CRAY RESEARCH, INC.

CRAY® COMPUTER SYSTEMS

SUPERLINK/MVS
LOGIC LIBRARY VOLUME 2:
CONTROL FUNCTIONAL UNIT

SI-0182

RECORD OF REVISION

CRAY RESEARCH, INC.

PUBLICATION NUMBER SI-0182

Each time this manual is revised and reprinted, all changes issued against the previous version are incorporated into the new version and the new version is assigned an alphabetic level.

Every page changed by a reprint with revision has the revision level in the lower righthand corner. Changes to part of a page are noted by a change bar in the margin directly opposite the change. A change bar in the margin opposite the page number indicates that the entire page is new. If the manual is rewritten, the revision level changes but the manual does not contain change bars.

Requests for copies of Cray Research, Inc. publications should be directed to the Distribution Center and comments about these publications should be directed to:

CRAY RESEARCH, INC.
1345 Northland Drive
Mendota Heights, Minnesota 55120

Revision	Description
	October 1987 - Original printing.

CRAY PUBLICATIONS STANDARDS HAVE NOT BEEN APPLIED IN PRODUCING THIS DOCUMENT. AS AN INFORMAL DOCUMENT, ITS DISTRIBUTION IS LIMITED TO SITE ANALYSTS UPON REQUEST.

Requests for the following informal SUPERLINK/MVS 2.0 documents should be directed to Linda Hughes, Customer Services, 2360 Pilot Knob Road, Mendota Heights, Mn. 55120, (612- 681-5923):

- SUPERLINK/MVS Logic Library Volume 2: Control Functional Unit, SI-0182
- SUPERLINK/MVS Logic Library Volume 3: Network Access Method, SI-0183
- SUPERLINK/MVS Logic Library Volume 4: Applications, SI-0184

These volumes supplement SUPERLINK/MVS Logic Library Volume 1, SI-0181, a formal CRI publication. Volumes 2 through 4 are not packaged as formal Cray publications because they provide information at a dynamic level of detail; not all detail in these volumes can be guaranteed to reflect the product exactly as shipped, so standardization of these volumes has been intentionally withheld.

Note that Volumes 2 through 4 will not be applicable to future releases of SUPERLINK/MVS.

Preface

SUPERLINK/MVS logically links a CRAY X-MP or CRAY-1 computer system running COS 1.16 (or a later version) and an IBM or compatible computer system running MVS XA 2.0 with Job Entry Subsystem version 2 or 3 (JES2 or JES3.) SUPERLINK/MVS has 3 Functional Units. Each of them are described in an overview volume (The SUPERLINK Logic Library Volume 1, Product and Component Descriptions, CRI publication SI-0181) and also in a comprehensive volume devoted to that particular Functional Unit.

This publication is the second volume of the four-volume set. Volumes 3 and 4 describe the SUPERLINK Network Access Method (SLNET) and the SUPERLINK Applications Functional Unit (SLAPPL) respectively. This manual provides a comprehensive description of the SUPERLINK/MVS Control Functional Unit (SLCN). It is comprehensive because it contains both the general description of the functional unit found in Volume 1, and a detailed guide to the code for this functional unit.

All four volumes assume the reader is familiar with both the MVS and COS operating systems. Familiarity with the International Standards Organization/Open Systems Interconnect (ISO/OSI) model is helpful but not necessary. Related ISO/OSI documents are listed in the SUPERLINK Protocol Manual, publication SI-0175.

SUPERLINK Documentation

The following table is a library guide to the SUPERLINK manuals. The manuals are listed as they relate to user tasks. The CRI publication number is given in parentheses after each manual title.

User Tasks	Typical Audience	Recommended Manuals	Brief Description
Introducing SUPERLINK/MVS	Users, system planners	SUPERLINK/MVS General Information Manual (SI-0177)	Provides a general overview of the SUPERLINK/MVS product
Planning, installing, operating, and modifying SUPERLINK/MVS	System planners, systems programmers, operators	SUPERLINK/MVS Installation, Tuning, and Customization (SI-0180)	Provides information for installing, configuring, customizing, tuning, and operating the product.
Programming and running applications	Users, application programmers	SUPERLINK/MVS User Guide (SI-0178)	Provides information on using the SUPERLINK interfaces
	Users, systems programmers, operators	SUPERLINK/MVS Messages (SI-0179)	Documents the messages produced by SUPERLINK/MVS

User Tasks	Typical Audience	Recommended Manuals	Brief Description
Diagnosis	Systems programmers, site analysts, users, operators	SUPERLINK/MVS Logic Library Volume 1: Product and Components Description (SI-0181) SUPERLINK/MVS Logic Library Volume 2: Control Functional Unit (SI-0182) SUPERLINK/MVS Logic Library Volume 3: Network Access Method (SI-0183) SUPERLINK/MVS Logic Library Volume 4: Applications (SI-0184) SUPERLINK/MVS Messages (SI-0179)	Provides an internal overview of the SUPERLINK components on MVS and a detailed logic description for each.
Implementation	Software programmers, systems programmers	SUPERLINK Protocol Information Manual (SI-0175)	Provides the necessary information to implement a network access method that will interwork with SUPERLINK/COS.

Conventions

The following conventions are used throughout this manual:

Convention	*Description*
< = = = = = = = >	In hierarchical structure diagrams, this symbol represents an error trapping routine that may be invoked from any place by the operating system services when an abend occurs during the execution of the code. No calls are made explicitly by the module from which the error trapping routine was invoked; the invoking module has nominated the error routine to handle the errors. In flow diagrams, this symbol shows the flow of data between two programs that do not have a normal transfer of control during error free processing.

Convention	*Description*
S@*xxxxxx*	Headings that relate to information about subcomponents or modules are preceded by the subcomponent or module name in the format S@*xxxxxx* *xxxxxx* specifies the specific subcomponent or module name.
italics	Variables are italicized.

Reader Comments

If you have comments about the technical accuracy, contents, or organization of this manual, please tell us. You have several options that you can use to notify us:

- Call our Technical Publications department directly at (612) 681-5729 during normal business hours
- Use the Reader's Comment form at the back of this manual
- Write to us at the following address:

  Cray Research, Inc.
  Technical Publications Department
  1345 Northland Drive
  Mendota Heights, MN 55120

We value your comments and assure a prompt response.

Table of Contents

List of Illustrations

List of Tables

List of Diagrams

1. Introduction

SUPERLINK/MVS is a general purpose, non-station communications product designed to link a CRAY X-MP or CRAY-1 computer system to an IBM or compatible computer system. The communications software used to perform this integration is based on the International Standards Organization/Open Systems Interconnection (ISO/OSI) model.

SUPERLINK/MVS addresses the need for random and sequential data access and distributed applications processing. It enables user applications running on the Cray mainframe to communicate with applications running on the IBM computer system. Through subroutine calls and programming macros, the SUPERLINK Network Access Method (SLNET) provides services that allow users to develop installation-specific, cross-system applications.

Architecture of SUPERLINK/MVS

The SUPERLINK/MVS product is composed of three Functional Units; SLCN, SLNET, and SLAPPL. SLCN utilizes the MVS Subsystem Interface; SLNET is an MVS Functional Subsystem (FSS) owned by SLCN; and SLAPPL is composed of many Cray-written applications that can use SUPERLINK/MVS services from within a batch job and/or TSO user address space. Both SLCN and SLNET are MVS-started task address spaces.

Figure 1 on page 1-2 provides a model of the SUPERLINK/MVS architecture.

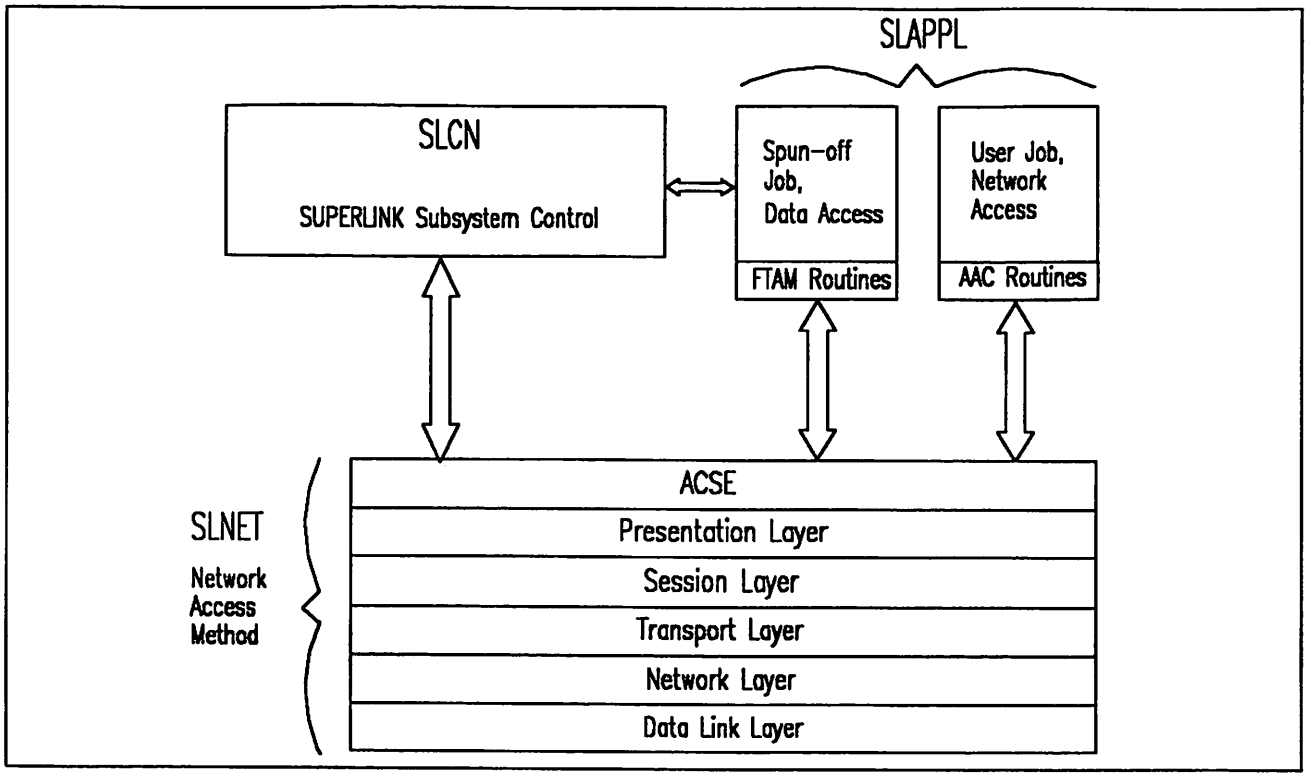


Figure 1. SUPERLINK/MVS Architecture

SUPERLINK Control Functional Unit (SLCN)

The SUPERLINK Control Functional Unit (SLCN) is an MVS subsystem that is configured using the MVS Subsystem Interface.

