ( NNNYN )  NOTB ( 8 )  NOTA ( 14 )
DSN ( NULLFILE )
```

Automation Server symbol processing

Whenever an action is performed, the contents of the template member are written to an appropriate output file. Each 80 byte record is written unchanged, unless symbol substitution is required. If an ampersand character is present in a record from the template member, the system symbol substitution routine ASASYMBM is called to process the record before it is written. You can use more than one symbol in a record. If an ampersand character does not denote the start of a recognized symbol, then that part of the data remains unchanged. Symbols available for use in template members include all z/OS system symbols and symbols defined locally by the Automation Server. Most Automation Server local symbols are derived from the catalog entry data set name which, when discovered, triggers the instance of the action.

System symbols supplied by the operating system, as well as the &SMF and &SYSLPAR symbols supplied by the Automation Server, are available for use in the HSIAPARM member. The &SYSLPAR symbol might resolve to a null string if the system is running in a virtual machine.

Automation Server local symbols are provided in the following table:

Table 11. Automation Server local symbols

Symbol	Description
&SMF	System SMF identifier.
&SYSLPAR	System LPAR name.

Table 11. Automation Server local symbols (continued)

Symbol	Description
&DATASETNAME.	The entire data set name.
&QUAL1.	The first qualifier of the data set name.
&QUAL2.	The second qualifier of the data set name.
&QUAL3.	The third qualifier of the data set name.
&QUAL4.	The fourth qualifier of the data set name.
&QUAL5.	The fifth qualifier of the data set name.
&QUAL6.	The sixth qualifier of the data set name.
&QUAL7.	The seventh qualifier of the data set name.
&QUAL8.	The eighth qualifier of the data set name.
&QUAL9.	The ninth qualifier of the data set name.

Example:

The data set triggering a JOB action is IBMUSER.IQ.ZIP. As a result, JCL DD statements referencing the data set in a template member can be represented as shown in this example:

