CA SOLVE:Operations® Automation

SOLVE Subsystem Interface Guide

Release 11.9

CA Technologies Product References

This document references the following CA Technologies products:

- CA NetMaster® File Transfer Management (CA NetMaster FTM)
- CA NetMaster® Network Automation (CA NetMaster NA)
- CA NetMaster® Network Management for SNA (CA NetMaster NM for SNA)
- CA NetMaster® Network Management for TCP/IP (CA NetMaster NM for TCP/IP)
- CA SOLVE:Access™ Session Management (CA SOLVE:Access)
- CA SOLVE:Central™ Service Desk for z/OS (CA SOLVE:Central)
- CA SOLVE:Operations® Automation

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Provide Feedback

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

If you would like to provide feedback about CA Technologies product documentation, complete our short customer survey, which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Introduction

This section contains the following topics:

SOLVE Subsystem Interface

SOLVE Subsystem Interface is an implementation of IBM's subsystem interface (SSI). The interface provides a general-purpose facility that allows product regions to communicate with other software on a system. It uses an SSIDB to store information such as business application name definitions for CA NetMaster NM for TCP/IP. One SOLVE SSI can serve multiple product regions.

SOLVE SSI provides the following facilities:

- DD SUBSYS
- Event notification facility (ENF) event listener
- Inter-SSI communication
- Packet Analyzer and SmartTrace
- Program-to-Program Interface (PPI)
- SMF record intercept

Products Supported

The SOLVE SSI component provides essential services to the following products:

- CA Mainframe Network Management product family:
  - CA NetMaster FTM
  - CA NetMaster NA
  - CA NetMaster NM for SNA
  - CA NetMaster NM for TCP/IP
- CA SOLVE:Access
- CA SOLVE:Central
- CA SOLVE:Operations Automation product family:
  - CA SOLVE:Operations Automation
  - CA SOLVE:Operations Automation for CICS

Chapter 2: Implementation

This section contains the following topics:

Deployment

SOLVE SSI runs as a started task that was built when your product was installed. The common Install Utility sets up the SOLVE SSI component based on the products you want the component to serve.

For each region that you want to connect to the SOLVE SSI, the interface ID is specified in *one* of the following locations:

- Sub-System Interface ID field of the SSI parameter group in Customizer
- SSID JCL parameter in the region's RUNSYSIN member

Note: For more information about implementing SOLVE SSI, see *Installation Guide*.

When you run your product on multiple systems, you set up a SOLVE SSI on each system. You can set up the SSIs using the following methods:

- Use the Install Utility to set up each of the SSIs.
- Use the Install Utility to set up one SSI, and then replicate the generated data sets such as the SSIDB and SSIPARM.
- Use the Install Utility to set up one SSI, share data sets such as the SSIPARM, and replicate the SSIDB.

Note: If you are upgrading a SOLVE SSI, use your existing SSIDB to retain stored information.

SOLVE SSI as Common Component

The SOLVE SSI is a common component for multiple CA product families and can serve multiple product regions on a system. The following methods are available:

- One shared SSI to serve all product families.
- A separate SSI for each product family (CA Mainframe Network Management, CA SOLVE:Operations Automation, and CA SOLVE:Access).
- A mix of the first two methods, for example, CA SOLVE:Access has its own SSI and CA Mainframe Network Management and CA SOLVE:Operations Automation share an SSI.

How You Share a SOLVE SSI Between All Products

The Install Utility can help you share a SOLVE SSI between all supported products on a system. Products are released by families. Usually, when a product family is released, the release comes with an updated version of the SSI. When you share the SSI, use the latest version. The following process guides you in sharing an SSI:

1. Use the Install Utility that comes with the latest release to set up your SSI.
   a. If there is an existing SSI, use the existing subsystem identifier for the new SSI.
   b. Select the products you want the new SSI to serve.
   c. Complete the setup.
2. Replace the existing SSI with the new SSI. Because you use the same subsystem identifier for the new SSI, product regions that require an SSI connect to the new SSI seamlessly.

How You Configure a SOLVE SSI for a Product Family

If you use a separate SOLVE SSI for each family, you can potentially run up to three instances of the SSI on the same system. Some SSI initialization parameters are restricted to be set only once on a system. The Install Utility can help you set up an SSI for each product family, but the utility can potentially specify a restricted parameter in more than one SSI. You review the parameter members to remove the conflict.

You set up an SSI for a product family by using the Install Utility that comes with that family. In the following example, you have a system that is running CA SOLVE:Operations Automation but has no CA Mainframe Network Management products. You want to add CA NetMaster NM for TCP/IP to the system.

1. Use the Install Utility that comes with CA NetMaster NM for TCP/IP to set up a separate SSI.
   a. Select CA NetMaster NM for TCP/IP as the product you want the SSI to serve.
   b. Complete the setup.
2. Review the parameter members for the two SSIs for CA SOLVE:Operations Automation and CA NetMaster NM for TCP/IP. Referring to the Multiple Instances of SOLVE SSI on the Same System (see page 13) topic, you notice that both SSIs specify the restricted UNIX and XEVNT parameters. Because CA SOLVE:Operations Automation is using an earlier version of SSI, remove those parameters from the member for that SSI.

   You configure the two SSIs to run without conflict on the same system.

Multiple Instances of SOLVE SSI on the Same System

Multiple instances of the SOLVE SSI component can run on the same system. These instances can include SSIs at different gen levels. For example, if you run products from different families on the same system, you can set up a different SSI for each family.

When you use the Install Utility to set up the SSIs, the default configuration enables the SSIs to communicate with each other (initialization parameter XCF=YES (see page 22)). The multiple SSIs works as a logical SSI. A product region connected to one SSI has access to the facilities in the other SSIs.

Some SSI initialization parameters can be set only once within the logical SSI; otherwise, failures can occur. If you plan to run multiple SSIs on the same system, ensure that the following parameters, if necessary, are set in one SSI only:

- IPSECNMI=YES (see page 19)
- PAEESTACK=*stack_job_name* (see page 26)
- PKTANALYZER=YES (see page 29)
- PPI=YES (see page 30)
- SMF=YES (see page 20)
- SNANMI=YES (see page 20)
- UNIX=YES (see page 21)
- XEVNT=YES (see page 22)

The following table provides a matrix that helps you identify the important parameters required within a logical SSI for the products running on a system. Where possible, include the parameters in the SSI at the latest gen level.

Table legend: FTM (CA NetMaster FTM), NA (CA NetMaster NA), SNA (CA NetMaster NM for SNA), TCP/IP (CA NetMaster NM for TCP/IP), SO (CA SOLVE:Operations Automation), SOCICS (CA SOLVE:Operations Automation for CICS), and SC (CA SOLVE:Central)

	FTM	NA	SNA	TCP/IP	SO	SOCICS	SC
IPSECNMI	N/A	N/A	N/A	**NO/YES**	N/A	N/A	N/A
PAEESTACK	N/A	N/A	N/A	*stack_job_name*	N/A	N/A	N/A
PASMFWRITE	**YES**	N/A	N/A	**YES**	N/A	N/A	N/A
PKTANALYZER	N/A	N/A	N/A	**YES**	N/A	N/A	N/A
PPI	N/A	N/A	N/A	N/A	N/A	**YES**	**NO/YES**
SMF	**YES**	N/A	N/A	**YES**	N/A	N/A	N/A
SNANMI	N/A	N/A	N/A	**YES**	N/A	N/A	N/A
UNIX	**YES**	N/A	N/A	**YES**	**YES**	**YES**	N/A
XEVNT	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	N/A

Note: CA SOLVE:Access requires no special parameters, except the standard XCF=YES.

Note: CA NetMaster FTM uses the NMFTP Monitor SSI, which shares the initialization parameters with the SOLVE SSI. When you use the Install Utility to set up the NMFTP Monitor SSI, the correct parameters are configured. The NMFTP Monitor SSI has IPSMFAPIREC=YES, which is mutually exclusive with the UNIX parameter in the same SSI.

Chapter 3: Administering the SOLVE SSI

This section contains the following topics:

Initialization Parameters

SOLVE SSI has various initialization parameters. These control important execution options and facilities.

The initialization parameters are specified in the SSIPARM(SSISYSIN) or SSIPARM(SSIPARMS) (if used) member created during region setup as described in *Installation Guide*. To review the initialization parameters and to add further initialization parameters, open this member.

Note: Parameters can be specified more than once. The last specification is used.

Note: Packet Analyzer and the PPI facility provide several additional parameters.

ABENDCODE={ 4095 | *n* }

Assigns an alternate user ABEND code for SOLVE SSI internal ABENDs.

Default: 4095

Limits: 1 to 4095

ABENDMSG={ NO | YES }

Specifies whether SSI service ESTAE exits write WTO messages if an ABEND is detected. You can use this parameter to find external interface errors, for example, PPI RC 90.

ADDSSID={ YES | NO }

Specifies whether the SSID is defined automatically if an SSID is not found when SOLVE SSI starts. If YES is specified, the SSID is defined automatically. If NO is specified, the SSID must be defined manually in the SYS1.PARMLIB(IEFSSN*xx*) member.

ARM={ NO | YES }

Specifies whether the SOLVE SSI region registers with the Automatic Restart Manager (ARM). If NO is specified, the region does not register with ARM. If YES is specified, the SOLVE SSI region attempts to register with ARM. If the registration fails, an error message is written to the SSILOG but the region starts.

ARMNAME=*elementname*

Overrides the default ARM element name. The default is SVS_*sysnamessid* where *sysname* is the system name and *ssid* is the ID of the SOLVE SSI region. ARM requires a unique element name for each SYSPLEX.

BLIP={ NO | *n* }

Specifies the wake-up interval, which prevents S322 time-outs of the subsystem job. This parameter can be important in a testing environment if SOLVE SSI is running as a batch job. If NO is specified, no wake-up timer is set.

Default: NO

Limits: Valid values of *n* are 1 to 60 minutes.

CDI={ YES | NO }

Specifies whether to enable or disable the Configuration Data Interface (CDI) facility. The facility is an EPS server that accepts requests for and obtains TCP/IP stack configuration data from IBM's Communications Server. CA NetMaster NM for TCP/IP uses this facility.

Default: YES

DAE={ NO | YES }

Specifies whether SOLVE SSI sets the DAE indicator to ON when requesting an SDUMP. This parameter enables these dumps for DAE processing. Use this parameter to stop duplicate Supervisor Call (SVC) dumps.

Default: NO

Note: Current releases of z/OS perform DAE processing regardless of the setting of this parameter.

DDSUBSYS={ NO | YES }

Specifies whether SOLVE SSI provides services needed by the DD SUBSYS facility.

If you are using the DD SUBSYS facility, specify YES.

Default: NO

ENF={ NO | YES }

Controls the setup of the z/OS ENF listener during SSI initialization. If NO is specified, no listener is inserted. If YES is specified, then an ENF listener is inserted and the ENF interface is activated.

Default: NO

ENFADD=*n*

Activates the nominated ENF code. If *n* does not correspond to a recognized ENF code that SOLVE SSI handles, then a warning message is produced during startup. This parameter can be repeated, as necessary.