SLCN Components

The following components are resident within SLCN:

Component	*Description*
S@CC0000	SUPERLINK Product Management component The Product Management component is responsible for managing the Subsystem Interface between SLCN and the MVS operating system.

Component	*Description*
S@C1000	SUPERLINK Options Processor component The Options Processor component interprets initialization OPTIONS in a parameter library (PARMLIB) from options specified by the customer installation. These parameters specify which Functional Units must be active and which components, resident within those Functional Units, are to be supported for this execution of SUPERLINK/MVS. The Options Processor component also checks the validity of the parameters provided and uses them to form control blocks applicable to each Functional Unit.
S@CF0000	SUPERLINK Functional Subsystem Manager component The Functional Subsystem Manager component is responsible for the initialization, termination, and recovery of each SUPERLINK Functional Subsystem (FSS). This component also provides the means for inter-address space communication.
S@CO0000	SUPERLINK Product Operator component The Product Operator component supports commands that allow an operator to control the SUPERLINK/MVS product. The operator commands are entered from the multiple console support (MCS) console interface and are processed by this component.
S@C2000	SUPERLINK LOG Processor component The LOG Processor component allows all other SUPERLINK components to write messages to the SUPERLINK LOG and/or the MVS system log. These messages indicate progress status for normal events and provide reports of error conditions.
S@CI0000	SUPERLINK Management Interface component The Management Interface component provides a permanent connection between SLCN and the SUPERLINK for COS management function (management interface responder (SLMIR)). This component uses the services of the SUPERLINK Network Access Method to convey data between SLCN and SLMIR.
S@C9000	SUPERLINK Association Manager component The Association Manager component provides a set of services that support process creation on the MVS system for either data access requests or installation-developed cross-system AAC applications.
S@@M0000	SUPERLINK Message Processor component The Message Processor component provides a universal scheme for handling all the messages from components within the SUPERLINK/MVS product. Therefore, consistency is maintained between the messages issued by the product and the message descriptions provided by SUPERLINK/MVS Messages, publication SI-0179.

Component	*Description*
S@CC0SVC	SUPERLINK SVC component The SUPERLINK SVC component enables nonauthorized, problem program mode users to use Network Access Method services (SLNET) and Association Manager services (SLCN) through an SVC call. Control is returned to the caller in the original processing mode, thus preserving MVS system integrity. Specifically, this component allows callers from high-level languages (such as Fortran) or from unauthorized assembler programs to make AAC calls.
S@CC0URM	SUPERLINK User Resource Manager component The User Resource Manager component provides a global service for registering users of SUPERLINK/MVS resources by task and ascending address space identifier (ASID) order. Currently this global service is only used by the ACSE component of the SUPERLINK Network Access Method.

Figure 2 shows the hierarchical structure of components within SLCN.

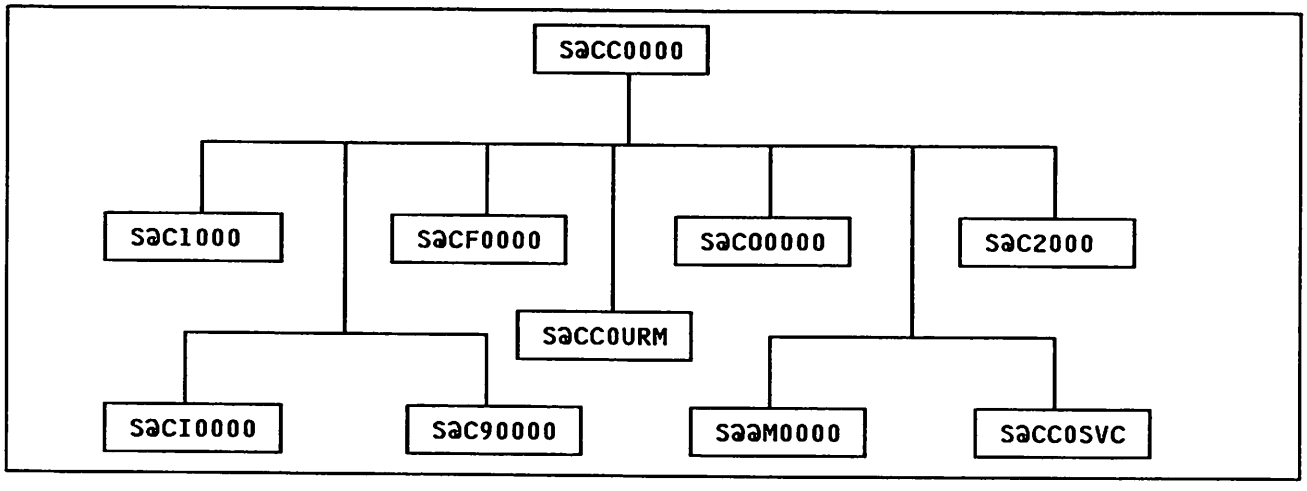


Figure 2. Component Structure of SLCN

SLCN External Interface

There are two external interfaces to SLCN:

- Initialization options validated and formatted by the Options Processor to define configuration parameters for the SLCN Functional Unit.
- Operator commands resident within the Product Operator component of SLCN. These commands allow an operator to do the following:
  - Start the SUPERLINK/MVS product
  - Display activity during product operation
  - Modify the SUPERLINK/MVS product configuration

- Shut down the SUPERLINK/MVS product

The SUPERLINK/MVS Installation, Tuning, and Customization Guide, publication SI-0180, provides a complete description of the SLCN initialization options and the supported operator commands.

SLCN Data Areas

The Options Processor component of SLCN creates control blocks describing the configuration parameters required for each Functional Unit. This subsection gives a brief description of these control blocks.

Data Area	*Description*
SC_CIOT	Control Initialization Options Table SC_CIOT determines the parameters that must be used to configure SLCN. It maintains pointers to the Initialization Options Tables for the other Functional Units within SUPERLINK/MVS.

A group of control blocks is necessary to support product management after SLCN is configured. SLCN maintains the following control blocks to support SUPERLINK/MVS product management:

Data Area	*Description*
SC_SSVT	Subsystem Vector Table MVS requires this control block because SLCN is configured as an MVS subsystem. The SC_SSVT functions as the anchor control block for the SUPERLINK/MVS product and describes the exit points within MVS that are supported by this subsystem. It is also the anchor for information concerning the FSSs owned by this MVS subsystem, and it holds information relating to the management of components within SLCN. SC_SSVT is addressable from any address space and lies in the common service area (CSA).
SC_GST	Global Service Table SLCN provides global services (for example, message services) for the SUPERLINK/MVS product. SLCN maintains a table of common storage locations that allows other address spaces within SUPERLINK/MVS to locate the routines that will satisfy the global services.
SC_SLASVT	Address Space Vector Table This table monitors the address spaces and the tasks within those address spaces that make use of SUPERLINK/MVS resources. SC_SLASVT consists of one entry per address space, in ascending address space identifier (ASID) order and a special entry for ASID value 0. A null value in one of these pointers indicates that the associated address space is not making use of SUPERLINK/MVS resources. A non-null value is a pointer to a chain of control blocks (one per Task Control Block (TCB) per association and so on) called User Registration Elements (SC_UREs). These are used to keep track of resource utilization by task/ASID.

The Subsystem Interface processing routines create the following control block to notify the relevant component managers of specific events:

Data Area	*Description*
SC_OPCB	Operator Command Buffer The SUPERLINK Subsystem Interface routine for operator command processing creates an operator command buffer, SC_OPCB. SC_OPCB contains the operator command and information concerning point of origin. Once the control block is complete, it is added to a queue of operator command buffers awaiting processing by the Product Operator component.

"Appendix A. Data Area Descriptions" on page A-1 provides descriptions of the previous control blocks.

Manual Organization

Each section of this manual describes a component of SLCN. Data area formats and interface macros are described in appendixes A and B respectively.

2. SUPERLINK Product Management Component

The Product Management component manages the central point for error handling, initialization, and termination of SUPERLINK components. It also manages the Subsystem Interface between SLCN and the MVS operating system.

Product Management Module Structure

The modules described in the following list manage the SUPERLINK Functional Units.

Module	*Function*
S@CC0000	Root module for SLCN
S@CC0RIM	Subsystem resource initialization routine, given control during MVS initial program loader (IPL) sequence
S@CC0SSI	Subsystem Interface initialization routine
S@CC0SST	Subsystem Interface termination routine
S@CC0EOT	Subsystem Interface support routine for end-of-task conditions
S@CC0EOM	Subsystem Interface support routine for end-of-memory conditions
S@CC0S34	Subsystem Interface support routine for SVC34 conditions such as operator command notification
S@CC0FSS	Subsystem Interface support routine for FSS CONNECT/DISCONNECT requests
S@CC0FSI	FSS Interface support routine

Figure 3 on page 2-2 shows the hierarchical structure of the modules within the Product Management component.