```
//*-----***
/* Sample JCL demonstrating the use of Automation Server local***
/* symbols derived from the data set name.                      ***
/*-----***
//BR14      EXEC   PGM=IEFBR14
//DD1       DD     DSN=&DATASETNAME.,DISP=SHR
//DD2       DD     DSN=&QUAL1..&QUAL2..&QUAL3.,DISP=SHR
```

Both JCL DD statements would be resolved by symbol substitution to:
DSN=IBMUSER.IQ.ZIP,DISP=SHR

This is the **DSN=** JCL statement output to the internal reader.

As symbol substitution is performed before the job is submitted, z/OS system symbols that cannot be used in batch job JCL, can be used in the Automation Server templates. The symbols are resolved using the system executing the Automation Server, which may not be the system where the submitted job executes.

Starting and stopping the Automation Server

When you install the Automation Server as a started task, you can run operator commands to start it and stop it. This task requires RACF security access.

Procedure

1. Assign RACF CONTROL access to the VSAM data set that is configured for the user ID that is assigned to the Automation Server.
2. Issue the system **START** command to start the Automation Server.
3. Issue the system **STOP** command to stop the Automation Server running as a started task or from running a batch job.

Excluding data sets

You can exclude data sets from Automation Server processing. To exclude a data set, it must have a record in the Automation Server control data set, with an indication in the record that the data set is already processed.

In the HSIAPARM member you define actions to be performed, and supply data set name masks specifying the data sets to be processed by the Automation Server. Data sets with these name patterns might already exist and have been processed before the Automation Server was implemented.

To exclude a data set, the name of the data must satisfy a selection mask pattern. To implement the exclusion, you can use the Automation Server data set name scouting program. The HSISCUST post-installation job creates the HSIASSCT member in the JCLLIB data set. Run the HSIASSCT job to start the scouting program.

The program reads the HSIAPARM member, searches the catalog for every specified data set name mask, and writes a record for each data set that it discovers. The job then sorts the records into key order and copies them into the VSAM control data set. Every record loaded into the control data set in this way indicates a specific action with a status of complete.

If you want to continue processing some of the data sets, you can manually delete them from the sequential data set before the data is copied into the control data set.

Automation Server control data set maintenance

A record is kept for every data set processed by the Automation Server in the Automation Server control data set, ACDS. The purpose of this record is to prevent the repeated processing of a data set for the same data set name mask. As records accrue, the size of the data in the ACDS continues to grow.

If a processed data set is deleted, or a data set name mask is removed from the set of masks processed by the Automation Server, then there is no reason to keep a record of that data set in the ACDS. The Automation Server performs a cleanup cycle for the ACDS on a daily basis. This cleanup cycle consists of reading the ACDS sequentially, and deleting records for data sets which have not been found by catalog search. This is based on the relevant data set name mask in the current calendar month, or in the prior calendar month.

As with most VSAM data sets with ongoing record insertion and deletion activity, it is advisable to periodically reorganize the ACDS.

Chapter 2. Running the utilities provided with Tivoli Asset Discovery for z/OS

Tivoli Asset Discovery for z/OS provides utilities that you run to perform routine functional tasks to maintain the product lifecycle.

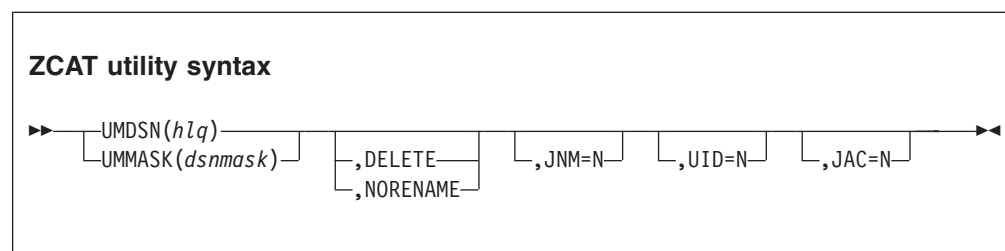
Condensing usage data with the ZCAT utility

The ZCAT utility concatenates and condenses Usage Monitor data sets and generates a file that is then processed by the Usage Import program. When you condense the data produced by the Usage Monitor program, you can save storage space and improve the performance of the Usage Import program.

The Usage Monitor started task produces at least one usage data set per day. You can design a work flow that runs the ZCAT utility on the data sets on a weekly, fortnightly, or monthly basis before the Usage Import program processes them. Running the ZCAT utility on a weekly basis is useful, but depends on the amount of data that is produced and processed at your site. The Usage Monitor program collects detail about which job, account ID, and user ID are using each module of a particular library on a specified date. This information is output into multiple files that are produced on a daily basis. The ZCAT utility condenses the files in the following manner:

- Usage data across multiple files is condensed to a monthly granularity, as are the records stored in the Repository database.
- Redundant records in files and records that are not stored in the database, are omitted.
- Optionally, condensation can apply to user IDs, job names, or account ID details.
- The ZCAT output file is compressed and ready to be transmitted for Usage Import processing.

The following diagram shows the syntax of program parameters to run the ZCAT utility.



Mandatory parameters

The UMDSN or the UMMASK parameter must be specified.

UMDSN(*hlq*)

hlq is the Usage Monitor data set high-level qualifier. When the **UMDSN** parameter is specified, ZCAT concatenates all data sets having names of *hlq*.Dyyyyddd.Thhmsst where *yyyyddd* and *hhmsst* are the timestamp patterns of data sets produced by the Usage Monitor. The *hlq* can contain wildcard characters of percent or asterisk. The percent character denotes a

TADŹ.**.D%%%%%%.T%%%%%%%%

UMMASK(*dsnmask*)

dsnmask is the full dsn mask search criteria. It can be used to search for a pattern of files that differ from the files produced by the Usage Monitor. This parameter is useful if the files produced by the Usage Monitor have been renamed, but still need processing. Specifying UMMASK(hlq.D%*%*%*%*%*%T%*%*%*%*%*) is equivalent to specifying UMDSN(hlq)

Optional parameters

One or more optional parameters can follow the mandatory parameters.

DELETE

Delete the input data sets after the output data set is successfully generated. The default is to retain the input data sets, which are renamed by default.

NORENAME

Do not rename input data sets from hlq.D*.T* to hlq.D*.S* after the output data set is successfully generated. The default is to rename these input data sets to stop them being reprocessed by the ZCAT utility. Use this option only to rename the data sets before further ZCAT processing. This option stops double counting of usage data. This parameter is automatically set when UMMASK is used.

The RENAME option is ignored, if **DELETE** is also specified.

Optional condensation parameters

Improvements in performance and data storage space are gained by using the ZCAT utility options to carry out further condensation of data, ignoring data differences that are not important at your site, and do not appear in your regular reporting. You can still point the Usage Monitor File Detail Report to the saved archive of the concatenated detail file (ZCATDETL), or to the Usage Monitor output files. ZCATDETL is produced by the ZCAT utility.

$$J_{NM} = N$$

Condense data for different job names to one of the following generic names based on the job type: -STC-, -JOB-, -TSO- or -SYS-. The default is to retain the job name, and to condense data that belongs to the same job name only.

UID= N

Condense data for different user IDs, which are converted to blank. The default is to retain the user ID, and to condense data that belongs to the same user ID only.

JAC=N

Condense data for different job account codes which are converted to blank.
The default is to retain the job account code, and to condense data that belongs
to the same job account code only

Note: Due to the various consolidating options, records that are ignored in the ZCATOUT data set are still written to the ZCATDETL output data set, which can be retained for archiving.

DD statements

ZCATOUT

Specifies the name of the ZCAT output data set. This data set can then be used as the input to the Usage Import program, where usage details are imported into the database. If the ZCATOUT DD card is omitted, ZCAT by default writes to a data set having the name hlq.Dyyyyddd.Uhhmmsst (U instead of T implied by the high level qualifier (hlq) option for input data sets), where yyyyddd and hhmmsst refer to the date and time timestamp of the first processed input data set.

ZCATDETL

If the ZCATDETL DD is allocated, all records are written to this data set. This data set includes any non condensed and non diagnostic data that is not written to the ZCATOUT data set. It enables the Job name, user ID, and job account details (which are ignored due to ZCAT options and are not written to the ZCATOUT file) to be archived into the detail file.

The ZCATDETL and ZCATOUT data sets are compressed by the ZCAT utility.