The valid ENF codes and their initial status are documented in IBM's *z/OS Authorized Assembler Programming Guide*.

Limits: *n* must be in the range 1 through 255.

ENFDEL=*n*

Deactivates the nominated ENF code. If *n* does not correspond to a recognized ENF code that SOLVE SSI handles, then a warning message is produced during startup. This parameter can be repeated, as necessary.

Limits: *n* must be in the range 1 through 255.

ENFARMWTO={ NO | YES }

Activates the old WTO messages that earlier versions of the ENF interface issued for ARM events.

Default: NO

IPSECNMI={ NO | YES }

Specifies whether to enable the IPSec network management interface (NMI) facility. If multiple SOLVE SSIs are active on a system, only one of these SSIs can have the facility enabled.

Default: NO

IPSMFAPIREC={ NO | YES }

Specifies whether to exploit the SMF API (or Network Management Interface API).

Notes:

- This parameter and the UNIX parameter are mutually exclusive. You can set only one to YES in any SSI.
- For CA NetMaster FTM, this parameter is set to YES in the NMFTP Monitor parameter member during setup if you use an IBM TCP/IP stack.

PROMPT={ YES | NO }

Specifies whether a message is written (WTO) to the system operator if the supplied SSID is not found or is unusable. If YES is specified, the operator has a chance to specify a new SSID, request a retry, or end the SSI. If NO is specified, and the SSID is not usable, the SSI is immediately terminated.

SDUMP={ NO | YES }

Specifies whether SOLVE SSI writes a dump to a SYS1.DUMP data set when an ABEND occurs.

The default is NO, and the dump is written in accordance with the JCL (for example, SYSUDUMP or SYSMDUMP).

Specifying YES suppresses any dump specifications (such as SYSUDUMP) in the JCL (except as noted), and forces a dump to a SYS1.DUMP data set. The dump includes symptom strings that aid analysis.

Note: The formatted dump (SSIDUMP) is still written. Sometimes, a system dump (for example, a CANCEL DUMP operator command) is written to a JCL-specified location (such as SYSUDUMP) before SOLVE SSI has a chance to suppress it. Therefore, it is recommended that the SYSUDUMP, SYSMDUMP, and SYSABEND statements be removed from the SOLVESSI JCL.

SMF={ NO | YES }

Specifies whether support for SMF record intercept is enabled. You can enable this support in only one SSI per system.

Note: Set this parameter to YES for the following products only:

- CA NetMaster NM for TCP/IP
- CA NetMaster FTM

Important! SMF record type 119 must be defined in the stack configuration. For more information, see *CA NetMaster Network Management for TCP/IP Installation Guide*.

SMFREPLACE={ NO | YES }

Specifies whether SMF exits are reloaded when SMF=YES. By default (NO), the SSI reloads an exit only if it is newer than the currently loaded one. If you specify YES, the SSI reloads the exits every time the SSI starts up.

Important! The SSI loads the exits into common storage. Specifying YES can lead to a depletion of the common storage under abnormal circumstances. Therefore, in a production environment, do not specify YES unless there are other considerations (for example, intermittent execution of a back-level SSI).

SNANMI={ YES | NO }

Enables and disables the SNA network monitoring. The SNANMI facility can be enabled in only one active SOLVE SSI region.

Default: NO

SOLVEJOB={ * | *jobname* }

Specifies the job names of the regions that can connect to SOLVE SSI. If * is specified, no restriction is placed on the regions that can connect to SOLVE SSI. *jobname* specifies the job name of a region that can connect to SOLVE SSI (it must be a valid z/OS job name). This parameter can be specified up to 16 times to nominate the regions that can connect to SOLVE SSI at once.

SSM={ NO | YES }

Specifies whether to enable support for CA OPS/MVS System State Manager (SSM).

Default: NO

SSMAPPL=*application_name*

Specifies the application name registered with CA OPS/MVS for this product or component.

Limits: Eight characters

Important! Each product or component registers a specific application name with CA OPS/MVS. *Do not change this name.*

The application name registered for SOLVE SSI is NMSSI.

SSMHBI=*heartbeat_interval*

For SSM=YES, specifies how often to send heartbeats to CA OPS/MVS SSM. The heartbeat interval is in minutes. A value of zero indicates no heartbeat.

Initial value: 5

Limits: 0 through 60

STATS={ NO | *n* }

Specifies the interval at which a SHOW SSISTATS command is issued. The output of this command is routed to the SOLVE SSI log only. This can be useful for following trends in SOLVE SSI usage. If NO is specified, no STATS timer is set.

Limits: *n* can be 1 through 60 minutes.

UNIX={ NO | YES }

Specifies whether to enable the UNIX System Services (USS) shell interface.

If you set this parameter to YES, and your operating system cannot access USS for some reason, then you receive a warning message, but the task starts successfully.

If you have multiple SOLVE SSIs active on a system, then set up the USS shell interface in one SOLVE SSI only. For the other SOLVE SSIs, specify NO.

XCF={ NO | YES }

Enables the SOLVE SSI region to communicate with other SSI regions by using XCF. If YES is specified, the SOLVE SSI region attempts to register with XCF using the group name and member name.

XCFGROUP=*groupname*

Overrides the default XCF group name that SOLVE SSI uses when registering with XCF.

Default: ZSOLVE01

XCFMEMBER=*membername*

Overrides the default XCF member name that SOLVE SSI uses when registering with XCF.

Default: *sysnamessid*

XEVNT={ NO | YES }

Specifies whether to include Simple One Shot Event Sender support. Only one SSI started task per z/OS system can have YES specified.

XMS={ * | NO | YES }

Specifies whether SOLVE SSI sets up and uses a cross-memory environment for external interfaces (for example, PPI). The following values can be specified:

*****

(Default) Uses cross memory services if supported by the operating system. This means that z/OS uses cross memory services by default.

NO

Prevents the use of cross memory services. Several SSI facilities (such as ENF, PKTANALYZER, and XEVNT) do not work without these services.

YES

Uses cross memory services. If the environment does not support these services, an error occurs.

With cross memory services, when SOLVE SSI stops, the address space identifier (ASID) is marked as unavailable. Frequent starting and stopping of SOLVE SSI can contribute to the exhaustion of available ASIDs, requiring a system IPL.

If the MVS DIAG*xx* parmlib member specifies REUSASID(YES), enable XMS and start SOLVE SSI with the REUSASID=YES MVS START command parameter.

If the DIAG*xx* parmlib member specifies REUSASID(NO), disable XMS under the following condition: You do *not* use the facilities that require cross memory services, and SOLVE SSI starts and stops frequently (for example, when used for testing). Otherwise, enable XMS.

Execute SOLVE SSI Commands

SOLVE SSI provides several commands that can be used for control and to display statistics. These commands can be issued from the following locations:

- From any system console. Command responses are delivered to the issuing console.
- Any suitably authorized user ID on the connected region. Command responses are delivered to the issuer.
- Internally. The STATS=*n* parameter internally issues a command in the SSI itself. The output of this command is routed to the SSI log only.

SSI Command Descriptions

The following commands are provided:

SHOW SSIEPS

Displays information about EPS connections and end points as seen from this SSI region. The following operands are supported:

DETAIL

Displays additional information about each end point.

LINKS

Displays a list of direct links to other end points.

TOPOLOGY

Displays a topology of all links between all end points.

NOTIFY

Displays a list of notifier end points for each link in the topology display.

STATS

Displays additional statistics lines.

SHOW SSISTATS

Displays statistics about internal SSI pools. This command is useful for tuning and debugging.

SHOW SSIUSERS

Displays a list of all signed-on SSI environments. The list shows the terminal (or console), user ID, and type.

SSI SIGNOFF

Terminates the current command/message environment. This is useful for preventing further receipt of unsolicited SSI messages after issuing some SSI commands.

SSI STATUS

Displays the status of the SSI. The display includes the version number and PUT level.

SSI STOP

Stops the SSI. This command causes the job to terminate. This command cannot be issued during initialization. A default command authority of 4 is assigned to this command.

Note: These are basic SOLVE SSI commands only. PPI (see page 32) and SMF (see page 25) support add some additional commands.

SMF Command Descriptions

The following SMF-related commands let you control and inquire about the SMF processing performed by your product. The commands are available only for the copy of the SOLVE SSI that has been started using the SMF=YES parameter, which activates SMF support.

SMF STATUS

Displays current status of SMF processing, status of each SMF exit, and common code component.

SMF DEREGISTER

Deregisters SMF exits, functionally deactivating SMF processing by your product

SMF REGISTER

Registers SMF exits and reactivates SMF processing by your product.

Note: Deregistering and then registering the SMF exits refreshes the exit code.

Packet Analyzer and SOLVE SSI

Note: This section only applies to CA NetMaster NM for TCP/IP and CA NetMaster FTM.

Packet Analyzer intercepts packets inbound and outbound for your stacks, and stores the connection information and general statistics (for example, byte and packet counts) in the SOLVE SSI. It also intercepts SMF records generated by stacks, and File Transfer Protocol (FTP) and Telnet servers to augment the connection information. It depends on having a SOLVE SSI task running.

Packet Analyzer Parameter Descriptions

Before you activate Packet Analyzer, you can specify its parameters in the SSIPARM(SSISYSIN) or SSIPARM(SSIPARMS) (if used) member created during setup.

PACCMINT={ *time_interval* | NO }

Specifies whether you want to monitor changes in the configuration of the stacks and the time interval used to scan for any changes.

time_interval

Specifies the scan interval (in seconds) to use to monitor changes in stack configuration.

Default: 30

Limits: 5 to 600

NO

Specifies that changes in stack configuration are not monitored.

PADSWTPCT={ *dynamic_database_threshold* | NO }

Specifies a percentage threshold to trigger messages to warn users of the dynamic database becoming full. NO disables the warnings.

Default: 80

Limits: 50 to 95 or NO

PAEESTACK={ *stack_job_name* | NONE }

Specifies the job name of the stack that VTAM uses for EE traffic. Setting the value to NONE disables EE packet processing.

Default: None

PAEXMODE={ TASK | ZIIP | BEST }

Specifies whether to move processing performed by the Packet Analyzer from the central processor (CP) to a z/Series Integrated Information Processor (zIIP):

- TASK executes the Packet Analyzer in Task mode on the CP.
- ZIIP executes the Packet Analyzer in SRB mode and makes the workload eligible for dispatching on a zIIP, where available.
- BEST executes the Packet Analyzer in SRB mode and makes the workload eligible for dispatching on a zIIP, where available (that is, as if PAEXMODE=ZIIP is specified). If no zIIP is available, BEST uses Task mode (that is, as if PAEXMODE=TASK is specified). Unlike PAEXMODE=ZIIP, no error occurs if no zIIPs are available.

Default: TASK

PALEVEL={ FULL | NOSTATS | NOPACKETS | NONE }

Specifies the processing level of Packet Analyzer:

- FULL enables all Packet Analyzer functions.
- NOSTATS enables packet processing and tracing, but keeps no statistics.
- NOPACKETS disables packet processing and tracing.
- NONE disables Packet Analyzer (same as PKTANALYZER=NO).