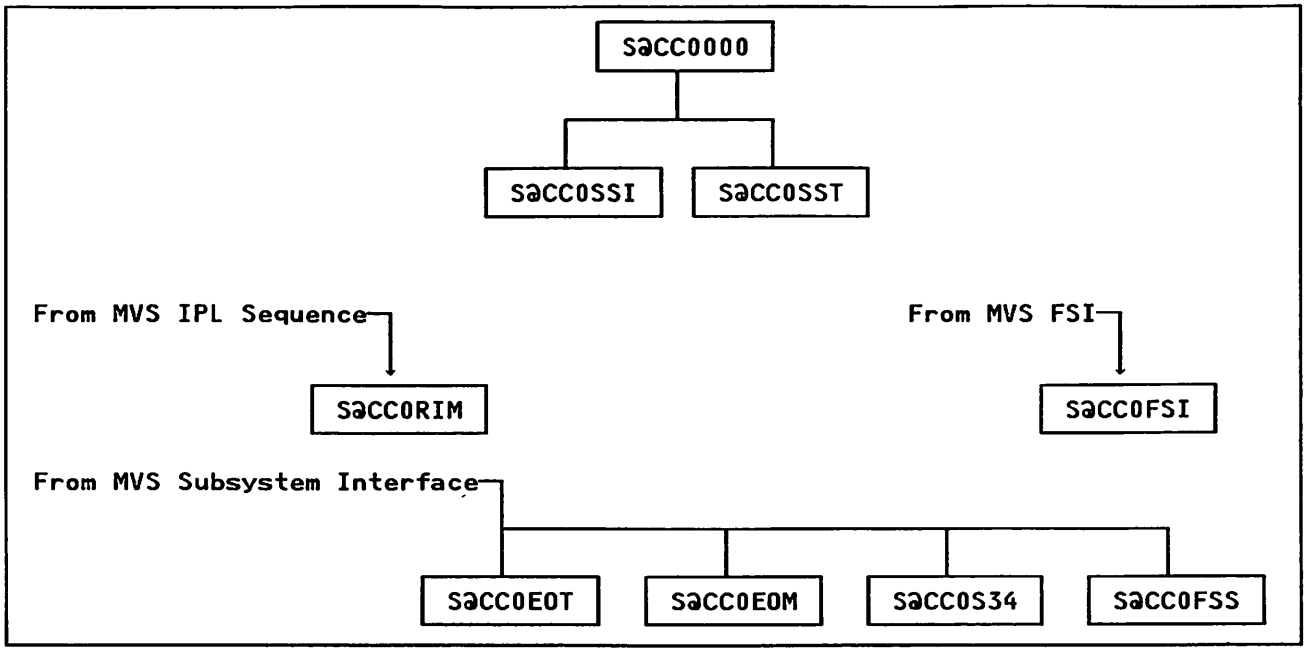


Figure 3. Module Structure of the Product Management Component

Product Management Services

SLCN is configured as an MVS subsystem. This configuration provides SUPERLINK/MVS with more than 50 exit points throughout MVS system processing at which SUPERLINK-supplied routines can gain control.

In addition to using the services provided by the Subsystem Interface, the Product Management component provides a central point for the initialization and termination of components, and the handling of failing components at a high level within SLCN. After initialization, this component is idle unless another SUPERLINK component fails.

Product Management Interfaces

The Product Management component provides services that use the MVS Subsystem Interface.

MVS enforces a control block structure to support a started task as an MVS subsystem. Figure 4 on page 2-3 is a diagram of the Subsystem Interface control blocks.

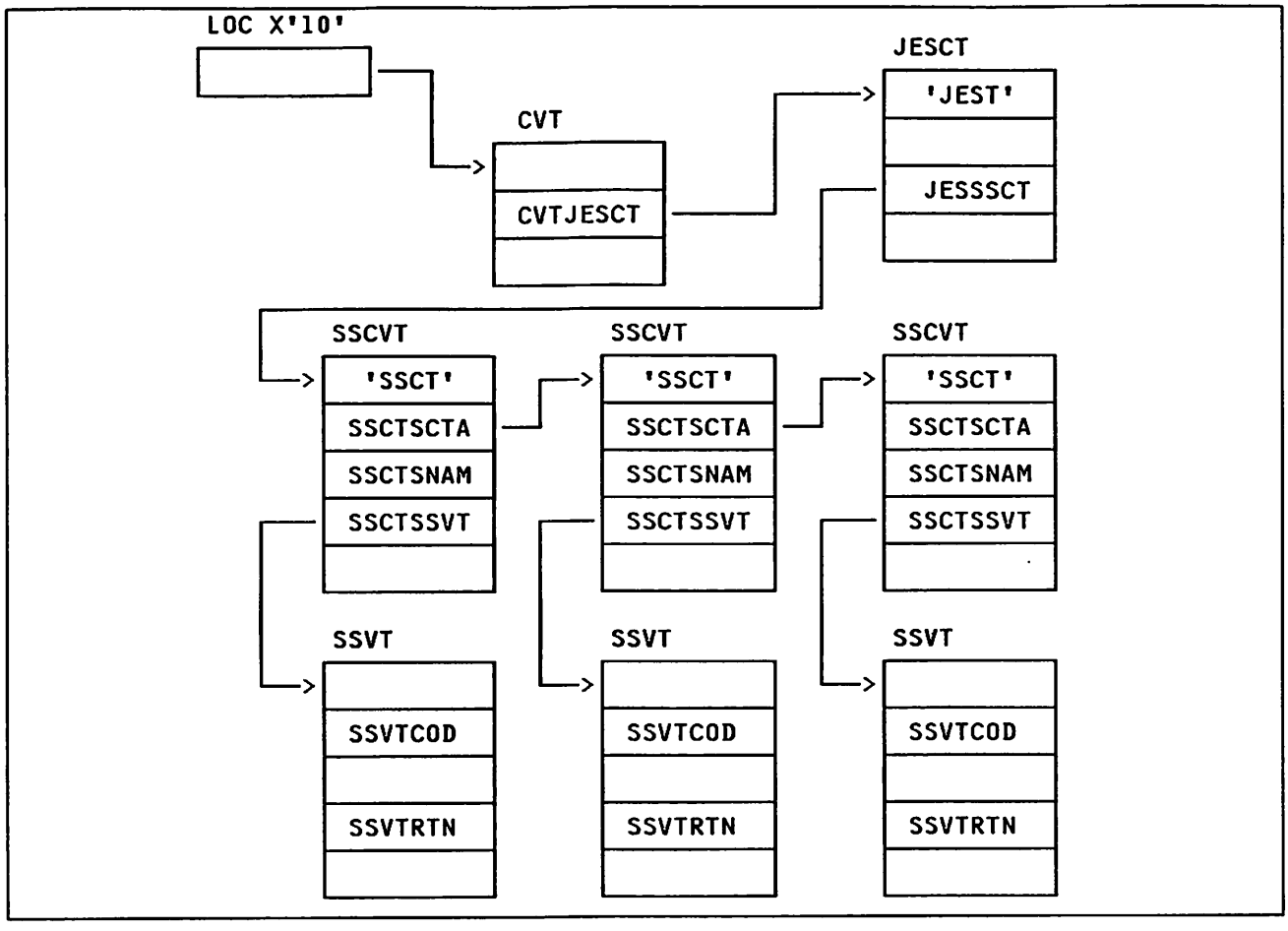


Figure 4. Subsystem Interface Control Blocks

Address spaces use Subsystem Interface requests to make requests for FSS connection and disconnection. MVS provides the control block structure required to satisfy these requests. Figure 5 on page 2-4 shows the control block structure for Subsystem Interface request control blocks.

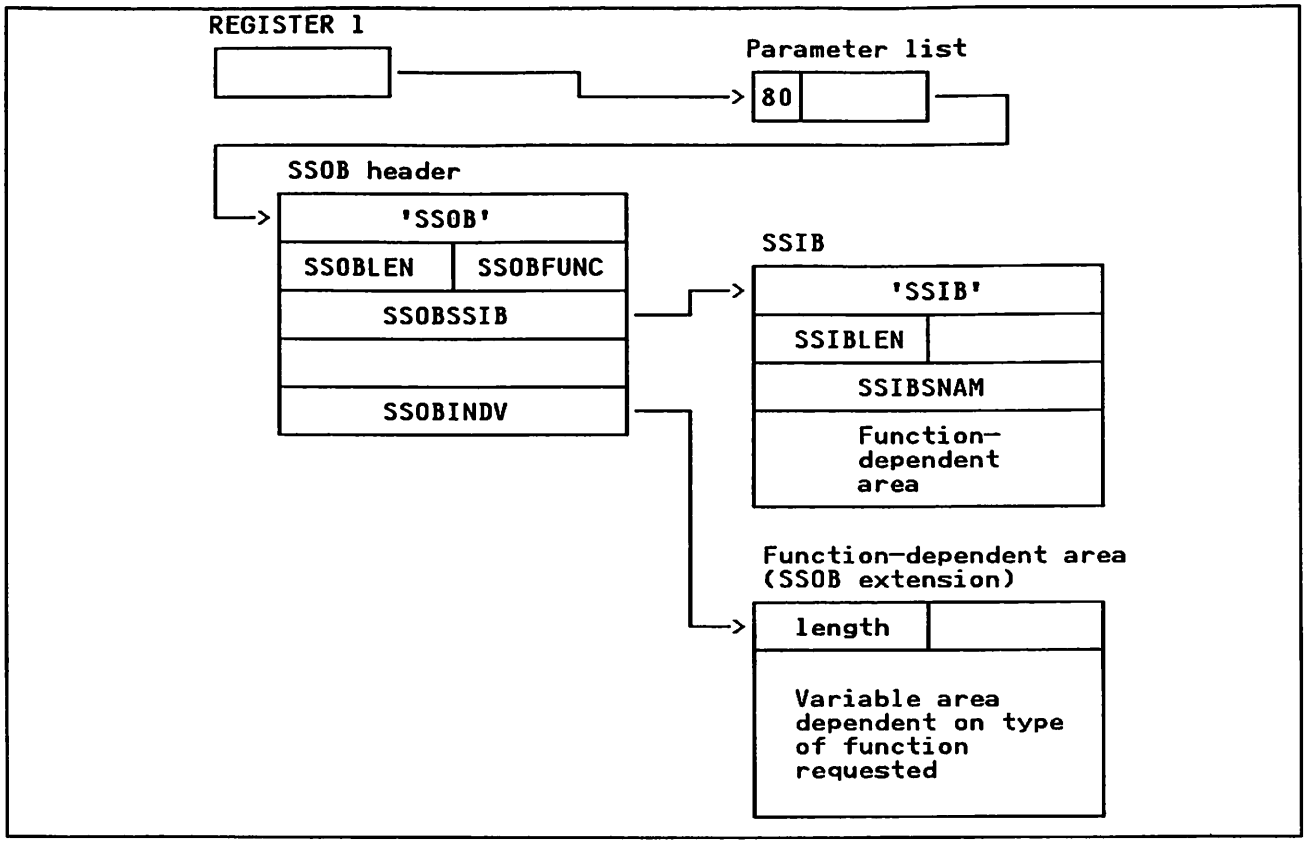


Figure 5. Control Block Structure for Subsystem Interface Requests

SUPERLINK/MVS provides routines for the following exit points:

Routine	*Description*
S@CC0EOT	**Notification of end-of-task** - Used specifically to indicate the termination of tasks with active SUPERLINK/MVS resources. Recovery of active resources is initiated.
S@CC0EOM	**Notification of end-of-address space** - Used specifically to indicate whether recovery actions are required. If an address space terminates while using SUPERLINK/MVS resources, the relevant Functional Unit must be notified.
S@CC0S34	**Notification of an operator command** - Used to support the MCS operator interface. SLCN inspects commands input from an operator console to identify the commands that must be processed by SUPERLINK/MVS.
S@CC0FSS	**FSS connect and disconnect** - Used to support SUPERLINK/MVS multi-address space management

Product Management Data Areas

The Product Management component makes use of the control blocks described under -- Heading id 'cndarea' unknown --. Specifically, the following data areas are used:

Data Area	*Description*
SC_SSVT	Subsystem Vector Table
SC_GST	Global Service Table
SC_SLASVT	Address Space Vector Table
SC_OPCB	Operator Command Buffer

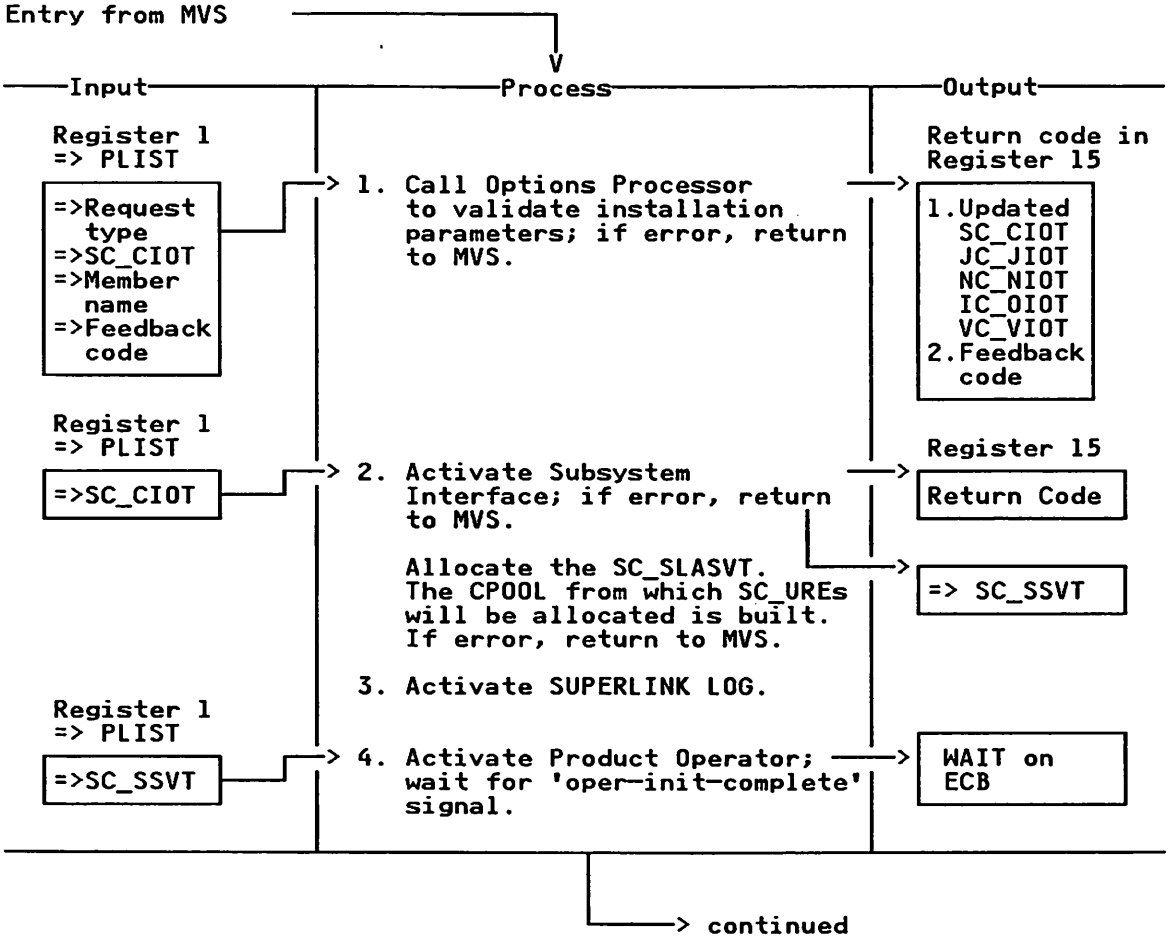
The Options Processor component creates control blocks describing the configuration parameters required for each Functional Unit. One of these control blocks, the Control Initialization Options Table (SC_CIOT), determines the parameters that must be used to configure SLCN. The SC_CIOT maintains pointers to the Initialization Options Tables for the other Functional Units within SUPERLINK/MVS.

Product Management Recovery

The Product Management component is notified of the failure of subordinate components within SLCN. If the failure is irrecoverable (the subordinate component will have made extensive recovery), the Product Management component brings SUPERLINK/MVS to a controlled halt.

Failures in the Product Management component itself are handled by an ESTAE routine that attempts to recover the internal error or produces a diagnostic dump if recovery is not possible.

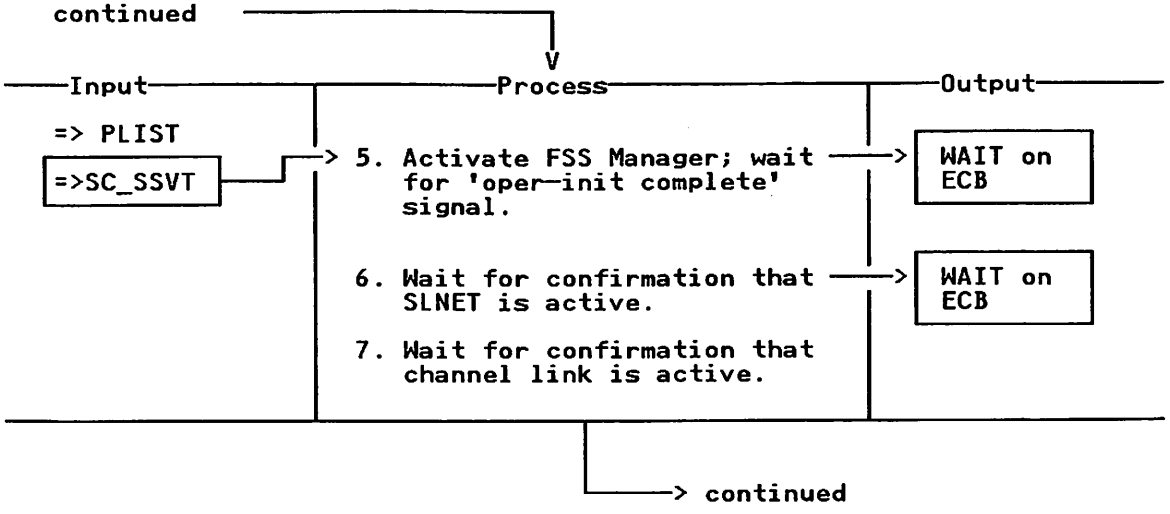
Diagram 2-1
S@CC0000 - Root Module of SLCN (part 1 of 4)



Extended Description

Explanation	*Module*	*Label*
1. The Control module passes a parameter list to the SUPERLINK/MVS Options Processor component. The parameter list contains the type of request for initialization. It also contains a pointer to the SC_CIOT and therefore to JC_JIOT, NC_NIOT, IC_OIOT, and VC_VIOT, which have all just been allocated, chained, and primed by S@CC0000. The list also contains the name of the member that contains the options parameters; this name defaults to SLINIT00, but may be overridden in the PARM field in the startup JCL. It returns the blocks completed from the initialization parameters with a return code and a feedback code describing any error.	S@CC0000	
2. Branch to control module S@CC0SSI to initialize the Subsystem Interface. A pointer to the SC_CIOT is passed so that S@CC0SSI initializes properly. A return code indicates whether or not initialization has been successful. The address of the SC_SSVT is returned in the PLIST upon a return code of 0 (successful completion) or 4 (successful recovery of an old SC_SSVT). The other codes represent failure; control is returned to MVS after releasing storage. The SC_SLASVT is GETMAINed and chained from SC_SSVT. SC_SLASVT is an array of fullword pointers with an index starting at 0 and ending at the maximum number of address spaces available on the system. It is initialized by clearing it to binary 0's. The pool of buffers for the SC_URE elements is built using CPOOL and is manipulated by the User Resource Manager component.	S@CC0000	
3. The SUPERLINK/MVS LOG Processor component is entered by its root module S@C2000 as an ATTACHed task. If the LOG processor task fails to initialize successfully, SUPERLINK/MVS will proceed to terminate.	S@CC0000	
4. Each component within SLCN must be activated in sequence. The control module must be ready for shutdown requests at a very early stage. The operator component (S@CO0000) is activated as an ATTACHed task and is passed a parameter list containing the address of the SC_SSVT that contains the anchor points for its queues. S@CC0000 WAITs for an ECB POSTed by the Product Operator component when it has initialized successfully. SLCN cannot be allowed to continue the initialization process until the Product Operator component has been activated, as this provides the mechanism for shutdown requests to be issued.	S@CC0000	