```
//ZCAT      EXEC PGM=HSICZCAT,PARM='UMDSN(TADZ.**),JNM=N'
//STEPLIB   DD  DISP=SHR,DSN=TADZ.V750.SHSIMOD1
//ZCATOUT   DD  DSN=&SYSUID..TADZ.ZCATOUT,
//           DISP=(NEW,CATLG),UNIT=SYSDA,SPACE=(CYL,(50,50),RLSE),
//           DCB=(DSORG=PS,RECFM=VB,LRECL=27994,BLKSIZE=27998)
//ZCATDETL  DD  DSN=&SYSUID..TADZ.ZCATDETL,
//           DISP=(NEW,CATLG),UNIT=SYSDA,SPACE=(CYL,(50,50),RLSE),
//           DCB=(DSORG=PS,RECFM=VB,LRECL=27994,BLKSIZE=27998)
//DEBUG     DD  SYSOUT=*,HOLD=YES,LRECL=200 //SYSPRINT DD  SYSOUT=*,HOLD=YES
//SYSERR     DD  SYSOUT=*,HOLD=YES
```

In this example, all data sets having names of TADZ.*.D%%%%%%.T %%%%%%% are processed due to the **UMDSN** parameter. The condensed output is written to sysuid.TADZ.ZCATOUT where the SYSUID system symbol is the user ID of the person submitting the job. This file is then transmitted for Usage Import processing. The **JNM=N** parameter instructs the utility to condense job names and ignore the original job name distinction. All valid records are written to the ZCATDETL DD card sysuid.TADZ.CATDETL, which is then archived for reference purposes.

Summarizing usage data with the Usage Summary utility

The Usage Summary utility summarizes usage data in the repository. The process deletes detailed usage records and creates monthly summary records by reducing the number of DB2 rows used to represent your old data. After each time that you run the Usage Summary utility, the usage data is aggregated to update the asset tables.

To minimize space utilization and improve SQL query performance, it is recommended that you keep detailed module usage data for the last three months and summarize all detailed module usage data older than three months. It is also recommended to delete summarized module usage data older than 18 months. Please see job HSISUDEL (Usage Deletion) for more details.

If you have not run the Usage Summary job for some time, then select a period of a few months at a time, in order to keep the run times down to a reasonable time.

Running the Usage Summary utility

To run the Usage Summary, use the job HSIUSUM, in the JCLLIB. This job is generated from the HSISCUST post-installation customization job.

TPARAM parameters

COMMIT=

Default is 1000. Number of records stored before issuing of COMMIT.

DSN= DB2 location. Value assigned, as defined in job HSISCUST.

SUMBY=

Use this parameter to condense the usage data.

SUMBY=1

Data is summarized at the Product, LPAR, Period, Module ID, and Job Type level.

User ID and job ID distinctions are ignored. Instead of Job IDs, events are attributed to Job Types (BATCH, TSO, DB2...).

SUMBY=2

Data is summarized at Product, LPAR, Period, Job ID, User ID level.

Load module and program names are ignored.

SUMBY=3

The rules for SUMBY=1 and SUMBY=2 apply.

Data is summarized by Product, LPAR, Period, Job Type.

KEEPDETAIL=

Default is 2. Number of months prior to the current month for which usage records are not summarized. Prior usage records are summarized. If KEEPDETAIL=0 is specified, all usage records, excluding those records for the current month, are summarized.

FIRSTDATE=

Start of the first date range. This is in the form YYYYMM. Only complete months are chosen.

LASTDATE=

End of the last date range. This is in the form YYYYMM.

Note: The date range of summarization is inclusive of the month specified in the FIRSTDATE and LASTDATE parameters.

MINUSAGETHRESHOLD

Default is 1000. Sets a value for Usage Summary to ignore summarization of usage records. If this parameter is set to 1000, then any product with a usage count of 1000 or less for any given month, does not have its usage records summarized. This allows you to view the usage records for low usage products.

REPSHEMA=

Repository qualifier. Name of qualifier is &REPZSCHM.

Deleting usage data with the Usage Deletion utility

You use the Usage Deletion utility to delete detailed, summarized, and aggregated usage data for a specified period for all systems in the repository. Each time you run the utility, usage data is aggregated to update the asset tables.

To minimize space utilization and improve SQL query performance, keep no more than 3 months of detailed module usage data and 13 months of aggregated product usage data. If you want to keep more than 3 months of detailed module usage data, run job HSIUSUM (Usage Summary) to summarize the detailed module usage data older than three months.

If you do not run the Usage Deletion utility for some time, select a period of a few months, in order to keep the run times down to a reasonable time.

Running the Usage Deletion utility

To run the Usage Deletion, use the job HSIUDEL, in the JCLLIB. This job is generated from the HSISCUST post-installation customization job.

TPARAM parameters

COMMIT=

Default is 1000. Number of records stored before issuing of COMMIT.

DSN= DB2 location. Value assigned, as defined in job HSISCUST.

KEEPDETAIL=

Default is 2. Number of months prior to the current month for which detailed and summarized module usage data are kept. KEEPDETAIL=0 means all detailed and summarized module usage data excluding those from the current month are deleted.

KEEPAGGR=

Default is 12. Number of months prior to the current month for which aggregated product usage data are kept. KEEPAGGR=0 means all aggregated product usage data, excluding those from the current month are deleted.

FIRSTDATE=

Start of the first date range. This is in the form YYYYMM. Only complete months are chosen.

LASTDATE=

End of the last date range. This is in the form YYYYMM.

Note: The date range of deletion is inclusive of the month specified in the FIRSTDATE and LASTDATE parameters.

REPSHEMA=

Repository qualifier. Name of qualifier is &REPZSCHM.

SID= System Identifier of system for which usage should be deleted. Specify SID=ALLSIDS to delete usage data for all SIDs.

Note: If KEEPDETAIL is set to a value, then FIRSTDATE / LASTDATE will be ignored. If detailed usage data are to be deleted within a certain date range, then comment out KEEPDETAIL and define dates for FIRSTDATE / LASTDATE. For further details, please see comments described in job HSIUDEL.

Deleting a specific system with the System Deletion utility

It can be necessary to delete data for a specified system to reconcile data and delete unreferenced records. The System Deletion utility deletes discovery, usage, and hardware data for a specified system.

Because libraries that are shared with other systems are not deleted, data for a specified system can become outdated.

You can also use the System Deletion utility to delete a system that was accidentally imported into the repository or to delete a system that is decommissioned.

Running the System Deletion utility

To run the System Deletion, use the job HSISLDEL, in the JCLLIB. This job is generated from the HSISCUST post-installation customization job.

TPARAM parameters

DSN= DB2 location. Value assigned, as defined in job HSISCUST.

REPSHEMA=

Repository qualifier. Name of qualifier is &REPZSCHM.

SID= System Identifier of system to be deleted.

Listing high-level qualifiers for the Usage Monitor utility

Tivoli Asset Discovery for z/OS collects large amounts of usage data. The High-level Qualifier Listing for the Usage Monitor utility creates a list of high-level qualifiers for the products that are to be identified.

Following are some examples that exclude all usage, but include some usage for the specified high-level qualifiers:

```
XDS(*)  
IDS(DB2.*)  
IDS(IMS.*)  
IDS(CICS.*)  
IDS(SYS1.*)
```

The high-level qualifier listing process is automated in the Inquisitor Import job. The high-level qualifier listing is written to a data set, and this data set is concatenated to the HSIZIN control file for the Usage Monitor program.

Running the High-level Qualifier Listing for the Usage Monitor utility

To run the High Level Qualifier for the Usage Monitor utility, use the job HSISLLST in the JCLLIB. This job is generated from the HSISCUST post-installation customization job.

TPARAM parameters

DSN= DB2 location. Value assigned, as defined in HSISCUST.

REPSHEMA=

Repository qualifier. Name of qualifier is &REPZSCHM.

Updating the TPARAM table

The TPARAM table in the repository can be set to an inconsistent state due to failures in jobs that update the repository tables. You can reset a parameter in the TPARAM table to rectify this inconsistent state.

To run the TPARAM table update, use the job HSISTPRM, in the JCLLIB. This job is generated from the HSISCUST post-installation customization job.

SYSIN parameter