Default: FULL

PAMAXULPORT=*high_port_number*

Specifies the highest local User Datagram Protocol (UDP) port number for which packet statistics are kept. Increasing it can cause the Packet Analyzer database to increase in size.

Default: 1024

Limits: 512 to 65535

PAMITTSIZE=*packet_trace_size*

Limits the number of packets you can specify for the size of the initial packet trace table in a multiple Transmission Control Protocol (TCP) connection trace definition.

Default: 100

Limits: 10 to 999

PAMMTTSIZE=*main_trace_size*

Limits the number of packets you can specify for the size of the main trace table in a trace definition.

Defaults: 20000

Limits: 10 to 99999

PAMFZKPTIME=*ended_trace_retain_time*

Limits the number of minutes you can specify in a trace definition to retain an ended trace.

Default: 1440 (one day)

Limits: 1 to 10080 (seven days)

PAMSSKPTIME=*snapshot_trace_retain_time*

Specifies the number of minutes to retain the snapshot of a running trace.

Default: 1440 (one day)

Limits: 1 to 10080 (seven days)

PAMTCCCRLIM=*number_tcp_traces*

Limits the number of TCP conversation traces (per stack) you can specify for a trace definition. 0 specifies no limit.

Default: 999

Limits: 0 to 9999

PASMFWRITE={ YES | NO }

Specifies whether SMF records can be requested to be written by SMF exits when SMF=YES.

Note: The TCP/IP application name definitions can override this value, subject to the actions of other SMF exits of the same type that are executed later.

PASSWTPCT={ *synchronous_database_threshold* | NO }

Specifies a percentage threshold to trigger messages to warn users of the synchronous database becoming full.

Default: 80

Limits: 50 to 95 or NO

PATSWTPCT={ *trace_database_threshold* | NO }

Specifies a percentage threshold to trigger messages to warn users of the trace database becoming full.

Default: 80

Limits: 50 to 95 or NO

PDCPDSSIZE={ *n* | 1024 }

Specifies the size of the Packet Analyzer decoupler in MB.

Default: 1024

Limits: *n* must be in the range 8 through 1024.

PDCPSGSIZE={ *n* | 256 }

Specifies the size of the Packet Analyzer decoupler segments in KB.

Default: 256

Limits: *n* must be in the range 68 through 1024.

PDYNDBSIZE={ *n* | 64 }

Specifies the size of the Packet Analyzer dynamic database in MB.

Default: 64

Limits: *n* must be in the range 8 through 1024.

PKTANALYZER={ NO | YES }

Specifies whether Packet Analyzer is enabled. You can enable Packet Analyzer in only one SSI per system.

Note: Set this parameter to YES for the following products only:

- CA NetMaster NM for TCP/IP
- CA NetMaster FTM

PSYNDBSIZE={ *n* | 12 }

Specifies the size of the Packet Analyzer synchronous database in MB.

Limits: *n* must be in the range 4 through 256.

Default: 12

PTRCDBSIZE={ *n* | 64 }

Specifies the size of the Packet Analyzer trace database in MB.

Default: 64

Limits: *n* must be in the range 8 through 1024.

STACK REFRESH Command—Refresh Stack Interface Configuration

The STACK REFRESH command refreshes the stack interface configuration for any changes (for example, when an interface is added or removed).

Use PPI with SOLVE SSI

PPI provides a general-purpose facility for programs, written in any language, to exchange data. It also provides a facility for any program to forward a generic alert to NetView or a region. The SOLVE SSI implementation closely follows the IBM implementation. Therefore, programs written to run with the IBM implementation work with the SOLVE SSI implementation with no changes. No special authorization is required to use the PPI and it does not depend on having NetView or a SOLVE SSI running.

Implement PPI

PPI is implemented by SOLVE SSI, allowing it to run regardless of the status of the region. This is important because applications may need to queue data across the PPI even if the region itself is not active.

The region can use PPI regardless of whether it is being provided by SOLVE SSI or NetView. The region need not have a connected SOLVE SSI and the connected SOLVE SSI need not be the PPI owner to use the facilities of PPI.

Note: Only the region connected to the PPI-owning SOLVE SSI can register receiver IDs that start with NETV or NETM.

When the region is initialized, it issues a conditional load for CNMNETV. If CNMNETV is found, then PPI facilities are available.

PPI Parameter Descriptions

Before you activate PPI, you specify its parameters in the SSISYSIN or SSIPARMS (if used) member. The following parameters are recognized:

PPI={ YES | NO }

Indicates whether this SOLVE SSI is to provide PPI services (YES) or not (NO).

Note: Only one SSI can provide the interface.

Default: No

PPIFREELIM={ 25 | *nnnn* }

Specifies the maximum number of pages (each 4 KB) of storage that are retained in the PPI data buffer free storage pool. Buffers longer than approximately 4,060 bytes are not allocated out of this pool.

The buffer-free storage pool is initially empty. Storage is obtained from the system as required, and, as data buffers are received, their storage is returned to the pool. If the number of free pages in the pool then exceeds this limit, the excess pages are freed to the system. The pool reduces overheads by eliminating most GETMAIN/FREEMAIN activity.

Limits: 10 to 1,000 pages

Default: 25 pages (100 KB)

PPIINATO={ *n* | NO | 5 }

Specifies the number of minutes that must elapse before a SHOW PPIUSERS command without the INACT operand hides inactive PPI users with nothing queued to them.

Limits: 0 to 1,440, or NO. Zero or NO indicates no time-out

Default: 5

PPIMAXQB={ 100000 | *nnnnn* }

Specifies the largest value allowed for the PPI receiver queue limit. Larger values are rejected.

Limits: 1000 to 1,000,000. The lowest limit that a PPI receiver can have is 0.

PPIMAXBL={ 65536 | *nnnnn* }

Specifies the largest data buffer size that can be queued to a PPI receiver. This parameter allows the setting of a reasonable limit.

Limits: 1,000 to 1,000,000

PPINETMR={ YES | NO }

PPINETVR={ YES | NO }

Specifies whether only the connected main task can define PPI receiver names starting with NETM or NETV.

Default: YES, meaning that the associated name is restricted.

PPINPREF={ * | *xxxx* }

Specifies the PPI name service prefix value. An asterisk (*) uses the SOLVE SSI SSID; otherwise, this value must be a one- to four-character name consisting of alpha, numeric, or national characters only. We recommend a value the same as the SOLVE SSI-connected domain ID as specified in the NMDID JCL parameter.

PPIRC90T={ YES | NO }

Controls the writing (WTO) of additional debugging messages from PPI whenever a PPI return code 90 is returned on a PPI application program interface (API) call.

PPIREUSE=*n*

Specifies the number of PPIINATO intervals that must elapse before an inactive PPI receiver with nothing queued, that was a PPI-supplied name (function 60), is purged. The name becomes available for reassignment.

Activate PPI

PPI is activated using the JCL parameter PPI=YES | NO, which is specified in the SSISYSIN or SSIPARMS (if used) member. If PPI is set to YES, then SSI attempts to activate PPI when SOLVE SSI initializes.

If PPI is not activated, then check the following:

- Whether another SOLVE SSI owns the PPI. Only one SOLVE SSI can provide PPI services at a time.
- That SOLVE SSI can build the required control block structure to support PPI.

PPI Command Descriptions

The following commands are available or have extended function when PPI is active:

SHOW PPIUSERS [=*name* | =*prefix] [INACT]**

Displays a list, in receiver ID order, of all defined PPI receivers. The display includes statistics on buffer counts and storage. If active, the owning job name and ASID are displayed.

SHOW SSISTATS

If PPI is active, additional statistics on PPI are displayed. This includes the number of receivers and statistics on the PPI buffer storage pool.

Chapter 4: SSI DD SUBSYS Support

This section contains the following topics:

DD SUBSYS

Note: Support for DD SUBSYS is available on MSP and z/OS systems only.

DD SUBSYS is a facility (provided by the system) that allows an authorized subsystem to provide access methods to existing programs. In effect, the subsystem appears to these programs as if it is a set of files, and in this way is able to provide data to, or receive data from, the programs.

For example, assume that you have a third-party program that writes an event log while it is running. The information in this event log could be very useful for system automation. Normally, however, the data written to this log cannot be accessed by another program until the program producing the log is shut down. DD SUBSYS allows you to intercept all data written to the file while logging is taking place. You can then use the WTO facility to send the data to operator terminals. The Advanced Operations Management (AOM) software then forwards the data to any specified NCL process in the region.

SOLVE SSI DD SUBSYS Support

The SOLVE SSI support for DD SUBSYS provides the following facilities:

- The ability to generate WTO output records.
- The ability to send output records to a nominated PPI receiver.
- The ability to act as a PPI receiver and to pass incoming records to a program as an input file.
- The ability to provide a general user function where a user-nominated program can be called to process I/O.

In addition, all the supplied facilities support the following functions:

- The ability to copy the data records to another data definition.
- The ability to call a user-written filter program that can determine whether individual records are to be accepted or rejected.

Enable DD SUBSYS Support in SOLVE SSI

To use the DD SUBSYS support provided by SOLVE SSI, you must enable it.

To enable the use of DD SUBSYS with SOLVE SSI, use the following startup parameter:

```
DDSUBSYS={ NO | YES }
```

The following example enables SSI for DD SUBSYS:

```
//DDSUBSYS EXEC PGM=NMSSI,TIME=1440,
//         PARM=('SSID=xxxx,DDSUBSYS=YES')
//SSILOG   DD SYSOUT=*
//SSIDUMP  DD SYSOUT=*
//SYSUCUMP DD SYSOUT=*
```

The SSID used for this SSI should have been specified in the IEFSSN*xx* or SUBSYS*xx* member in SYS1.PARMLIB during the installation and setup of SOLVE SSI.

You can start an SSI as a started task or as a job.

Considerations When Stopping the SSI

You can stop the SSI by using the following system operator command:

```
F job-name,SSI STOP
```

However, you should not stop or restart the SSI while using it to find DD SUBSYS—JCL errors, because allocation errors can occur.

Also, open errors can occur if you stop the SSI after a job has started but before it opens a DD SUBSYS data set. These errors cause OPEN ABENDs (013-C0) that can crash the user program.

Finally, if you stop the SSI while a DD SUBSYS data set is open, the next I/O request to that data set ABENDs with a U0001 ABEND code. This is preceded by the following message, which is sent as a WTO message to the system console:

```
NS4199 ACCESS ERROR on ddname
```

Restarting the SSI does not prevent ABENDs or errors from occurring in jobs that are active but dormant. Jobs that require the SSI that has been stopped (and restarted) for DD SUBSYS do *not* automatically use the restarted SSI. Subsequent SSI requests from those jobs can fail because the original SSI has stopped. Restart those jobs to use the new SSI.

Most of these restrictions are due to operating system limitations.

Use DD SUBSYS

This section describes how to use DD SUBSYS in general terms. You may also want to refer to the *MVS JCL Reference* manual for further JCL considerations, and the *MVS Programming: Authorized Assembler Services Guide* for details of the Dynamic Allocation interfaces.

Implementation of DD SUBSYS

While implementing the DD SUBSYS facility, you should consider the following:

- An extra operand on the JCL DD statement lets you nominate the name of a subsystem that is to process the I/O requests of this file.