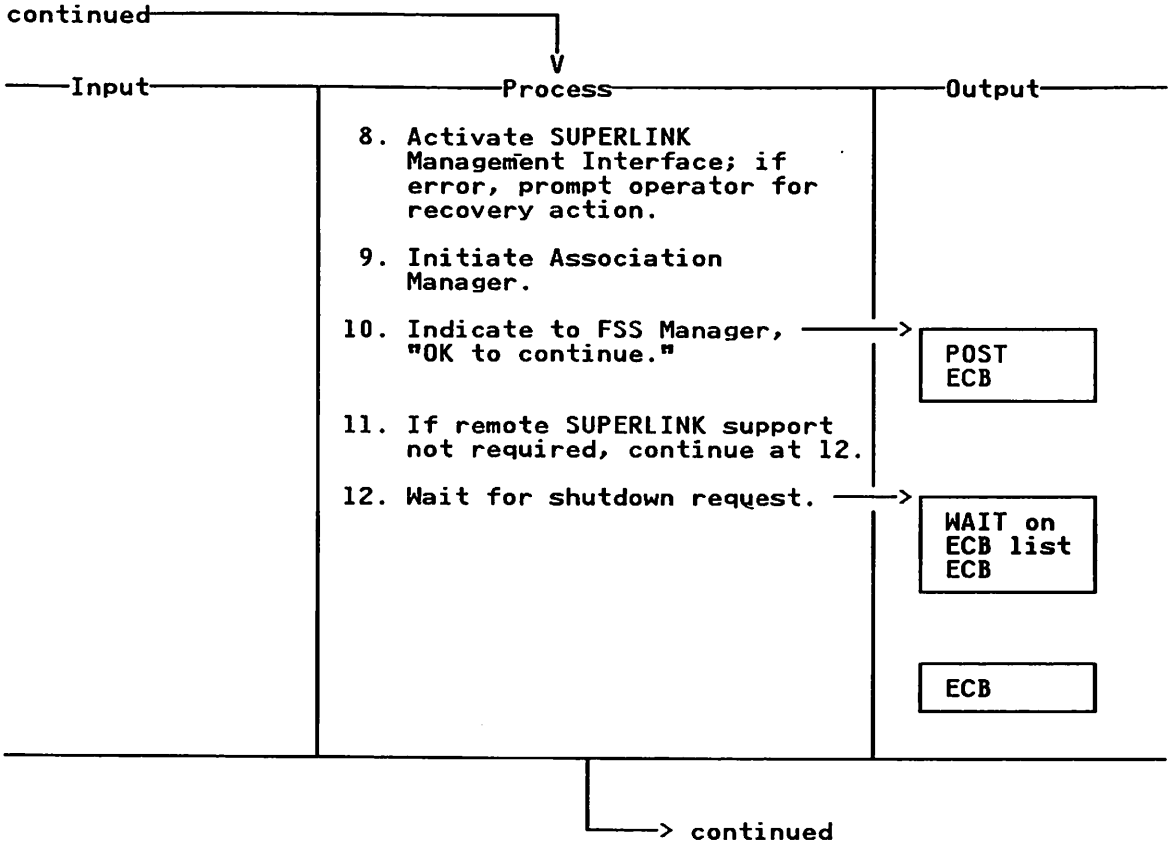
Diagram 2-2
S@CC0000 - Root Module of SLCN (part 2 of 4)



Extended Description

Explanation	*Module*	*Label*
5. The FSS Manager (S@CF0000) is activated as an ATTACHed task and is passed a parameter list containing the address of the SC_SSVT. S@CC0000 WAITs on an ECB list until the FSS Manager signals that it has initialized properly. Shutdown requests are valid from this point. SUPERLINK/MVS "listens" for possible shutdown requests while waiting for a signal that the FSS Manager is active.	S@CC0000	
6. The FSS Manager activates SLNET, the Network Access Method address space. SLNET must signal that it is initialized before SLCN processing can continue. S@CC0000 WAITs for an ECB list until SLNET is initialized, or else WAITs for a terminate request.	S@CC0000	
7. The channel connection to the Cray computer system must be allocated and active before the SUPERLINK/MVS Management Interface can be initialized. The MCS console operator requests activation of the channel connection. SLNET must signal to SLCN that initialization processing can continue. S@CC0000 again WAITs for a terminate request.	S@CC0000	

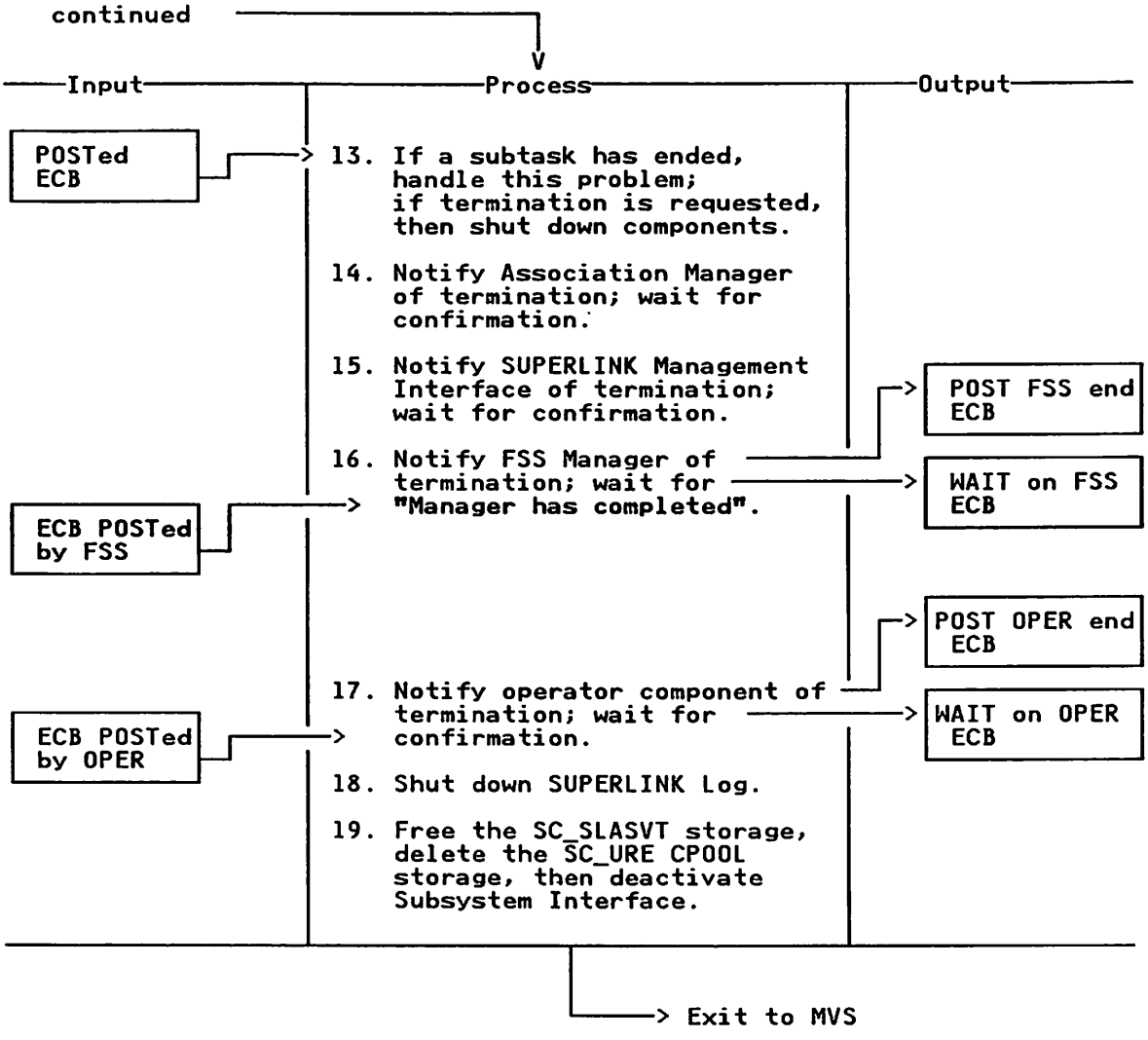
Diagram 2-3
S@CC0000 - Root Module of SLCN (part 3 of 4)



Extended Description

Explanation	*Module*	*Label*
8. The SUPERLINK/MVS Management Interface component is entered by root module S@CI0000.	S@CC0000	
9. The Association Manager handles all FTAM and AAC requests for association. Once SLNET is active, the Association Manager must be active to support session establishment requests. If the Association Manager cannot initialize successfully, SUPERLINK/MVS will terminate indicating the reason for failure.	S@CC0000	
10. The FSS Manager cannot initiate the application FSSs until the SUPERLINK/MVS Management Interface has been successfully established. The FSS Manager WAITs for an ECB POSTed by S@CC0000.	S@CC0000	
11. The SUPERLINK/MVS control initialization options determine the applicability of remote SUPERLINK/MVS support.	S@CC0000	
12. SUPERLINK/MVS is now initialized. S@CC0000 must WAIT for a shutdown signal for the complete product; an ECB list containing the termination ECB and ECBs is POSTed when any of the subtasks ATTACHed in S@CC0000 terminate prematurely.	S@CC0000	