```
UPDATE &REPZSCHM.TPARAM SET FVALUE = '0' WHERE FKEY = 'PROCRUN';
```

Tagging unidentified products with the Product Tagging utility

The Product Tagging utility helps you to identify software that has not been predefined in the Global Knowledge Base (GKB). You can also use the Product Tagging utility to supersede GKB entries without changing the contents of the GKB.

Product tagging process

Product tagging is a manual process where you must provide the product name, the vendor, and the location of the programs. The Product Tagging utility uses the same method as the Inquisitor program to scan the programs and records the results in dedicated program members.

The SYSIN file contains the control statements that describe which licensed programs are to be tagged. This file contains the program name, vendor name, product identifier, and product version. The program library which contains the software to be tagged is allocated to the SYSLIB file.

You can have only one set of identifying attributes for each program name. If conflicting attributes are found for one or more program names, the Product Tagging utility issues a message and stops.

Information about all discovered programs relating to the nominated product is compiled into a single object module. This module is written to the scanned library allocated to SYSLIB file or to the program library allocated to the optional HSIREDIR file. Using the HSIREDIR file, you can nominate to keep all tag data separate from licensed program software. The HSIREDIR file data sets must be included in the standard Inquisitor scan processing, even if these data sets contain no other program.

The tag data members created by the Product Tagging utility are recognized by the Inquisitor (by their SSI value) during normal program library scanning. The Inquisitor program extracts the tag data from the member contents and writes it to an output file. The Inquisitor import process uses these program tags to maintain entries for the programs in the local knowledge base. The match engine can then accurately identify the tagged product level, regardless of which library the product is deployed to and which system the data is collected from.

Each time you run the Product Tagging utility, it scans a single library and tags a single software product, or optional feature of a product. For products with multiple program libraries, each library is processed in a separate job or step. To ensure effective software identification by the match engine as it processes each

library, use the **OPTION** statement to differentiate the identification entities between the different libraries of a product. Do not tag distribution libraries.

You can override the default output member name of **@HSIPTAG** by specifying a **TAGMEM** statement. All output members from the Product Tagging utility are flagged with an SSI value of **X'D7E3C1C7'**, which is **'PTAG'** in EBCDIC.

If there is no preexisting member of the same name, the Product Tagging utility creates a new program member to contain the tag data. If a member exists, the new tag data is added to the existing data that relates to other products or optional features. Any data relating to the same software identified by {**VENDOR + PRODUCT + OPTION + VERSION**} is replaced. The data relating to each software piece resides in its own control section. Tag data members contain no executable code, and are bound with the only loadable attribute. These data members are bound as reentrant, with a residence mode of **ANY**, to minimize the impact of being placed in a library which is loaded into the Link Pack Area.

To erase the effects of processing with the Product Tagging utility, delete the tag data members which are identified by their SSI value. If you are using ISPF, employ the **SORT SSI** member list command.

The software processed when you run the Product Tagging utility has a key of {**VENDOR + PRODUCT + OPTION + VERSION**}. If non-key data items, such as the values specified in the **PPNUM** or **LICENSED** statements are incorrect, you can correct them by fixing the input statement values and rerunning the utility. This action replaces all non-key tag data. However, if a key data item is incorrect, it will not be erased by running the Product Tagging utility with the correct data.

If you are processing libraries that are not dedicated to a single licensed program, use member name masking to prevent tagging programs not related to that product. Some installations place multiple software products in a combined common library. If the products are tagged before they are combined, you must use different tag data member names.

Product tagging job and control statements

You use the **HSISPTAG** job in the **JCLLIB** to run the Product Tagging utility. This job is generated from the **HSISCUST** post-installation customization job. You input control statements using the **SYSIN** file.

General syntax rules are:

- Fixed length, variable length, and undefined record formats are processed.
- Short records are extended to 72 bytes of data, with blanks if necessary.
- Only the first 72 bytes of data for each record are processed by the Tagger.
- Records beginning with an asterisk are treated as comments and do not alter continuation status.
- The first nonblanks of a statement must identify the statement type.
- One or more blanks must follow the statement type.
- A statement with no value or operand specified is invalid.
- For statement types other than **SELECT**, the specified value is deemed to start with the first nonblank after the statement type name.
- Statements can be placed in any order. All statements are processed before any tagging activity commences.

- SELECT is the only statement type which can be supplied more than once in an input file.
- SELECT is the only statement type which can be continued over more than one record.

The following table lists all of the statement types that you can use with the Product Tagging utility:

Table 12. Product Tagging utility statement types

Statement Type	Value	Default Value	Required	Maximum length
VENDOR	Vendor name	-	Yes	30 bytes
PRODUCT	Product name	-	Yes	50 bytes
PPNUM	Licensed program number	blanks	No	16 bytes
OPTION	Optional feature name	BASE	No	30 bytes
VERSION	Software level	-	Yes	8 bytes
LICENSED	Separately licensed feature? (YES or NO)	NO	No	3 bytes
TAGMEM	Output member name	@HSIPTAG	No	8 bytes
SELECT	Program name filter	PGM(*)	No	8 bytes per mask

SELECT is not a value-oriented statement type. It has operands which have values specified in parentheses. The PROGRAM or PGM inclusion operand can be abbreviated to P. The XPROGRAM or XPGM exclusion operand can be abbreviated to XP.

The Tagger stops parsing a SELECT record and the current statement continues on to the next record whenever a continuation character is encountered. Valid continuation characters are plus and hyphen. A continuation cannot occur within an operand name, or a value mask.

SELECT syntax



member-mask

A string up to 8 bytes in length, representing one or more possible member names of a PDS or PDSE. Use a percent sign to indicate that any single character is to be considered a match in the exact location of the compared character string. Use an asterisk to indicate that any zero or more characters are a match.

Example 1

A company called ISV has created a build of several programs (build 97) it is developing under the Swisho4U brand. The data sets created by this build have their own disk volume called BLD097. The tag data is to be redirected to a data set dedicated for this purpose.