  This is the SUBSYS operand and is used as follows:

  ```
  //ddname DD SUBSYS=(parm1,parm2...parmn)
  ```

  This operand tells the operating system that the nominated subsystem is to process this file.

  Note: If individual parameters of the SUBSYS operand contain special characters, including equals signs (=), then the subparameter must be enclosed in quotes. The quotes are removed before the parameter is passed to the subsystem.

- Equivalent facilities to the DD SUBSYS operand are available when using dynamic allocation. Two text units can be used, as follows:

  ```
  DALSSNM  X'005F'
  ```

  Specifies the subsystem name (corresponding to the first subparameter of the SUBSYS operand in JCL).

  ```
  DALSSPRM  X'0060'
  ```

  Specifies subsystem parameters (corresponding to the additional subparameters of the SUBSYS operand in JCL).

  Note: Since each subparameter passed in the text unit has a length, the previous comments regarding quotes in the JCL parameter do not apply.

- The SSI provides several function codes that must be supported by a subsystem to allow it to use DD SUBSYS.

  These function codes and their purpose are shown in the following table:

Function Code	Purpose
07	Unallocation (UN)—called at job step end or by dynamic deallocation of the file.
16	Open (OP)—called when the file is opened.
17	Close (CL)—called when the file is closed.
38	Converter/interpreter (CI) called when JCL is parsed (not used for dynamic allocation).
39	Allocation group (AG)—called at job step allocation or dynamic allocation time.

- The subsystem must also provide a set of routines that receive control whenever a program that has opened a DD SUBSYS file issues an I/O request to it. These routines get control as a logical subroutine called by the user program and must simulate the processing of normal I/O requests. How these routines provide data to the user program (for input) or data from the user program (for output) is up to you.
- DD SUBSYS interface routines in your operating system translates all DCB-based I/O requests to an ACB/RPL-based interface for the subsystem. This is the normal mode of operation. If the subsystem is coded with appropriate logic, it can handle an application program that opens an ACB to DD SUBSYS. In this case the subsystem could simulate VSAM.

Who Can Use DD SUBSYS Facilities?

Any job in the same operating system as the providing subsystem can use the DD SUBSYS facilities. There is no requirement for the user of DD SUBSYS facilities to be in the same region as the subsystem itself. Of course, a specific implementation of DD SUBSYS may impose restrictions. The SOLVE SSI implementation does *not* impose any restrictions.

An implementation of DD SUBSYS may not support some types of file or I/O. For example, the SOLVE SSI implementation discussed here provides two functions (WTO and PPISEND) that are applicable to output files only. If you attempt to open these files for input the operation fails.

Similarly, some subsystems may not be able to simulate enough VSAM information for a program to open an ACB directly to the subsystem.

To use DD SUBSYS, the providing subsystem must be active; otherwise, you get JCL errors or ALLOCATION and OPEN statements fail.

DD SUBSYS Syntax

All supplied functions of the SOLVE SSI DD SUBSYS facility use the following syntax:

```
//ddname  DD SUBSYS=(ssid,function[,'keyword=value',...])
```

ssid

Specifies the SOLVE SSI SSID.

function

A supported function name. This must be the second subparameter.

keyword=value

Specifies one or more functions or general parameters. Each parameter is a keyword-value pair. Due to JCL requirements, you must use quote marks, because there are special characters to specify, such as the equals sign (=). The parameter names depend on the specific function.

When using dynamic allocation, the subparameters are provided as individual length/value pairs on the DALSSPRM text unit. In this case, there is no need to quote the parameter values.

Supported Function Names

The following function names are supported:

WTO

Invokes the WTO facility that allows output records to be sent to the system console through the WTO macro.

PPISEND

Invokes the PPI SEND function that sends output records through PPI to any PPI receiver.

PPIRECV

Invokes the PPI RECEIVE function that allows records sent to the nominated PPI receiver name to be fed to a program as input.

USER or USERI or USERO

Invokes the USER facilities that allow a user-specified program to be driven to process input or output records.

Common Functions

There are two additional facilities available when using SOLVE SSI DD SUBSYS functions. These facilities enable the copying of data to another file and the filtering of data through a user exit program.

COPY Facility

The COPY facility lets you copy the DD SUBSYS file records (input or output) to another file.

This lets you see and process a data flow while using DD SUBSYS and sending the data to its original destination, for example:

```
//LOG  DD  SUBSYS=(NMSS,WTO,'COPY=LOG2')
//LOG2 DD  SYSOUT=A
```

'COPY=*ddname*'

ddname

Specifies the target data definition name. There are special DCB attribute defaulting rules. You can override these by using a DCB parameter on the target COPY ddname.

The rules are as follows:

- If the DCB has no RECFM specified, then the attributes are forced to RECFM=U, LRECL=0, and BLKSIZE=6233; however:
  - If RECFM is U and BLKSIZE is specified, then the specified BLKSIZE is used.
  - If RECFM is U and no BLKSIZE is specified, then BLKSIZE=6233 is set.
- If LRECL=0, and:
  - If the RECFM is FB, then LRECL=132 is set, or LRECL=133 if the RECFM has A or M. If B (Blocked), LRECL*40 is set as the BLKSIZE, otherwise the BLKSIZE is set to the LRECL.
  - If the RECFM is VB, then LRECL=136 is set, or LRECL=137 if the RECFM has A or M. If B (Blocked), 6233 is set as the BLKSIZE, else the BLKSIZE is set to the LRECL+4.
- If BLKSIZE=0, and:
  - If the RECFM is U, BLKSIZE=6233 is set.
  - If the RECFM is F, BLKSIZE=LRECL is set.
  - If the RECFM is FB, a BLKSIZE that is the largest integral multiple of the LRECL less than 6233 is set. Otherwise, the value of the LRECL is used, if it is greater than 6233.
  - If the RECFM is V, BLKSIZE=LRECL+4 is set.
  - If the RECFM is VB, BLKSIZE=6233 is set (if LRECL is less than 3110), otherwise BLKSIZE=LRECL+4 is set.

You can cascade DD SUBSYS statements; the DD statement for the *ddname* specified for one COPY operand can itself be a DD SUBSYS definition. This can allow several facilities to see output from a program.

Note: The order in which the FILTER and COPY statements are used on a single DD SUBSYS statement is important. If the FILTER operand precedes the COPY operand, then the COPY facility copies only records that were not rejected by the filter exit. If the FILTER operand follows the COPY operand, all records are copied, regardless of the result of the filtering process.

If the COPY file has an I/O error or an ABEND, the application program abends.

FILTER Facility

The FILTER facility calls a user exit program to determine whether each individual record is to be passed on for processing.

If processing an output file, for example, for PPISEND, you can use the FILTER facility to prevent a record from being sent.

If processing an input file, you can use the FILTER facility to prevent a record from being passed to the user program.

Additionally, if the COPY facility is also being used on the same DD statement, filtering may affect the copy facility.

The syntax of the FILTER operand on DD SUBSYS is:

```
'FILTER=pgmname'
'FILTER=(pgmname)'
'FILTER=(pgmname,)'
'FILTER=(pgmname,parm)'
```

Note: Due to JCL requirements, you *must* use quote marks when specifying the FILTER operand, because it contains special characters. (Quoting is not necessary, however, if you are using dynamic allocation.)

FILTER

Can intermingle with other DD SUBSYS subparameters, although there is an interaction between it and the COPY parameter.

pgmname

Specifies the name of a user exit program. This program is loaded during OPEN processing for the file. It is then called to open the file, once for each GET or PUT of a logical record, and to close the file.

parm

(Optional) If specified, it must be a one-character to eight-character value, in PDSNAME format. It is passed to the user filter exit. The exit can use it as, for example, a data definition name (*ddname*) of a control file.

Note: If using both the FILTER and COPY facilities on a single DD SUBSYS statement, the order is important. If the FILTER operand precedes the COPY operand, the COPY facility copies only records that were not rejected by the filter exit. If the FILTER operand follows the COPY operand, all records are copied regardless of the result of the filtering process.

If the FILTER program abends while processing a FILTER request, the application program also abends.

Carriage Control

When an output data set has carriage control specified (that is, the RECFM has A or M in it), the following special considerations apply:

- The ACB/RPL interface does not normally provide the control character as part of the record. Rather, the control character is pointed to separately. Because of this, SOLVE SSI DD SUBSYS support normalizes the record.
- When using facilities such as PPISEND or WTO, you probably do not want to send the control character. These facilities default to not sending it, but this can be overridden.
- The COPY facility always copies the entire record, including the control character.
- The FILTER facility user exit is passed the entire record, including the control character, and is notified of its existence so that it can take appropriate processing steps.

WTO Facility

The WTO facility provided by SOLVE SSI lets you send records that are written to a file as WTO messages. This allows the messages to be viewed by an operator, and picked up by an automation product.

Note: Use the WTO facility sparingly; otherwise, you can fill up console buffers and crash the system.

A logical record can have an optional prefix added to the front, then the total record is truncated to 126 characters before being sent as a WTO message.

WTO DD SUBSYS Operands

The following are the DD SUBSYS operands for the WTO function:

Note: Other than for the WTO name itself, the order of the operands is not important.

WTO

Specifies the WTO function and must be the first subparameter after the subsystem name.

'ROUTCDE=(list)'

Specifies an optional list of routing codes to apply to the WTO messages. If this operand is omitted, ROUTCDE=11 is assumed. Quote marks are required.

Limits: The list of routing codes can be a single number (in which case, no parentheses are necessary) or a list of numbers from 1 through 16 (MSP) or 1 through 128 (z/OS).

Note: If using routing codes greater than 20, authorization is required. Unauthorized programs cannot issue WTO messages using these routing codes and, if they do, an ABEND D23 results.

'DESC=(list)'

Specifies an optional list of descriptor codes to apply to the WTO messages. If this operand is omitted, DESC=7 is assumed. Syntax is as for ROUTCDE, values range from 1 through 16.

Note: It is not advisable to use descriptor codes 1, 2, or 11, as these messages will stay on the consoles.

'PREFIX=*value*'

Allows for the specification of an optional WTO prefix. If this operand is omitted, the SSID is used (but you can force the prefix to be dropped by coding PREFIX=NO). Otherwise, you can specify a 1-character to 12-character prefix value. There must be a single blank between the prefix and the text of the message.

'BASE=*n*'

Controls the starting column number for the data that is sent as a WTO message. The value of n must be a number from 1 through 32760. It is a logical column number. The default value is 1 if the file does not have carriage control; otherwise, it defaults to 2. This means that a print file will not normally have carriage control characters in the WTO text. BASE can also be used to skip a record prefix, and so on.

'COPY=*ddname*'

'FILTER=*parms*'

Allow for copying and filtering of data.

Notes:

You need to consider the following:

- A null record after BASE considerations is ignored.
- Routing codes above 16 are not supported unless the operating system is z/OS.
- The FILTER facility is especially useful here to limit the amount of data sent to the consoles.

PPISEND Facility

The PPISEND facility provided by SOLVE SSI lets you send logical records written to a file to any PPI receiver. The receiver can be, for example, an NCL process in a region. It can also be any user program that uses the PPI API.