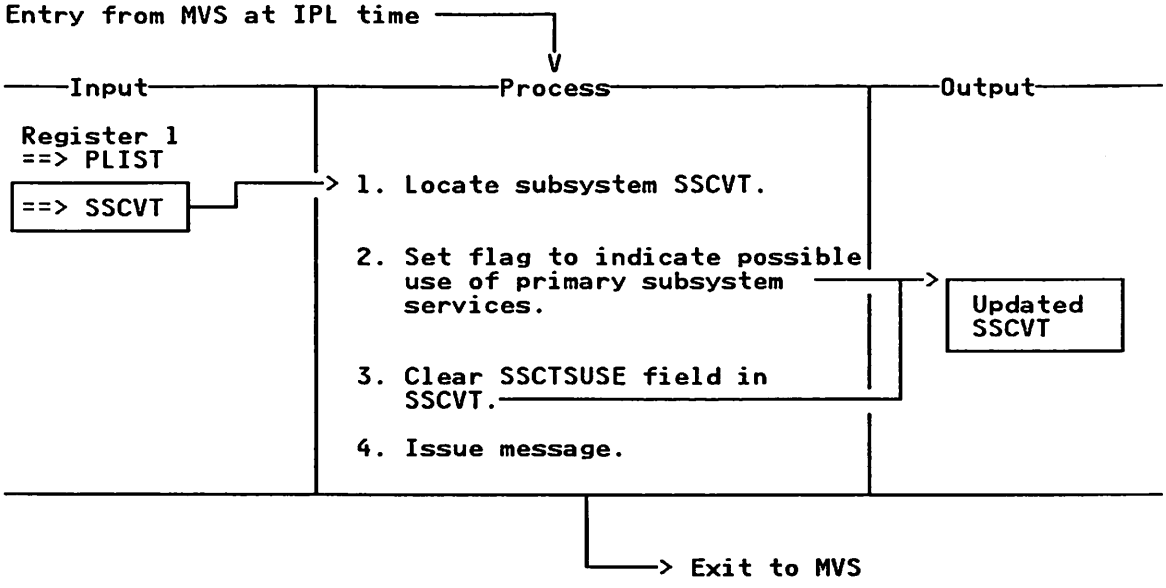
Diagram 2-4
S@CC0000 - Root Module of SLCN (part 4 of 4)



Extended Description

Explanation	*Module*	*Label*
13. If the ECB POSTed is for termination of one of the subtasks ATTACHed by this module, recovery action is attempted. If recovery is not possible, termination proceeds as though the terminate ECB was POSTed. If the ECB POSTed is for termination of the SUPERLINK/MVS system, the various components are shut down.	S@CC0000	
16. Notify the FSS Manager of shutdown by POSTing its terminate ECB. WAIT until it completes its shutdown processing via the POSTing of its shutdown-complete ECB.	S@CC0000	
17. Notify the Product Operator component of the shutdown by POSTing its terminate ECB. WAIT for it to complete its shutdown processing via the POSTing of its shutdown-complete ECB. To prevent any additional operator command from being eligible for SUPERLINK/MVS subsystem processing, the SC_SSVT must be updated to indicate that the subsystem is ending.	S@CC0000	
18. Branch to routine S@C2000 to shut down the SUPERLINK/MVS Log component.	S@CC0000	
19. Free the SC_SLASVT. Any SC_URE blocks chained to it are already free, since they are managed using CASE CPOOL storage management. Branch to routine S@CC0SST to deactivate the Subsystem Interface.	S@CC0000	

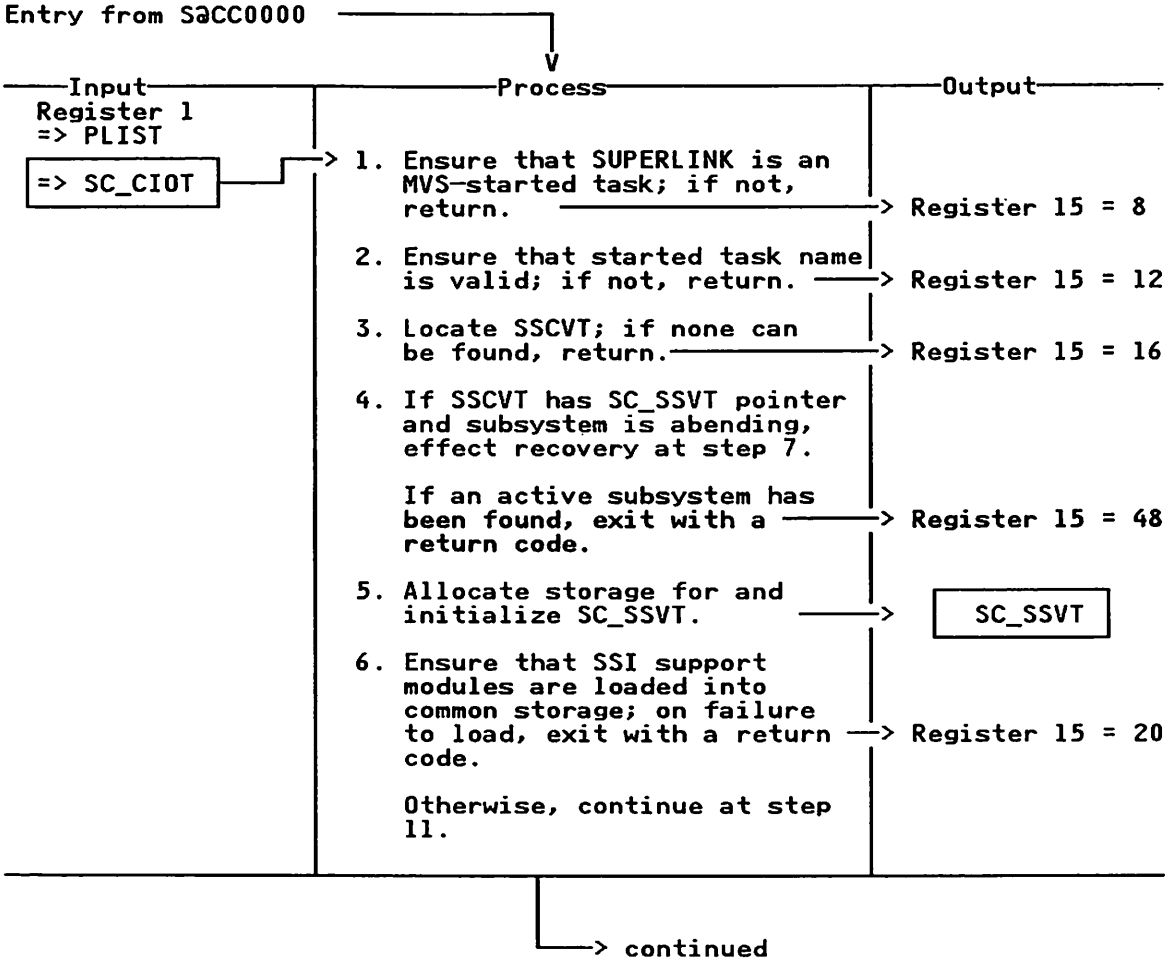
Diagram 2-5
S@CC0RIM - Sybsystem Resource Initialization Routine



Extended Description

Explanation	*Module*	*Label*
1. On entry at IPL, register 1 points to a parameter list containing a pointer to the SSCVT for this subsystem.	S@CC0RIM	
2. If a subsystem uses primary subsystem services (for example, SYSOUT processing), an indicator must be set in the SSCVT. The indicator determines whether or not Subsystem Interface requests are processed by the Master Scheduler address space.	S@CC0RIM	
3. To perform recovery of a SUPERLINK/MVS subsystem, the SSCTSUSE field in the SSCVT is put to special use. On IPL the field is cleared. When the subsystem is activated, the pointer to the SC_SSVT is placed in the SSCTSSVT field, indicating to MVS that the subsystem is active. The pointer is also placed in the SSCTSUSE field. Upon an abnormal end-of-address-space event for the subsystem, the SC_SSVT pointer is cleared from the SSCTSSVT field in the SSCVT, but the SSCTSUSE field pointer to the SC_SSVT is left unchanged. This enables the SUPERLINK/MVS subsystem initialization routine to perform recovery of the subsystem in the event of abnormal termination (for example, if the operator CANCELs).	S@CC0RIM	
4. Issue message to the operator indicating that the subsystem has been defined.	S@CC0RIM	

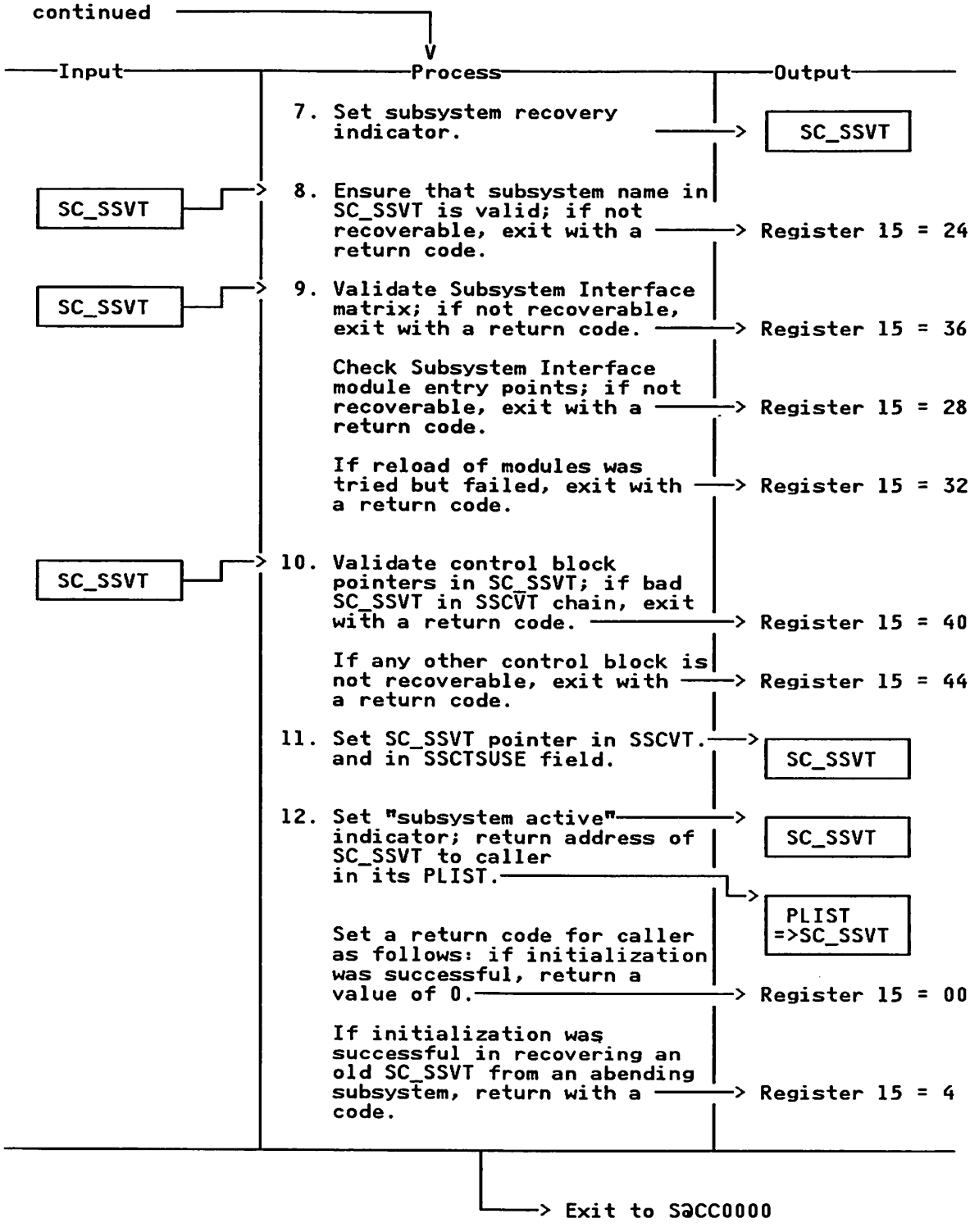
Diagram 2-6
S@CC0SSI - Sybsystem Interface Initialization Routine (part 1 of 2)