```

//STEP1 EXEC PGM=HSITAGP
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=S4U.LOADLIB,DISP=SHR,UNIT=3390,VOL=SER=BLD097

```

```
//HSIREDIR DD DSN=S4U.TAGLIB,DISP=SHR
//SYSIN DD *
VENDOR ISV
PRODUCT Swisho4U
VERSION BUILD097
/*
```

Example 2

The BigBiz Inc. data center is about to deploy the contractor data processing component for Version 4 Release 2 of its internally developed human resources application called HU-MAN. The software is tagged in its own library, but the default tag member name is not used in case it is later loaded into a program library common to several applications. All programs in HU-MAN have names beginning with HU, but the contractor component is the only component which has program names beginning with HUC. The relevant program library can be accessed by using the catalog.

```
//TAGRUN EXEC PGM=HSITAGP
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=HUMAN.V4R2M0.LOAD,DISP=SHR
//SYSIN DD *
VENDOR BIGBIZ INCORPORATED
PRODUCT HU-MAN Human Resources Management
OPTION Contractor Handling
VERSION 04.02.00
TAGMEM HUMANT@G
SELECT PGM(HUC*)
/*
```

Example 3

Version 1.5 of the product MVSBL0AT from MiscWare has been deployed on a system which has a dedicated tag data library called SYS2.TAGLIB. Link list programs for the product have been placed in SYS2.LINKLIB and ISPF application modules have been placed in SYS2.ISPLLIB. The product does not have optional features, but only the base component installed. All the installed programs have names beginning with MVSBL. The OPTION statement is used to ensure that the contents of each library can be identified by the Match Engine.

```
//STEP1 EXEC PGM=HSITAGP
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=SYS2.LINKLIB,DISP=SHR
//HSIREDIR DD DSN=SYS2.TAGLIB,DISP=SHR
//SYSIN DD *
VENDOR MiscWare
PRODUCT MVSBL0AT
OPTION BASE (Batch)
VERSION 01.05.00
TAGMEM $$OEMTAG
SELECT PGM(MVSBL*)
/*
//STEP2 EXEC PGM=HSITAGP
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=SYS2.ISPLLIB,DISP=SHR
//HSIREDIR DD DSN=SYS2.TAGLIB,DISP=SHR
//SYSIN DD *
VENDOR MiscWare
PRODUCT MVSBL0AT
OPTION BASE (Dialogs)
VERSION 01.05.00
TAGMEM $$OEMTAG
SELECT PGM(MVSBL*)
/*
```

Product tagging examples

Three examples are provided to show the ways that you can use the Product Tagging utility to tag unidentified products.

Example 1

A company called ISV has created a build of several programs (build 97) it is developing under the Swisho4U brand. The data sets created by this build have their own disk volume called BLD097. The tag data is to be redirected to a data set dedicated for this purpose.

```
//STEP1 EXEC PGM=HSITAGP
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=S4U.LOADLIB,DISP=SHR,UNIT=3390,VOL=SER=BLD097
//HSIREDIR DD DSN=S4U.TAGLIB,DISP=SHR
//SYSIN DD *
VENDOR ISV
PRODUCT Swisho4U
VERSION BUILD097
/*
```

Example 2

The BigBiz Inc. data center is about to deploy the contractor data processing component for Version 4 Release 2 of its internally developed human resources application called HU-MAN. The software is tagged in its own library, but the default tag member name is not used in case it is later loaded into a program library common to several applications. All programs in HU-MAN have names beginning with HU, but the contractor component is the only component which has program names beginning with HUC. The relevant program library can be accessed by using the catalog.

```
//TAGRUN EXEC PGM=HSITAGP
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=HUMAN.V4R2M0.LOAD,DISP=SHR
//SYSIN DD *
VENDOR BIGBIZ INCORPORATED
PRODUCT HU-MAN Human Resources Management
OPTION Contractor Handling
VERSION 04.02.00
TAGMEM HUMANT@G
SELECT PGM(HUC*)
/*
```

Example 3

Version 1.5 of the product MVSBL0AT from MiscWare has been deployed on a system which has a dedicated tag data library called SYS2.TAGLIB. Link list programs for the product have been placed in SYS2.LINKLIB and ISPF application modules have been placed in SYS2.ISPLLIB. The product does not have optional features, but only the base component installed. All the installed programs have names beginning with MVSBL. The OPTION statement is used to ensure that the contents of each library can be identified by the Match Engine.

```
//STEP1 EXEC PGM=HSITAGP
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=SYS2.LINKLIB,DISP=SHR
//HSIREDIR DD DSN=SYS2.TAGLIB,DISP=SHR
//SYSIN DD *
VENDOR MiscWare
PRODUCT MVSBL0AT
OPTION BASE (Batch)
```

```

VERSION 01.05.00
TAGMEM $$OEMTAG
SELECT PGM(MVSB*)
/*
//STEP2 EXEC PGM=HSITAGP
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=SYS2.ISPLLIB,DISP=SHR
//HSIREDIR DD DSN=SYS2.TAGLIB,DISP=SHR
//SYSIN DD *
VENDOR MiscWare
PRODUCT MVSBL0AT
OPTION BASE (Dialogs)
VERSION 01.05.00
TAGMEM $$OEMTAG
SELECT PGM(MVSB*)
/*

```

Importing subcapacity reporting data with the SCRT Import utility

The SCRT Import utility reads data created by the IBM Subcapacity Reporting Tool and generates CSV files that you can then import to the repository. You can use the Analyzer to query this data for trending of SCRT data and to compare the data with the corresponding usage data.

Running the SCRT Import utility

To run the SCRT import utility, use the job HSISSCRT, in the JCLLIB. This job is generated from the HSISCUST post-installation customization job.

Data input

DDNAME CSVIN contains the CSV output from the IBM SCRT tool which can be from a data set with DSORG of PS or PO. Binary uploaded CSV files are supported. DDNAME SIDMAP maps duplicate SMFIDs to a unique SID. The SCRT Import utility handles data where the same SMFID is used on multiple machines concurrently.

Data input example

Map SMFID on specific machines to your desired SID. As described in this example, when processing data for CPU serial 11111, SMFID IP01, use SID QIP1, and so on.