PPISEND DD SUBSYS Operands

The following are the DD SUBSYS operands for the PPISEND function:

Note: Other than for the PPISEND name itself, the order of the operands is not important:

PPISEND

Specifies the PPISEND function and must be the first subparameter after the subsystem name.

'TARGET=*targetid*'

Nominates the identifier of the PPI receiver. *targetid* must be a valid PPI receiver name. The name must be defined to PPI at the time the data set is opened (although the PPI need not be active). This operand is required.

'SOURCE=*sourceid*'

Provides an optional PPI sender ID. This must be a valid PPI name, but is not registered to PPI. As well as a valid name, the following special names are supported:

***JOB**

Use the job name.

***STEP**

Use the job step name. If the job step name does not exist, use the job name.

***PSTEP**

Use the procedure step name. If the procedure step name does not exist, use the step name. If the step name does not exist, use the job name.

Default: *JOB

'QFULL=*option*'

Controls the action to take if the PPISEND receives a PPI queue full condition (PPI return code 35). The following actions can be specified:

ERROR

Specifies that a Dataset Full condition be reflected to the application program with an RPL error of 28 (X'1C'). If the application is using a DCB (as is normally the case), then this is reflected as an I/O error (not an X37 ABEND).

IGNORE

Specifies that the queue full return code be ignored. This means that the record is not sent; however, processing continues. Note that, if you are using the COPY facility, the COPY is still performed.

Default: ERROR

'BASE=*n*'

Controls the starting column number for the data that is sent to PPI. n must be a number from 1 through 32760. It is a logical column number.

Default: 1, if the file does not have carriage control; otherwise, it defaults to 2. This means that a print file will not normally have carriage control characters sent across PPI. BASE can also be used to skip a record prefix, and so on.

'COPY=*ddname*'

'FILTER=*parms*'

Allow for copying and filtering of data.

Notes:

You need to consider the following:

- The *targetid* and *sourceid* values are validated at JCL syntax check time. If they are invalid then a JCL error occurs. This syntax check is repeated at step allocation time.
- When the data set is opened, PPI is called to check that the receiver exists. If PPI is inactive, or if the receiver is not defined, then the open fails. This leads to an 013-C0 OPEN ABEND occurring for a DCB. For an ACB, an ACB open error code is set. Messages are written informing of the problem. Any other unexpected PPI return codes also cause an open error.
- The data set must be open for output, otherwise the open fails.
- When records are written to the data set, a PPI SEND is issued for each logical record. Any return code other than 0 (All OK), or 4 (Inactive But Queued) results in an I/O error. The return code of 35 (Receiver Queue Full) is translated into an RPL error code of 28 (X'1C')—data set Full, for ACB/RPL-based callers (if QFULL=ERROR is in effect; otherwise, it is ignored).
- The PPI receiver receives the logical records (with no prefix, even if the user program is writing V or VB), exactly as sent, or as adjusted by the BASE value. The receiver can use the sender ID to determine where the records came from.
- No null record (not even one that is logically null after considering the BASE value) is sent.

PPIRECV Facility

The PPIRECV facility provided by SOLVE SSI lets you receive data from other PPI senders and to provide the received data as an input file. The senders can be any other PPI users, including NCL processes in a region. They can also be any program that uses the PPI API.

PPIRECV DD SUBSYS Operands

The following DD SUBSYS operands are required or optional with the PPIRECV function:

Note: Other than for the PPIRECV name itself, the order of the operands is not important:

PPIRECV

Specifies the PPIRECV function and must be the first subparameter after the subsystem name.

'ID=*name*'

Nominates the PPI receiver name. name must be a valid PPI receiver name. The name must not be defined to PPI at the time the data set is opened or, if defined, must not be active. This operand is required.

'APF=*option*'

Allows you to specify whether or not PPI senders must be APF authorized to send to this name.

NO

Specifies that senders do not need APF authorization.

YES

Specifies that senders must be APF authorized.

Default: NO

'IQEMPTY=*option*'

'QEMPTY=*option*'

Specifies the action to take when a PPI Queue Empty return code (30) is returned. The two operands allow a different action to be taken for the initial read request (IQEMPTY), and for all subsequent read requests (QEMPTY).

The following actions can be specified:

EOF

Specifies that an end-of-file is returned to the application. This is the default for QEMPTY.

WAIT (WAIT)

Specifies that an indefinite wait is performed until data is queued.

(WAIT,*n*)

Specifies that a wait for an interval of n seconds is performed. *n* is in the range 1 through 86400. If no data arrives in this interval, an end-of-file is returned.

Default: WAIT

'MAXQUEUE=*n*'

Specifies the PPI receiver buffer queue limit. The value of *n* must be in the range 1 through 9999.

Default: 10.

'COPY=*ddname*'

'FILTER=*parms*'

Allow for copying and filtering of data.

Note: Because PPIRECV is an input facility, the filtering is performed on records received from PPI, before being returned to the application as input records.

Notes:

You need to consider the following:

- If the program opens the file as RECFM=F or FB, short input records are padded with blanks to the LRECL.
- Records that are too long cause an I/O error.
- Use of IQEMPTY and QEMPTY can allow you to set up a job to wait indefinitely for initial input, then process input until no more arrives in a certain time, then return EOF and terminate.
- If you are filtering, you may not get a return on input for several wait intervals. This is because, if the filter program rejects the input record, the wait is re-executed.

USER, USERI, and USERO Facilities

The USER*x* facilities let you supply your own DD SUBSYS I/O routines. This is an open-ended facility. If one of the supplied functions does not allow you to do what you want to do, then you can use the USER*x* facilities to implement it.

Full details on the API for the user program are provided as well as a sample.

USERx DD SUBSYS Operands

The following DD SUBSYS operands are used with the USER*x* function:

Note: Other than for the USER*x* name itself, the order of the operands is not significant.

USER

USERI

USERO

Specifies the user function to be performed and must be the first subparameter after the subsystem name.

USER

Enables the user program to handle most data set options. The only option that is blocked by the SOLVE SSI front-end is the use of ASY mode I/O.

USERI

Enables the user program to handle sequential input only. This is the most common type of input processing and removes the need for the user program to validate ACB and RPL options.

USERO

Enables the user program to handle sequential output only. This is the most common type of output processing and removes the need for the user program to validate ACB and RPL options.

'PGM=*pgmname*'

Names the user program. *pgmname* must be a valid program name. The program must be available for loading at OPEN time (for example, in the STEPLIB or link list).

The user program need not be APF-authorized, unless the application opening the file is APF-authorized, in which case the user program must come from an APF-authorized library. The PGM operand is required.

'PARM=*parm*'

Provides an optional one-character to eight-character parameter to be passed to the user program. If specified, the value must be in a valid partitioned data set (PDS) name format. If omitted, an 8-character value of all blanks is used.

'COPY=*ddname*'

'FILTER=*parms*'

Enables the copying and filtering of data.

If the file is being written to, filtering and copying occur before the user program is called. Records rejected by filtering are not passed to the USER program.

If the file is being read, filtering and copying are performed on the records provided by the USER program. If the filter program rejects a record, the user program is called to provide another one.

Note: A sample user program is provided in source form (UTIL0038 (see page 69)).

FILTER Exit API

This section describes the supplied API that lets you write a filtering program.

The filter program can be used to select which records that are written to or read from a DD SUBSYS facility are actually processed (written records), or returned to the requester (records read).

The filter program can be written in any language that supports standard linkage conventions. However, for performance reasons, it is probably best written in assembler.

The sample filter program that is supplied contains extensive comments.

APF Authorization

The use of DD SUBSYS could provide an opportunity to circumvent system security. This is because the exit programs run during OPEN and CLOSE processing, and normally the system is in supervisor state or has a protection key set (to less than 8) at these times.

Consequently, the following rules are implemented:

- If the application program (that is, the job step program) is not APF authorized, then the filter exit need not be APF authorized, and need not reside in an APF-authorized library. Although it can be APF authorized, and can reside in an APF-authorized library, the APF authorization is ignored.
- If the application program is APF authorized, then the filter exit must come from an APF-authorized library, although the exit need not be APF authorized itself. Since APF libraries are normally security protected, this prevents a potentially dangerous filter program from being loaded and executed in an environment where APF authorization exists.

  Note: Specifying any unauthorized step libraries results in all step libraries being considered not authorized for that step.
- The OPEN and CLOSE calls to the filter exit are made in the job step TCB key and state.
- The FILTER calls to the filter exit are made directly from the application, thus the system state and key at this time is dependent on the application.

The above rules mean that a simple application can use a filter program without any need to access APF libraries, and so on. Only when using authorized applications is there any need to place the filter program in an APF library.

Abnormal Termination (ABEND)

If the filter program abends during an OPEN or CLOSE call, the ABEND is trapped by the DD SUBSYS code and is reflected as an OPEN error (during an OPEN), or is ignored (during a CLOSE).

If the filter program abends during a FILTER call, the ABEND is reflected by the application (as the filter program is merely being called as a subroutine of the application program). For this reason, you should ensure that the filter program is well tested before placing it in a production environment.

Dynamic Allocation

While the user program can open its own files at any time (including during an OPEN and a CLOSE call), it cannot make Dynamic Allocation requests during OPEN or CLOSE calls. This is because a system ENQ is held, and a call to Dynamic Allocation results in an error.

Thus, the user program should have any required files pre-allocated in the JCL before an OPEN is performed on the DD SUBSYS file.

Calling Details

The filter program is called with registers set up as follows:

Register	Content
R0	Indeterminate.
R1	Points to a parameter list, as described below.
R2...R12	Indeterminate.
R13	Points to a standard 18-word save area.
R14	Contains the return address and AMODE.
R15	Contains the entry point and filter exit AMODE.

In a 31-bit environment, the filter exit is called in the AMODE, as established by the linkage editor. It need not return in the AMODE specified in register 14 on entry (that is, it can use a BR R14 instruction to return), because the return address is guaranteed to be below the line. The caller of the filter exit restores its own AMODE.

For the OPEN and CLOSE calls, the PSW key and state is as for the job step program (normally key 8, problem state).

For the FILTER calls, the PSW key and state is as for the application program at the time it issued the I/O request. This is normally key 8, problem state.

On completion, the filter program must restore registers 2 through 12 to the values they had when the filter program started processing.

Return Codes

The filter exit sets a return code in register 15 to reflect the results of its processing.

Setting a non-zero value for the OPEN call means that the attempted opening of the data set failed. An optional error message can be supplied in this case.

The return code for the CLOSE call is ignored.

A non-zero return code for the FILTER call means that the current record is to be rejected.

Reentrancy

The filter exit need not be written to be reentrant. However, if the same filter exit is to be used for more than one DD SUBSYS filter in a single job step, it is a good idea to make it reentrant. This is because each OPEN will cause a separate non-reentrant copy to be loaded. However, at CLOSE time, there is no way to tell the system which copy is to be deleted. This means that an active copy of the exit could be deleted instead of the one that has been closed.

Parameter List

The parameter list for the filter exit is provided by the Assembler macro $NMDDSFP. The following are described:

- The individual fields in the parameter list, including the name of the pointer in the parameter list
- The target field

Note: The parameter list is pointed to by register 1 on entry to the filter exit. Many of the parameters are in storage and cannot be altered by the filter exit.