Extended Description

Explanation	*Module*	*Label*
1. The MVS Subsystem Interface can only be supported from a started task. If SUPERLINK/MVS is not a started task, initialization of the subsystem cannot continue, and an error code is returned to the caller in register 15.	S@CC0SSI	
2. An MVS subsystem must have a name between 1 and 4 characters long. If it is suitable, the started-task name is used for this purpose. Otherwise, an error code is returned to the caller in register 15.	S@CC0SSI	
3. At IPL, MVS builds an SSCVT for each defined subsystem. The chain of SSCVTs is scanned for one matching the subsystem name.	S@CC0SSI	
4. MVS recognizes an inactive subsystem by the SC_SSVT pointer field in the SSCVT. If this field is non-zero, the subsystem is either already active, or the previous execution of the subsystem abended. If the SC_SSVT pointer is zero, but the SSCTSUSE field is non-zero, a previous execution of the subsystem abended. At the abend, the end-of-memory routine deactivated the subsystem but left a pointer to the SC_SSVT in the SSCTSUSE field for recovery. If the SC_SSVT Subsystem Condition flag (refer to field SC_SSVT_COND in the SC_SSVT format described in section "Appendix A. Data Area Descriptions" on page A-1) indicates a "subsystem abending" status, subsystem recovery is attempted. The SC_SSVT pointer is used to access the existing SC_SSVT. If the SC_SSVT Subsystem Condition flag indicates "subsystem active", the request for initialization is erroneous. An error code is returned to the caller in register 15, indicating that no further processing is possible.	S@CC0SSI	
5. The SC_SSVT control block must reside in CSA (below the 16Mb line). The function matrix and entry point addresses are initialized, and information from the SC_CIOT is used to complete the initialization.	S@CC0SSI	
6. Subsystem Interface support routines must reside in common storage. The routines can either be loaded permanently in LPA or loaded into global storage at each subsystem initialization process. If the routines are not loaded correctly, the routine is exited with a return code after any allocated control blocks are freed.	S@CC0SSI	

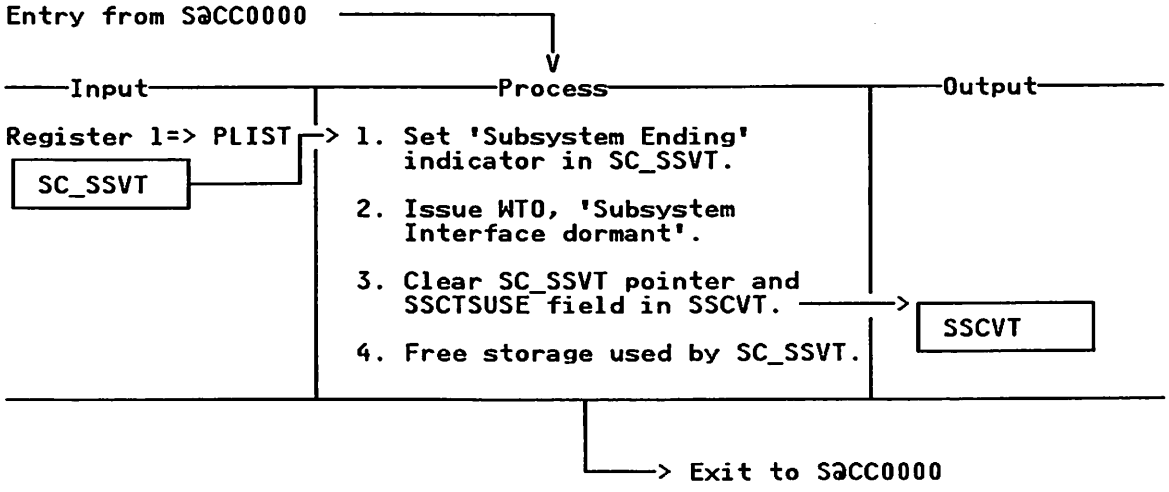
Diagram 2-7
S@CC0SSI - Sybsystem Interface Initialization Routine (part 2 of 2)



Extended Description

Explanation	*Module*	*Label*
8. For successful recovery of an existing SC_SSVT, the subsystem name in the SC_SSVT must be the same as that in the SSCVT.	S@CC0SSI	
9. The Subsystem Interface matrix in the SC_SSVT must be validated to ensure that the address of the interface routines are correct. If a routine is detected as erroneous, it is reloaded into global storage, and the relevant routine address is updated.	S@CC0SSI	
10. All the control block pointers in the SC_SSVT must be validated to ensure that the relevant control blocks are still available. The ASCB pointer must be updated. If an error is detected with any other control block field, the MVS operator is prompted with a WTOR to indicate either that recovery is to continue (that is, set field in error to 0) or that recovery is to be abandoned, the existing SC_SSVT deleted, and a new SC_SSVT created and initialized.	S@CC0SSI	
11. An update of the SC_SSVT pointer in the SSCVT control block ensures that MVS recognizes the subsystem as active. To facilitate recovery, the SSCVT user field is also updated to point to the SC_SSVT.	S@CC0SSI	

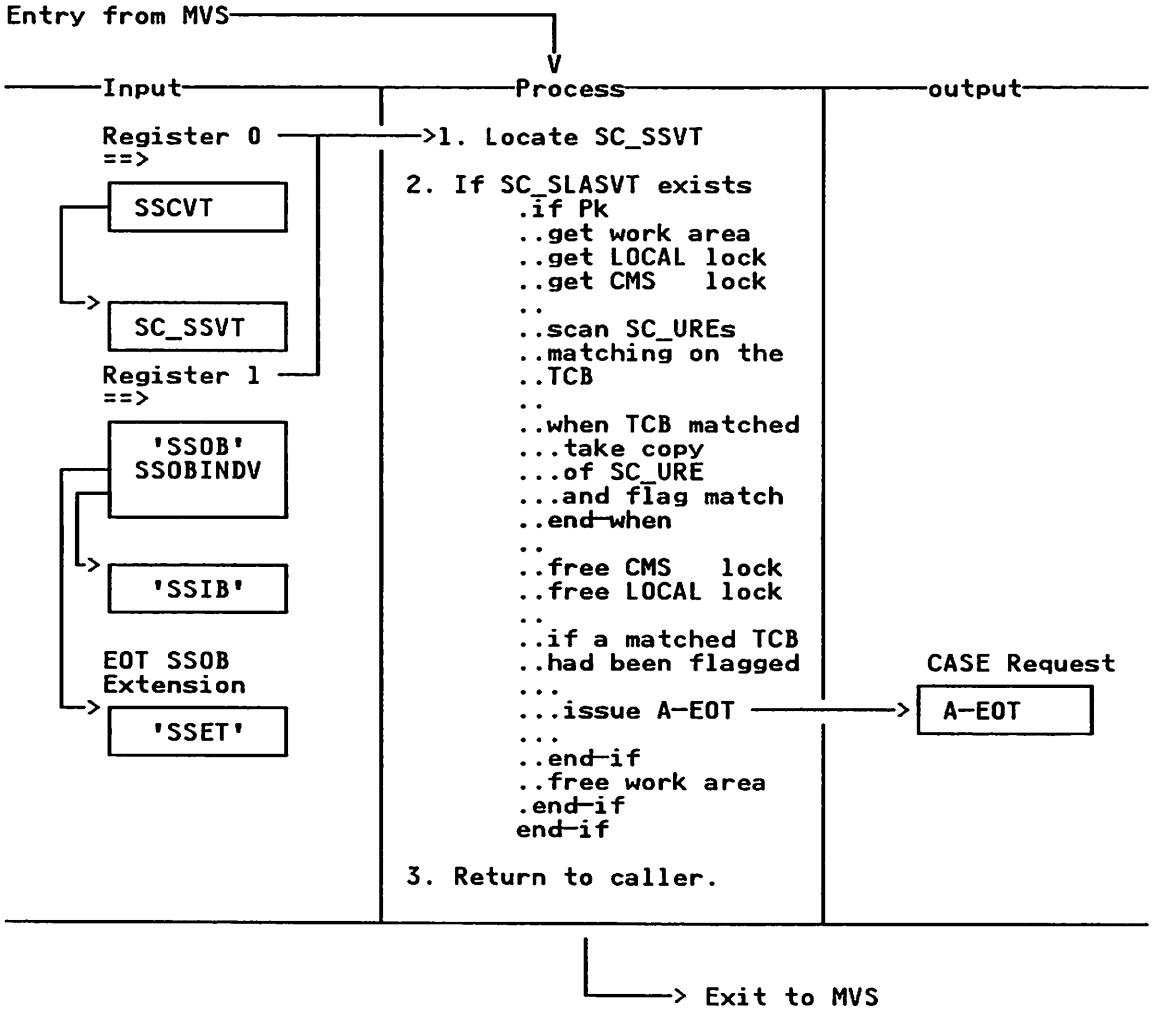
Diagram 2-8
S@CC0SST - Subsystem Interface Termination Routine