```

//SIDMAP DD *
11111-IP01=QIP1
11111-IP02=QIP2
11111-IP03=QIP3
/*

```

CPU serial

5 alphanumeric characters

SMFID

1 to 4 alphanumeric characters

Unique SID

1 to 4 alphanumeric characters. This must be the same as the SID value being used by the Usage Monitor for that z/OS system.

Data output

Several DB2 tables are populated from the data contained in CSVIN, including NODE, NODE_CAPACITY, and PRODUCT_NODE_CAPACITY. Ensure that the CSVIN DD points to the .CSV output file created by the SCRT tool. This may be a DSORG=PO or PS data set.

TPARAM parameters

SSID=

DB2 subsystem name. Value assigned, as defined in job HSISCUST

REPSHEMA=

Repository qualifier. Name of qualifier is *&REPZSCHM*.

GKB=

Global Knowledge Base qualifier. Name of qualifier is *&GKBZSCHM_GKB7*

Capturing historical SMF data with the SMF Scanner utility

You can run the SMF Scanner utility to get historical usage information from existing SMF data. This SMF data enables you to view trending results from before Tivoli Asset Discovery for z/OS is installed.

To start the process you need to run two jobs to capture scanned data (Inquisitor) and historical usage data (SMF Scanner). The output from the SMF Scanner (usage data) can then be processed to produce historical trending.

A sample job HSIIBM can take a file from either the Inquisitor or the Usage Monitor and filter out non-IBM programs. You might use this function when sending data to IBM Support for diagnosis.

The output of the SMF Scanner may also be used as input to HSIIBM. The SMF Scanner only tracks usage of the Job Step EXEC PGM modules and does not include modules that have been invoked from within the task.

Running the SMF Scanner utility

To run the SMF scanner utility, use the job HSISSMF in the JCLLIB. This job is generated from the HSISCUST post-installation customization job.

Extracting data with the XML Export utility

The XML Export utility extracts information in XML format that you can then import into SmartCloud Control Desk.

The extracted information can be either:

- A catalog of the products that are installed in your system.
- A catalog of the products defined in the Global Knowledge Base (GKB).

Running the XML Export utility

To run the XML export utility, use the job HSISKBT, in the JCLLIB. This job is generated from the HSISCUST post-installation customization job.

The output XML file generated from this utility needs to be transferred by FTP to a distributed environment and then loaded into SmartCloud Control Desk. The XML

file must be translated from EBCDIC to ASCII.

TPARAM parameters

SSID=

DB2 subsystem name. Value assigned, as defined in job HSISCUST.

SCHEMA=

Repository qualifier or Global Knowledge Base qualifier.

a) Using the Repository qualifier value means that a catalog of products installed on your site is selected

b) You can also use the Global Knowledge Base qualifier value. This would mean that a catalog of all products defined in the Global Knowledge Base is selected.

Transferring output XML by FTP

The output XML file that is generated when you run the XML Export utility must be transferred by FTP to a distributed environment before you can load it into SmartCloud Control Desk.

Procedure

1. To connect to the host system, in a command line, enter the following command:
`C:\temp ftp host name.`
2. When prompted, enter your user name and password.
3. To set the input to ASCII format, enter the following command:
`ftp > quote type a.`
4. Optional: To transfer non-ASCII characters, enter an ENCODING command before you enter the GET command:
`quote site ENCODING=MBCS MBDATACONN=(IBM-939,UTF-8).` This example specifies encoding for a Japanese codepage.
5. To specify the location of the file to transfer, enter the following command:
`ftp > get 'hsiinst.SWKBT.XML' C:\XML.FILE.`
6. To complete the FTP transfer, enter the following command:
`ftp > exit.`

Compressing and decompressing data sets with the HSIZIP utility

HSIZIP is a utility program that can compress a sequential or partitioned data set into a zip archive, decompress the contents of a zip archive into a sequential or partitioned data set, and report on the contents of a zip archive.

In this context, an archive is a sequential file that contains one or more logical files for the purpose of reducing the space occupied by the data. The archive can serve as a backup and convenient transport format for the data it contains. The Inquisitor and Usage Monitor components usually create zip archives to contain the data that they collect.

The HSIZIP utility has two compress and decompress functions; one for text data and one for binary data.

Text data processing with the HSIZIP utility

The HSIZIP utility processes text data to be compatible with the zip processing utilities available on other platforms.

When compressing a text record, the HSIZIP utility performs the following tasks:

- Translates non-text extended binary coded decimal interchange code (EBCDIC) code points to periods. Only code points in the range from x'40' to x'FE' plus x'0E' (SO) and x'0F' (SI) are deemed to be EBCDIC text characters.
- Translates EBCDIC data to american standard code for information interchange (ASCII) data.
- Appends an ASCII carriage return line feed (CRLF) sequence (x'0D0A') to encode the record extent.
- Compresses the data and writes it to the archive.

Each compressed member is marked as an ASCII text file and the internal attribute value of the central file header is set to 1.

When decompressing text data, the HSIZIP utility performs the following tasks:

- Accumulates data until an ASCII LF (x'0A') is encountered.
- Truncates the trailing ASCII carriage return (CR) (x'0D') if present in accumulated data.
- Translates the ASCII data to EBCDIC and writes the data as a single record.

The translation tables used for conversion between EBCDIC and ASCII that are originally sourced from the EZAESENU member in the SEZATCPX library are reciprocal so that applying one translate table and then the other yields the original data. Consequently, all EBCDIC single byte character set (SBCS) and double byte character set (DBCS) text can undergo a ZIP and UNZIP cycle without corruption.

Binary data processing with the HSIZIP utility

The HSIZIP utility processes binary data in order to preserve record boundaries, while other platforms typically consider binary data to be a byte stream without structure.

When compressing a record of binary data, the HSIZIP utility performs the following tasks:

- Prefixes the data with a multiple virtual storage (MVS) type of record descriptor word (RDW), where the first two bytes contain the length of the record including the RDW, and the third and fourth bytes contain zeros.
- Compresses the data and writes it to the archive.

Each compressed member is marked as a binary file and the internal attribute value of the central file header is set to 0.

When decompressing binary data, the HSIZIP utility performs the following tasks:

- Decompresses 4 bytes from the archive and extracts the record length.
- Decompresses the RDW-indicated length minus 4 bytes and writes it as a record.