FPLS@FC

A pointer to a binary full word that contains a function code. The function code determines what processing is required. Valid function code values are:

0-OPEN CALL

4-CLOSE CALL

8-FILTER CALL

FPLS@DDN

A pointer to the eight-character, blank padded *ddname* for the file being filtered. This *ddname* is in protected storage and cannot be altered.

FPLS@PRM

A pointer to the eight-character, blank padded parameter value specified as the filter (for example: 'FILTER=(MYPGM,MYPARM)' would result in the parameter being CL8'MYPARM'). If no value is specified, then the field is blank.

This parameter value is in protected storage and cannot be altered.

Note: If a value is specified, it is edited to be a valid *ddname* (or PDS member name). This means that no editing is required if you want to use it as a *ddname* or member name (although obviously you need to test for the existence of the *ddname* or member name).

FPLS@JFC

A pointer to the Job File Control Block (JFCB) for the file. This control block contains various useful fields, such as the DSNAME (assuming one was specified along with the SUBSYS parameter; otherwise a system-generated name is provided).

The IEFJFCBN mapping macro maps this control block. The JFCB is in protected storage.

FPLS@UWD

A pointer to a four-byte, aligned area, initialized to binary zeros prior to the OPEN call. You can update this value. The updated value is then passed to subsequent FILTER calls and to the CLOSE call. If the value is further updated, then the new value is passed on subsequent calls.

An excellent use for this field is to anchor a work/save area that you obtain on the OPEN call. This makes it very easy to make the filter program fully reentrant.

Note: If a non-zero return code results from the OPEN call (that is, the OPEN fails), then the program is not called again. In this case, any work areas obtained should also be freed, as the CLOSE call is unable to free them later.

FPLS@FLG

A pointer to a one-byte flag, in protected storage. This flag contains several useful equated bits (the equates are in $NMDDSFP):

FFLG1OIN X'80'

The file is open for input.

FFLG1OOT X'40'

The file is open for output (FFLG1OIN can also be set).

FFLG1OUP X'20'

The file is open for update (both FFLG1OIN and FFLG1OOT are set).

FFLG1CCH X'01'

The file is open for output and there are control characters present (that is, the original data set had A or M in the RECFM).

FPLS@LEN

A pointer to a length field.

- For the OPEN call, it points to a full word in protected storage that has the value F'120'-this is the length of the supplied error message return area.
- For the CLOSE call, it points to a full word in protected storage that has the value F'0'.
- For the FILTER call, it points to a full word that contains the length of the record that is to be filtered. Note that this value is not in protected storage, but that altering it will not have any effect on other processing.

FPLS@REC

A pointer to the record area.

- For the OPEN call, it points to a 120-byte, blanked area. This area can be used to return an error message when you return a non-zero return code to fail the open. In this case, if the area is not blank, the message is sent through WTO to inform of the open failure reason.
- For the CLOSE call, it points to a full word F'0' in protected storage.
- For the FILTER call, it points to the record to be filtered. This record must not be altered. There is no guarantee that you will be able to access storage past the length pointed to by FPLS@LEN.

The record starts with the control character, if one is present.

FPLS@ACB

A pointer to the ACB for the file. On OPEN, this ACB is a copy in protected storage. On CLOSE and FILTER, this is the real ACB. It must not be altered.

FPLS@RPL

A pointer to the RPL for the I/O request. For OPEN and CLOSE, it points to a dummy full word 0 in protected storage. For I/O, it points to the real RPL. Because the record address, length, handle control character prefixing, and LOCATE mode are provided, there should be little need to actually refer to the RPL.

The parameter list provides all the information that is needed to perform record filtering.

Sample Filter Exit

A sample filter exit, UTIL0037 (see page 65), is supplied in source form. It illustrates the use of the filter parameters to provide a general-purpose filter facility.

USERx Facility API

This section describes the API that lets you write your own DD SUBSYS I/O handler.

The user program can be used to replace any DD SUBSYS function with your own code. Some suggestions are:

- Reformatting files for programs that cannot be rewritten
- Encryption and decryption of data without intermediate files
- Real-time monitoring of messages

The user program can be written in any language that supports standard linkage conventions. However, for performance reasons, it is best written in assembler.

The user program includes extensive comments.

APF Authorization

The use of DD SUBSYS could provide an opportunity to circumvent system security. This is because the user programs run during OPEN and CLOSE processing, and normally the system is in supervisor state or has a protection key set (to less than 8) at these times.

To avoid security exposure, the following rules are implemented:

- If the application program (that is, the job step program) is not APF authorized, then the user program need not be APF authorized, and need not reside in an APF-authorized library. Although it can be APF authorized, and can reside in an APF-authorized library, the APF authorization is ignored.
- If the application program is APF authorized, then the user program must come from an APF authorized library, although the program need not be APF-authorized itself. Since APF libraries are normally security protected, this prevents a dangerous user program from being loaded and executed in an environment where APF authorization exists.

  Note: Specifying any unauthorized step libraries results in all step libraries being considered not authorized for that step.
- The OPEN and CLOSE calls to the user program are made in the job step TCB key and state.
- The I/O calls to the user program are made directly from the application, thus the system state and key at the time is dependent on the application.

The above rules mean that a simple application can use a user program without any need to access APF libraries, and so on. Only when using authorized applications is there any need to place the filter program in an APF library.

Abnormal Termination (ABEND)

If the user program abends during an OPEN or CLOSE call, the ABEND is trapped by the DD SUBSYS code and reflected as an OPEN error (during an OPEN call), or is ignored (during a CLOSE call).

If the user program abends during an I/O call, the ABEND is reflected by the application (as the user program is merely being called as a subroutine of the application program). For this reason, ensure that the user program is well tested before placing it in a production environment.

Supported I/O Requests

The USER*x* facilities allow most I/O requests to be processed. The restrictions are as follows:

- For the USER option, any Open mode is allowed. The only I/O option that is not allowed is ASY (asynchronous).
- For the USERI option, Open For Input Only is allowed. The only I/O request that is permitted is for synchronous input.
- For the USERO option, Open For Output is required. The only I/O request that is permitted is for synchronous output.
- The DD SUBSYS code always checks and disallows illogical conditions such as PUTLOCATE (not allowed by VSAM).
- The API hides most details of the ACB/RPL interface. Only the USER option needs to examine the ACB and RPL. USERI and USERO are protected from most things, including the use of GETLOCATE.

Dynamic Allocation

While the user program can open its own files, and so on, at any time (including during the OPEN and CLOSE calls), it cannot make dynamic allocation requests during OPEN or CLOSE calls. This is because a system ENQ is held and calling dynamic allocation results in an error.

Thus, the user program should have any required files pre-allocated in the JCL before an OPEN is performed on the DD SUBSYS file.

Calling Details

The user program is called with registers set up as follows:

Register	Content
R0	Indeterminate.
R1	Points to a parameter list, as described below.
R2...R12	Indeterminate.
R13	Points to a standard 18-word save area.
R14	Contains the return address and AMODE.
R15	Contains the entry point and filter exit AMODE.

In a 31-bit environment, the user program is called in the AMODE, as established by the linkage editor. It need not return in the AMODE specified in R14 on entry (that is, it can just use a BR R14 instruction to return), because the return address is guaranteed to be below the line. The caller of the user program restores its own AMODE.

- For the OPEN and CLOSE calls, the PSW key and state is as for the job step program (normally key 8, problem state).
- For the I/O calls, the PSW key and state is as for the application program at the time it issued the I/O request. This is normally key 8, problem state.

On completion, the user program must restore registers 2 through 12 to the values they had when the user program started processing.

Return Codes

The user program reflects its processing by setting a return code in register 15.

Setting a non-zero value for the OPEN call means that the OPEN of the data set failed. An optional error message can be supplied in this case.

The return code is ignored for the CLOSE call.

A non-zero return code for the I/O call causes an RPL logical error code to be set, unless the user program has already set a return error code. This means that you should normally return R15 = 0.

Reentrancy

The user program need not be written to be reentrant. However, if the same user program is to be used for more than one DD SUBSYS file in a single job step, it is a very good idea to make it reentrant. This is because each OPEN causes a separate, non-reentrant copy to be loaded. However, at CLOSE time, there is no way to tell the system which copy is to be deleted. This means that a still active copy of the program could be deleted, instead of the one that has been closed.

Parameter List

The Assembler macro $NMDDSFP provides the parameter list for the filter exit. The following are described:

- The individual fields in the parameter list, including the name of the pointer in the parameter list
- The target field

Note: The parameter list is pointed to by register 1 on entry to the filter exit. Many of the parameters are in storage and cannot be altered by the filter exit.

UPLS@FC

A pointer to a binary fullword that contains a function code. The function code determines what processing is required. The function code values are:

0-OPEN call

4-CLOSE call

8-I/O call (see UPLS@IOF)

UPLS@DDN

A pointer to the eight-character, blank-padded *ddname* for the file being processed. This *ddname* is in protected storage and cannot be altered.

UPLS@PRM

A pointer to the eight-character, blank-padded parameter value supplied as the value for the PARM operand (for example, 'PGM=MYPGM', 'PARM=MYPARM', would result in the parameter being CL8'MYPARM '). If no value is supplied, then the field is blank.

Note: If a value is supplied, it is edited to be a valid *ddname* (or PDS member name). This means that no editing is required if you want to use it as a *ddname* or member name (although obviously you need to test for *ddname* or member name existence).

This parameter is in protected storage and cannot be altered.

UPLS@NME

A pointer to the eight-character, blank-padded USERx name. The value is USER, USERI, or USERO. By checking this value on OPEN, you can ensure that the I/O calls are made only for expected values (for example, if the value is USERI, we guarantee that only synchronous GET requests are allowed).

If you allow the value USER, you need to be prepared to handle any I/O request type (particularly if the application opens an ACB as if it were a VSAM file).

UPLS@UWD

A pointer to a four-byte, aligned area, initialized to binary zeros before the OPEN call. You can update this value. The updated value is then supplied to subsequent I/O calls and to the CLOSE call. If the value is further updated, then the new value is supplied on subsequent calls.

An excellent use for this field is to anchor a work/save area that you obtain on the OPEN call. This use makes it easy to make the USER program fully reentrant.

Note: If a nonzero return code results from the OPEN call (that is, the OPEN fails), then the program is not called again. In this case, any work areas obtained should also be freed, as the CLOSE call is unable to free them later.

UPLS@FLG

A pointer to a 1-byte flag that is in protected storage. This flag contains several useful equated bits (the equates are in $NMDDSUP):

UFLG1OIN X'80'

The file is open for INPUT.

UFLG1OOT X'40'

The file is open for OUTPUT (UFLG1OIN can also be set).

UFLG1OUP X'20'

The file is open for UPDATE (both UFLG1OIN and UFLG1OOT are set).

UFLG1FIX x'02'

The data set records are fixed length.

UFLG1CCH X'01'

The file is open for output and there are control characters present (that is, the original data set had A or M in the RECFM).

UPLS@RLN

A pointer to a binary halfword (two bytes long) that contains the maximum record length. This is the record length as defined on OPEN and is data only. When UFLG1FIX is on, this value is useful for determining the correct record length to return for GET requests to prevent I/O errors.

This field is in protected storage and cannot be altered.

UPLS@IOF

A pointer to the I/O function code for I/O calls only (points to a dummy fullword 0 for OPEN/CLOSE). The values of this function code are re-equated in the IFGRPL Assembler macro for the RPLREQ field. The field is a binary fullword.

To avoid the need for USERI and USERO programs to refer to the IFGRPL macro, the GET and PUT equates are defined in the $NMDDSUP macro:

```
RPLGET     X'00000000'    (UPLSFGET)

RPLPUT     X'00000001'    (UPLSFPUT)
```

UPLS@IOA

A pointer to an I/O area.

- For OPEN, points to a 120-byte blank-padded error message area. You can set an error message here for output if you fail the OPEN call.
- For CLOSE, points to a fullword 0 in protected storage (cannot be written to).
- For I/O (GET/PUT requests only for the USER program), points to an I/O area. This area might or might not be the actual user I/O area, but this is not significant. A work I/O area is supplied in the case of GET LOCATE.
- For other I/O requests (other than GET/PUT), it points to a fullword 0 in protected storage.

UPLS@IOL

A pointer to the length field for I/O.

- For OPEN, points to a fullword with the value 120 (the length of the error message area) in protected storage.
- For CLOSE, points to a fullword 0 in protected storage.
- For PUT I/O requests, points to a fullword that contains the length of the record being output.
- For GET I/O requests, points to a fullword containing the length of the I/O area. You must update this fullword to be the actual length of the record that you are providing.

  Note: You do not need to make any special considerations for GET LOCATE, as a dummy record area is supplied in this case.
- For other I/O requests (USER only), points to a fullword 0 in protected storage.

UPLS@ERF

A pointer to a 2-byte error return field for I/O procedures.

- For OPEN and CLOSE, points to a fullword 0 in protected storage.
- For I/O, points to a 2-byte field that is initialized to binary zeros. This field is where you can return RPL error codes as required. If no errors are to be returned, then these fields can be ignored.

The first byte must be set to X'08' for a logical error, or X'0C' for a physical error.

The second byte must be set to the reason code. For example, if you are returning a logical error (X'08'), you could set X'04' for EOF (input) or X'1C' for data set full (output).

If you set a nonzero value in the first byte other than 08 or 0C, or set a nonzero value in the second byte and leave byte 1 zero, then the interface routines force an error code of X'08' with reason code X'DA'.

Note: If you want the RPL error fields to be set directly, you leave these fields set to zero.

UPLS@DSN

A pointer to the 44-character data set name (dsname).

This name is the dsname specified in the DD SUBSYS statement or a system-generated name if you did not specify a dsname. This field is in protected storage and cannot be altered.

UPLS@ACB

A pointer to the ACB for the file.

- For OPEN, this ACB is a copy in protected storage.
- For CLOSE and I/O, this is the real ACB. The value must not be altered.

Note: If the application program opened a DCB, this ACB is the dummy one built by the SAMSII routines.

If you want to map the ACB, the appropriate Assembler macro is IFGACB.

UPLS@DEB

A pointer to the DEB for the file.

This is a dummy DEB as built for DD SUBSYS files. The value is in protected storage and cannot be altered. The IEZDEB Assembler macro maps this control block.

UPLS@JFC

A pointer to the Job File Control Block (JFCB) for the file. This control block contains various useful fields, such as DSNAME (assuming that one was specified with the SUBSYS parameter; otherwise, a system-generated name is supplied).

The IEFJFCBN Assembler mapping macro maps this control block. The JFCB is in protected storage.

UPLS@SOB

For OPEN and CLOSE, points to the SSOB for the OPEN (SSOBFUNC=16) and CLOSE (SSOBFUNC=17) calls.

The SSOB can be used to locate other SSI control blocks. These control blocks are all in protected storage. The IEFJSSOB DA Assembler macro can be used to obtain the SSOB and OPEN/CLOSE extension maps.

For I/O, the pointer is a zero (that is, it points to nothing).

UPLS@RPL

A pointer to the RPL for the I/O request.

- For OPEN and CLOSE, points to a dummy fullword 0 in protected storage.
- For I/O, points to the real RPL. Because the record address and length are provided, and control character prefixing and LOCATE mode (and so on) are handled automatically, there is little requirement to refer to the RPL.

USER programs can refer to the RPL for exotic option flags, KEY pointers, and so on.

The IFGRPL Assembler macro maps this control block.

Note: The parameter list provides all the information that is required to perform your own I/O processing.

Sample USER Program Exit

A sample filter exit, UTIL0038 (see page 69), is supplied in source form. It illustrates the use of the USER facility to provide a simple encryption/decryption facility.

Sample FILTER Exit: UTIL0037

To illustrate the FILTER facility, a sample filter exit program, UTIL0037, is supplied (in source form as well as in compiled form).

This section explains how UTIL0037 is used by filter records.

UTIL0037 Processing

UTIL0037 filters records by referring to a control table, which is built at OPEN time from a sequential file of control statements.

This file of control statements is named (ddname) by the PARM option of the FILTER operand.

Example

```
//OUTFILE  DD  SUBSYS=(NMSS,WTO,
//             'FILTER=(UTIL0037,FILTCTL)')
//*
//FILTCTL  DD  DSN=MY.FILTER.CONTROL,DISP=SHR
```

Each record to be filtered is actioned against the filter table and accepted or rejected based on strings in the record.

If there are syntax errors in the control file, the attempted OPEN fails and an error message is generated that includes the line in error.

Control File Format

The control file for UTIL0037 is in the following format:

- The control file must be F or FB, LRECL=80. It can be a sequential file or a PDS member (specify the member name on the DD statement for the control file).
- Only columns 1 through 72 of the input record are examined.
- Blank lines are ignored.
- Lines with an asterisk (*) as the first non-blank character are ignored and thus can be used as comments.
- All other lines must contain valid control statements.

Filter Processing

UTIL0037 processes a record to be filtered as follows:

- The record is processed against each statement in the control file in turn.
- Some statements can cause the record to be immediately accepted or rejected. In this case, a return to DD SUBSYS is made with the appropriate return code (0 for ACCEPT, 4 for REJECT).
- If the record reaches the bottom of the control file, then it is implicitly accepted.

Control Statements

In the following descriptions, *string* is a character string containing any characters (including unprintable ones), delimited by a pair of one of the following characters: /, \, ¢, and |.

Whatever opening delimiter is used, it cannot appear in the string and must be the closing delimiter. The closing delimiter must be followed by a blank.

The following control statements are recognized:

BASE *n*

Provides a way to set a logical column number.

By default column 1, as used by the other statements, corresponds to the first byte of a record, unless the file is open for output and has control characters. In this case, column 2 is used as logical column 1.

By specifying BASE *n*, you nominate which column (*n*) of the record is to be treated as column 1.

Note: Because the other statements do not permit a column number less than 1, you cannot back up to columns before the logical column set by the BASE statement.

The BASE statement affects any following statements, and you can have several BASE statements in the control file.

There is a default BASE 1 or BASE 2 statement assumed at the start of the control file (the position depends on the presence of a control character).

REJECT *string* [*scol* [*ecol*]]

ACCEPT *string* [*scol* [*ecol*]]

SELECT *string* [*scol* [*ecol*]]

Describes a set of records that are to be rejected, accepted, or selected by filtering.

For REJECT, if the match criteria is satisfied, then the current record is immediately rejected by filtering.

For ACCEPT, if the match criteria is satisfied, then the current record is immediately accepted by filtering.

For SELECT, if the match criteria is not satisfied, then the current record is immediately selected by filtering.

If neither *scol* or *ecol* are provided, then the string must match in the current logical column 1 (see the description for the control statement BASE n in this table).

If just *scol* is provided, then it is either the starting logical column for the string to be in (a number from 1 to 32760), or it can be an asterisk, meaning anywhere in the input record (from logical column 1 onward).

If *ecol* is provided, it sets a logical column range for the string to occur in. *ecol* cannot be specified if *scol* is an asterisk. *ecol* can be either a number (greater than or equal to *scol* plus the length of the string minus 1) or an asterisk, meaning anywhere from *scol* to the end of the record. Note that *ecol* sets the ending column position for the end of the string. So ACCEPT /XYZZY/ 10 20 says search for 'XYZZY' starting in columns 10, 11, 12, 13, 14, and 15.

REJECT *

ACCEPT *

SELECT *

Provides a way of altering the default action at the bottom of the table. By default, a record that reaches the bottom of the table is accepted because it has passed the filtering process. However, you may want to reject all records that pass the filtering process.

Specifying ACCEPT, REJECT, or SELECT with an asterisk instead of a string means: match everything and perform the action.

By default, an ACCEPT * is generated. However, a REJECT * at the end of the table discards (rejects) all records that get that far. Do not code any other statements after a REJECT *, SELECT *, or ACCEPT *, because they will never be actioned.

Filter Table Example

The following sample filter table illustrates how to use the statements:

```
* SAMPLE FILTER TABLE FOR UTIL0037
*
* FILTER OUTPUT FROM IEBCOPY TO SHOW ERROR MSGS ONLY.
*
*
* NOTE THAT THIS IS FOR A PRINT FILE.  WE USE THE DEFAULT BASE WHICH
* MEANS THAT COLUMN 2 (AFTER CTL CHARS) IS LOGICAL COLUMN 1.
*
*
* REJECT PAGE HEADERS AND STATEMENT ECHOES (BLANK COL 1)
*
REJECT  / PAGE / 110
REJECT  / /      1
*
* SELECT END-OF-JOB MSG
*
ACCEPT  /IEB147I/
* SELECT ANY E MSGS (3 OR 4 DIGIT NUMBERS)
ACCEPT  /E/ 7
ACCEPT  /E/ 8
* REJECT THE REST
REJECT *
```

Sample USER Exit: UTIL0038

To illustrate the USER*x* facility, a sample user exit program, UTIL0038, is supplied (in source form as well as compiled).

This section discusses the use of UTIL0038 (as supplied) to encrypt and decrypt data.

UTIL0038 Processing

UTIL0038 processes in two modes, depending on whether it is being used for input (USERI), or output (USERO). If called by USER (not USERI or USERO), the attempted OPEN fails.

UTIL0038 USERI Processing

When being used to process input, UTIL0038 reads from a data set (the *ddname* supplied in the PARM operand), decrypts records, and provides them to the application as input.

The *ddname* named on the PARM operand must reference a data set that was written as an output file by some other invocation of UTIL0038.

The DCB attributes for the data set that UTIL0038 reads are RECFM=VBS, LRECL=32760, BLKSIZE=6233. These are the same attributes that it writes.

The DCB attributes for the DD SUBSYS data set are irrelevant. However, if records of an incorrect length are returned, I/O errors result.

UTIL0038 USERO Processing

When used to process output, UTIL0038 writes to a data set (the *ddname* supplied as the value for the PARM operand), encrypting the written records (that is, those provided by the application as output).