During decompression of binary data, the embedded RDWs are checked for validity. If an RDW does not indicate a positive length greater than 4 or does not end with two bytes of zeros, the HSIZIP utility switches to byte stream mode. In byte stream mode, the utility considers data as a stream of bytes without an inherent record structure. If the RDW that fails the validity test is the first four

bytes of the file, the resultant decompression is broadly compatible with the decompression that most other platforms perform and the utility issues an informational message. If the RDW that fails the validity test is not at the start of the file, the utility issues a warning message, sets the final condition code to be greater than zero, but continues processing so that the output data is available for any necessary data recovery activity.

HSIZIP program parameters

The HSIZIP utility can accept up to two program parameters. The first parameter specifies the function the program is to perform and the second parameter can provide a data definition override list for programs that dynamically invoke the utility.

When you invoke the HSIZIP utility as a stand-alone batch program, the PARM value on the EXEC statement specifies the functional request. DD statements define the details of the following files:

- The SYSPRINT report file
- The SYSUT1 input file
- The SYSUT2 output file

You can specify program parameters in the function request in mixed case. The following information describes valid program parameters.

LIST If you specify this parameter, the utility produces a list of the contents of the input archive.

TEST This function is similar to the LIST function except the report contains data from both the local file headers and the central file directory, including file offsets, so that the integrity of the archive can be verified.

ZIP or ZIP=filename.ext

Use this parameter to compress a partitioned data set into an archive where each member is loaded as a separate zipped file within the archive. A sequential input file is processed as a single member stored in the archive under the name specified in the parameter. If no name is specified in the parameter, the name **seq.txt** is used. The data is treated as text.

ADD or ADD=filename.ext

This parameter performs the same function as ZIP except that the output file must be an existing zip archive. The utility writes the compressed data as additional member(s) and prints a report of the original contents of the output archive before it starts to process any new data. The data is treated as text. There is no dependency on the text or binary nature of the existing zipped files in the archive.

UNZIP or UNZIP=filenamemask

Use this parameter to decompress an archive into a partitioned data set and load each zipped file into a separate member. The parameter restores data sets from archives made by the HSIZIP utility with **PARM=ZIP**. If the output data set is sequential, only the first file in the archive is unzipped. You can use the file name mask specification to filter the files to be unzipped.

ZIPBIN or ZIPBIN=filename.ext

Use this parameter to compress a partitioned data set into an archive and load each member as a separate zipped file within the archive. A sequential input file is processed as a single member stored in the archive

under the name specified in the parameter. If no name is specified in the parameter then the name **seq.bin** is used. The data is treated as binary data and no translation is performed.

ADDBIN or ADDBIN=filename.ext

This parameter performs the same function as the **ZIPBIN** parameter except that the output file must be an existing zip archive. The utility writes the compressed data as additional member(s) and prints a report of the original contents of the output archive before it starts to process any new data. The data is treated as binary and no translation is performed. There is no dependency on the text or binary nature of the existing zipped files in the archive.

UNZIPBIN or UNZIPBIN=filenamemask

Use this parameter to decompress an archive into a partitioned data set and load each zipped file into a separate member. The parameter restores data sets from archives made by the HSIZIP utility with **PARM=ZIPBIN**. If the output data set is sequential, only the first file in the archive is unzipped. Use the file name mask specification to filter the files to be unzipped.

The filenames and filename masks that you specify in program parameters must not exceed 128 bytes in length. File name mask matching is case insensitive. The following characters are generic masking characters for filename masks:

- ? (question mark) matches any single character.
- * (asterisk) matches any zero or more contiguous characters.

If the function request is absent or invalid, the utility writes usage notes to the report file. If the request is absent, the utility attempts to run the LIST function.

HSIZIP files

The HSIZIP issues report files, input files and output files.

The HSIZIP utility uses the following files:

- SYSPRINT is a report file. RECFM=VBA and LRECL=137 are used in the DCB.
- SYSUT1 is an input file that describes the data set that contains data to be zipped or the zip archive that contains data to be listed or unzipped.
- SYSUT2 is an output file that contains the results of a compression or a decompression operation. This file is not required by the LIST and TEST functions.

The HSIZIP utility does not support spanned records for any file. The main compression and decompression input and output to archive files uses the queued sequential access method (QSAM) locate mode. Apart from the lack of support for spanned records, an input archive allocated to SYSUT1 can have any valid record format and reside on any device that can be read by QSAM. An archive allocated to SYSUT2 must have variable-length records and support update-in-place processing. In effect, a SYSUT2 file must be an MVS DASD data set that is not also a compressible extended-format data set.

Dynamic invocation of the HSIZIP program by other programs

Other programs can call the HSIZIP utility to perform compression and decompression processing requests. When the HSIZIP utility receives control, it examines the program parameter list and proceeds accordingly.

The first program parameter must begin with a halfword counter indicating the length of the function request text that immediately follows. The format is the same format as the system uses to pass the parameter specified in the PARM operand of the EXEC statement in JCL.

A second program parameter can be specified to override the default filenames used by the HSIZIP utility. If the value of the halfword length indicator at the start of the parameter is not a multiple of 8 or is not less than 256, the HSIZIP utility ignores it. A series of 8-byte file name entries immediately follow the length indicator and each can specify the DD name to use instead of the default name. Set a slot to 8 bytes of zeros to avoid overriding that particular default file name. SYSPRINT, SYSUT1 and SYSUT2 correspond to the sixth, eighth and ninth file name slots respectively.

HSIZIP data set support

The data control block (DCB) attributes of the original data set that the HSIZIP utility compresses are not encoded into the archive. The success of a compress and decompress cycle requires the user to supply suitable DCB attributes for the ultimate destination of the data.