The *ddname* supplied as the value for the PARM operand *must* reference a data set. The DCB attributes of this data set are forced to RECFM=VBS, LRECL=32760, BLKSIZE=6233.

The DCB attributes for the DD SUBSYS data set are irrelevant. UTIL0038 writes variable length records to its output file.

UTIL0038 Encryption

The encryption logic used is very simple and is only supplied to illustrate what is possible.

Encryption is to simply XOR (exclusive OR) the record against a table that contains the values X'FF' down to X'00'. This operation is reversible by simply redoing the XOR.

Note: This encryption technique is not foolproof. Do not attempt to use this program to provide data security.

Examining the source of UTIL0038 shows how to easily write USERI or USERO programs. It illustrates how to write the exit to be reentrant, how to use the various parameters supplied, how to return OPEN errors, and so on.

Chapter 5: Non-VTAM Terminal Support

This section contains the following topics:

Overview

There are times when you want to be able to use terminals to communicate with regions while VTAM is not available. For example, during system IPL, you might want to log on and run automation in full-screen mode without waiting for VTAM to become active.

A region can provide non-VTAM support in the following ways:

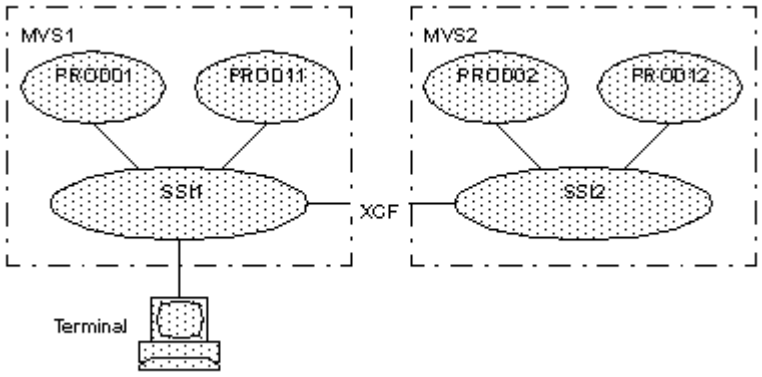
- By using a local terminal as a non-VTAM terminal
- By using Telnet

Implement Non-VTAM Terminal Support by Using a Local Terminal

Non-VTAM terminal support is implemented by using the SSI access method. This method supports communication between a local terminal attached to a program called NMSSI (SOLVE SSI) and up to 16 regions.

Sysplex Support

In a sysplex environment, you can use the cross-system coupling facility (XCF) to enable a local terminal to access a region on another system. The following illustration shows an example:



To register the SSI region to the XCF component, add the XCF=YES parameter in the SSISYSIN or SSIPARMS (if used) member.

Enable Terminal Support in SSI

An SSI startup parameter, TERMINALS={YES|NO}, is used to specify whether the NMSSI program is to provide support for non-VTAM terminals.

If TERMINALS=YES is specified, communications for terminals are initialized when the NMSSI program is initialized.

The TERMINALS parameter is specified in the SSISYSIN or SSIPARMS (if used) member. If you want to implement non-VTAM terminal support you need to set this parameter to YES.

Note: If you are running multiple SSI regions on a single system, only one of them may be set up for non-VTAM terminals.

You need to review other startup parameters for the SSI (in particular, the SSID parameter).

Specify Accessible Regions

To specify the regions that can be accessed by the terminals attached to an SSI region, add TERMACCESS parameters to the SSISYSIN or SSIPARMS (if used) member.

You can specify up to 16 TERMACCESS parameter statements.

Enable Automatic Logon to a Region

The automatic logon parameter for a non-VTAM terminal is set in the SSI parameter group.

When automatic logon is enabled for a terminal, the terminal is automatically logged on to the region specified in the first TERMACCESS parameter statement only.

TERMACCESS Parameter

A TERMACCESS parameter specifies the region that a non-VTAM terminal can access and the function key that is used to access the region.

`TERMACCESS=(PF`*`nn`*`,`*`region-id`*`,`*`description`*`)`

PF*nn*

Specifies the function key that is used to access the *region-id* region.

nn must be in the ranges 5 through 12 and 17 through 24.

region-id

Specifies the ID of a region that a terminal can access. If you registered the SSI regions with the XCF component in a sysplex environment, the region does not need to be on the same system.

region-id is specified in the SYSTEMID parameter group. The default is the value specified for the PRI= JCL parameter in the RUNSYSIN member.

description

Specifies a short description to identify the region that the function key accesses. This description is displayed on a terminal function key menu.

To represent a blank character, use an underscore.

Limits: 1 to 20 characters; quoted strings not supported

Example

The following example enables you to press PF11 to access the PROD region from a non-VTAM terminal.

```
TERMACCESS=(PF11,PROD,REGION_1)
```

Attach and Detach Terminals

The ATTACH and DETACH commands can be used to attach terminals to and detach terminals from NMSSI.

When you use these commands, you need to specify the device address of the terminal you want to attach or detach. NMSSI uses this address to dynamically allocate and deallocate the device.

Example: Attach a Device

The following example attaches device 4DF:

```
ATTACH 4DF
```

Example: Attach a Device and Notify Region

The following example attaches device 4DF and assigns it the name TERMNAME. The accessed region is notified of the attachment, and the user's details are specified.

```
ATTACH 4DF NAME=TERMNAME AUTOLOG=YES DATA=USERID PASSWORD OPT
```

Example: Detach a Device

The following example detaches the device 4DF:

```
DETACH 4DF
```

Example: Detach All Devices

The following example detaches all devices:

```
DETACH ALL
```

Define Terminal Names

Non-VTAM terminal support allows both 3-character and 4-character terminal addresses.

NMSSI assigns an eight-character symbolic name to a terminal that consists of a 1 to 5-byte prefix followed by the terminal device address.

If a terminal has a 4-character address and the first character is not zero (for example, 1FFF), you must limit the length of the prefix to four characters.

The symbolic name prefix defaults to $LOCL—you can, however, specify a different prefix using the TPREFIX operand in NMSSI startup parameters.

You can also specify a terminal name through the NAME operand of the ATTACH command. This name overrides the name specified in the TPREFIX operand for the terminal being attached.

Activate the SSI

You need to activate the SSI before you can use the non-VTAM terminal support facility.

Display Attached Terminals

The SHOW SSITERMS command displays terminals attached to NMSSI.

Control Non-VTAM Terminals Through Customization Parameters

Terminals that are attached to the region through NMSSI can also be controlled through the SSI parameter group. This parameter group provides a panel-driven interface for controlling non-VTAM terminal access (press F1 (Help) on the SSI Initialization Parameters panel for additional information). If NMSSI stops and restarts, you need to reapply the parameter group.

To avoid having to reapply the parameter group (that is, to ensure that the terminals always remain attached), you can update the NMSSI parameters by specifying CMD='ATTACH …' for each terminal.

Remote Device System (RDS) Considerations

When using RDS to control allocations of devices to multiple systems, you need to modify the buffer size on the terminal device definition. The buffer size must be at least 65. For example, to attach device 04B to the NMSSI, specify the following:

```
DEFINE DEV ADD=04B, DEVTYPE=3779, BUF=65
```

Use a Non-VTAM Terminal

A terminal that is connected through NMSSI initially displays a banner page.

This screen displays the NMSSI subsystem ID, the system name, the terminal name, and the following available function keys:

ENTER (Menu)

Displays the NMSSI menu that lists the function keys for accessing the defined regions.

F1 and F13 (Help)

Displays online help for the non-VTAM terminal facility.

F3, F4, F15, and F16 (Detach)

Detaches the terminal from NMSSI. Use the ATTACH command to reconnect the terminal to NMSSI.

F5 through F12 and F17 through F24 (Solve)

Passes the terminal to the appropriate region for logon if a TERMACCESS parameter is defined for that function key. The terminal bypasses EASINET if it is active and displays the logon screen. At this point the terminal functions as if it were connected through VTAM.

Access a Defined Region

You can access a defined region by pressing the appropriate function key from the NMSSI logo screen. You can also press ENTER to display the NMSSI menu that shows which function keys are defined and press the appropriate function key to access the region.

The NMSSI logo screen is displayed only when a terminal is first connected to NMSSI. Thereafter, the NMSSI menu is your NMSSI interface, for example:

```
SOLV MVS1           *** NMSSI MENU ***           Terminal: $LOCL4DF

Pfk  Nmid         Sysname  Status     Description
PF05 PROD0001     MVS1     IN-SESSION REGION_0001          *OUTPUT*
PF06 PROD0002     MVS2     STARTING   REGION_0002
PF07 -            -        -          -
PF08 -            -        -          -
PH09 -            -        -          -
PH10 -            -        -          -
PF11 PROD1001     MVS1     IN-SESSION REGION_1001          CURRENT
PF12 PROD1002     MVS2     INACTIVE   REGION_1002
PF17 -            -        -          -
PF18 -            -        -          -
PF19 -            -        -          -
PF20 -            -        -          -
PF21 -            -        -          -
PF22 -            -        -          -
PF23 -            -        -          -
PF24 -            -        -          -



F1/13=Help  F3/15=Logoff Current F4/16=Detach  Enter/Sysreq=Current
```

Use the SYSREQ Key

Use the SYSREQ key to switch between a region and the NMSSI menu. By returning to the NMSSI menu, you can establish multiple sessions and access them any time by pressing the appropriate function key.

Terminate a Session

Press F3 (Logoff Current) to terminate the session to the last accessed region.

To terminate another region from the NMSSI menu

1. Press the appropriate function key to access the region to make it current.
2. Press SYSREQ to return to the menu.
3. Press F3 to terminate the session.

Session Status

Sessions displayed on the NMSSI menu can have the following statuses:

CONNECTED

Indicates that the region is connected to an SSI region.

INACTIVE

Indicates that the region is not connected to an SSI region.

IN-SESSION

Indicates that the terminal has an active session with the region.

STARTING

Indicates that the terminal is establishing a session with the region.

Implement Non-VTAM Terminal Support by Using Telnet

You can customize the TCP/IP interface to support Telnet connections.

To enable Telnet access

1. Enter the **/PARMS** shortcut at the prompt.

   The region initialization parameter groups are listed.

2. Enter **F TELNETSRVR** to find the TELNETSRVR parameter group.

3. Enter **U** beside the parameter group, and complete the fields as follows:

   Allow TELNET Connections?

   Enter **Yes**.

   Port 1

   Enter **Shared** to use the port number specified in the Inbound Connections Port field of the SOCKETS parameter group.

   Note: If Shared is not suitable (for example, to permit or prevent access through a firewall), you can specify up to five port numbers.

   Leave the other fields at their default values.

4. Press F6 (Action).

   Telnet access is enabled.

5. Press F3 (File).

   Parameter values are saved so that Telnet access is enabled during region initialization.

After you have enabled Telnet access, users can use Telnet to access the region by using the specified port numbers.

Note: For information about the parameters that you can use to customize Telnet access, see the online help.

Connect to a Region by Using Telnet

To access a region that supports Telnet connections, use the IP address and port number that have been set up.

To get the address and port number, enter **SHOW TCPIP** in the region to which you plan to connect.

Index

R

S

T

U

V

W