The following points are provided to help you to assess whether the HSIZIP utility can successfully process a data set:

- There is no internal limit on the size of the uncompressed data. The file headers within the archive contain 32-bit counters for the uncompressed and compressed data byte counts. It is important that the low-order 32 bits of the uncompressed file size is recorded accurately in these headers. It is not important if the counter wraps around one or more times other than being able to correctly report the uncompressed files size from the header data.
- The compressed byte count of an archive member cannot exceed the number that can be stored in an unsigned 32-bit binary integer. This limit is a fundamental HSIZIP limit.
- When processing a whole partitioned data set, the file name specified after ZIP= or ADD= is ignored because the member names are used to label the archived files.
- When ZIP processing detects that a PDS member is a zip archive, the member is stored as a byte stream as is without attempting further compression or record boundary preservation.
- ZIPBIN processing of PDS members containing zip archives usually causes the compressed size to be larger than the uncompressed size, due to the inability to further compact the data and the insertion of RDWs to preserve record boundaries. So, if the only non-text data in a PDS is in members which are themselves zip archives, specify ZIP rather than ZIPBIN to minimize the resultant file size.
- When using ADD or ADDBIN, avoid duplicate file names in the resultant archive.
- You can use the ADD and ADDBIN parameters to create an archive with a mixture of text and binary file members.
- The binary or text nature of an unzip process is set by the program parameter and not from the attribute values in the file header.
- When the HSIZIP utility creates a zip archive, the data set name of the input file is stored as the zip archive comment.

- PDS member user data such as system status information (SSI), ISPF statistics, and load module attributes are stored in the comment field of the central file header of the archive member and can be restored during unzip operations.
- Alias members are stored as files with zero bytes. The alias member data is preserved only if the real member associated with the alias member is also processed.
- Use ZIPBIN and UNZIPBIN when processing load module libraries.
- Segment overlay programs are not restored properly, unless the TTRs happen to match, because the TTRs in the segment tables are not updated by the HSIZIP utility.
- The HSIZIP utility cannot restore program PDSE data sets because only the program binder can write to program PDSEs. There is no restriction on data PDSEs.

HSIZIP return codes

When you run the HSIZIP utility, several codes are returned that indicate whether the program ran successfully.

Table 13. HSIZIP utility return codes

Return code	Description
0	Request processed successfully.
2	Warning message issued. The warning is for a condition that does not affect the operation of the current request, but will probably impact on the intended use of the file created by the request.
4	No data was found to process, or an I/O error was encountered.
8	An error occurred. Look at SYSPRINT for more details.
Other	As set by another routine. Look at SYSPRINT for more details.

Chapter 3. Configuring language support

Tivoli Asset Discovery for z/OS includes Japanese language support for MVS™ Message Service (MMS) message information. You can also configure the Analyzer utility to create and view reports in Japanese.

Configuring Japanese messages

To configure messages in the Japanese language, there are two areas to customize. You must compile the Japanese language MMS messages and then you must enable messages in the Japanese language in the language environment.

About this task

Japanese language MMS message information is contained in the hsi.SHSIMJPN program directory. You must compile the message file and then install the most recent system runtime message file. The HSISMCMP job in the JCLLIB library compiles MMS messages. This job is generated by the HSISCUST post-installation customization job.

Procedure

1. Run the HSISMCMP job to compile the MMS files into a system runtime message file, for example the his.MMSJPN99 file. The HSISMCMP job is generated by the HSISCUST post-intallation customization task.
2. Create an entry named **MMSLSTJ9** in the z/OS PARMLIB library, with the following values:
DEFAULTS LANGCODE(JPN)
LANGUAGE LANGCODE(JPN) DSN(SYS2.MMSJPN99) CONFIG(CNLJPN00)
3. Enter the following MVS system command to install the system runtime message file:
SET MMS=J9
4. Optional: Enable Japanese messages for the Inquisitor Import and the Usage Import, Summary, and Deletion components. To configure the language environment, refer to the following documents:
 - For information about the Language Environment MSGFILE and NATLANG options, refer to the *Language Environment Programming Reference* (SA22-7562).
 - For information about specifying Language Environment runtime options, refer to the *Language Environment Programming Guide* (SA22-7561).
 - For information about setting NATLANG(JPN) as an installation default, refer to the *Language Environment Customization* (SA22-7564).

Enabling the Analyzer utility for Japanese

You can configure the Analyzer utility so that you can view and create reports in Japanese.

Procedure

1. In the Analyzer HSIJANLO started task, update the DD statements with the following commands:
//HSICUST hsi.SHSIPARM(HSISANCJ)
//HSINLS hsi.SHSIANL1(HSINLSJP)

2. Optional: Perform the following tasks to customize your Japanese Analyzer reports:
 - a. Copy the **hsi.SHSIPARM(HSISANCJ)** parameter to the **hsiinst.PARMLIB(HSISANCJ)** library.
 - b. Modify the **hsiinst.PARMLIB(HSISANCJ)** library to customize your reports.
 - c. Enter the following command to update the Analyzer HSIJANLO started task:
`//HSICUST hsiinst.PARMLIB(HSISANCJ)`

Configuring the Japanese DB2 subsystem for use with Tivoli Asset Discovery for z/OS

Tivoli Asset Discovery for z/OS is implemented with a Japanese DB2 subsystem that is configured with the MCCSID=939 code page (Japanese extended English).

Notices

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

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

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

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

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 websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

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

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

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

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

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

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.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

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

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 on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.



Printed in USA