Extended Description

Explanation	*Module*	*Label*
3. Clearing the SC_SSVT pointer deactivates the Subsystem Interface for MVS. Clearing the SSCTSUSE field indicates that the subsystem terminated sucessfully. Recovery processing checks this field when the subsystem is next initialized.	S@CC0SST	

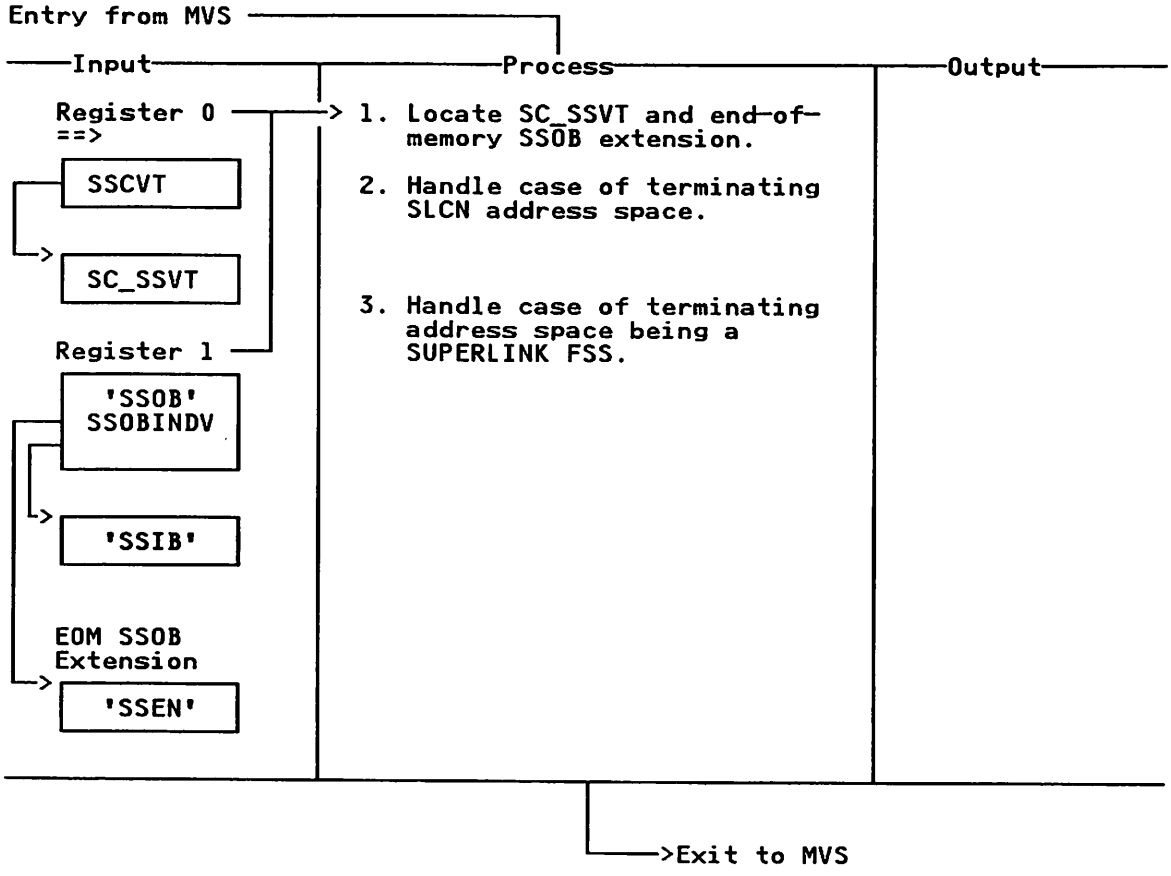
Diagram 2-9
S@CC0EOT - Sybsystem Interface End-of-Task Routine



Extended Description

Explanation	*Module*	*Label*
2. End-of-task processing concerns only tasks designated as SUPERLINK/MVS service users (that is, only tasks with an SC_URE entry chained from the appropriate entry in the SC_SLASVT table). To make best use of the time spent in this exit and under the CMS lock, the predicate, Pk, is checked: **Pk** The full-word pointer at offset 4 * k into the SC_SLASVT is not null (k is an integer in the range $1 \leq k \leq$ maximum ASID available). In most cases, the predicate is not true and no further action is required. If it is false, the CMS lock must be OBTAINed to search the queues of SC_UREs, and the queues must not be altered by other SUPERLINK/MVS components. If a match is made on the ASCB and TCB of the terminating task, then a CASE AL_ARE request area is built in the module's work area and an A-EOT CASE local primitive is executed to clean up the resources used by the task.	S@CC0EOT	
3. Return to caller.	S@CC0EOT	

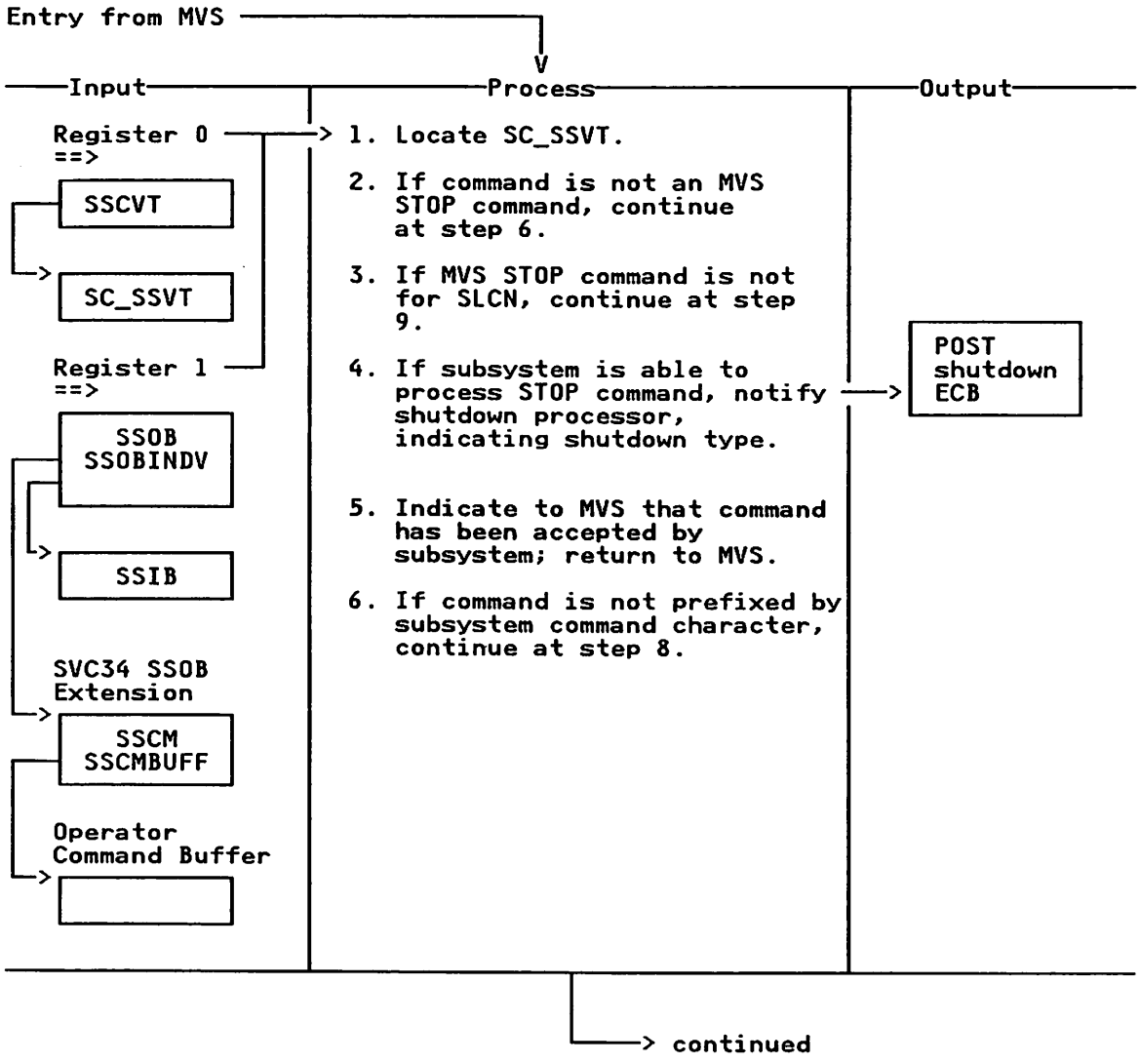
Diagram 2-10
S@CCEOM - Subsystem Interface End-of-Memory Routine



Extended Description

Explanation	*Module*	*Label*
1. End-of-memory processing is invoked from within the Master Scheduler address area. Pointers to the SSCVT and SSOB are provided on entry.	S@CC0EOM	
2. If the SC_SSVT pointer in the SSCVT is not null and the ASCB pointer in the SC_SSVT so chained matches that in the SSOB extension for the terminating address space, SUPERLINK/MVS is terminating without having cleaned up the Subsystem Interface. Flag the SC_SSVT as "abending"; clear the pointer to the SC_SSVT to deactivate the Subsystem Interface to MVS. This prevents abends, but leaves the SSCVT USER field pointing to the SC_SSVT. Consequently, recovery processing on a new initialization may recover the old control blocks.	S@CC0EOM	
3. If the SC_SSVT pointer in the SSCVT is not null, and the ASCB pointer in one of the SC_FSSCB blocks matches that in the SSOB extension for the terminating address space, an FSS is terminating. If the SC_FSSCB indicates that the FSS should not terminate, POST the FSS Manager task, indicating the unexpected termination of the FSS.	S@CC0EOM	

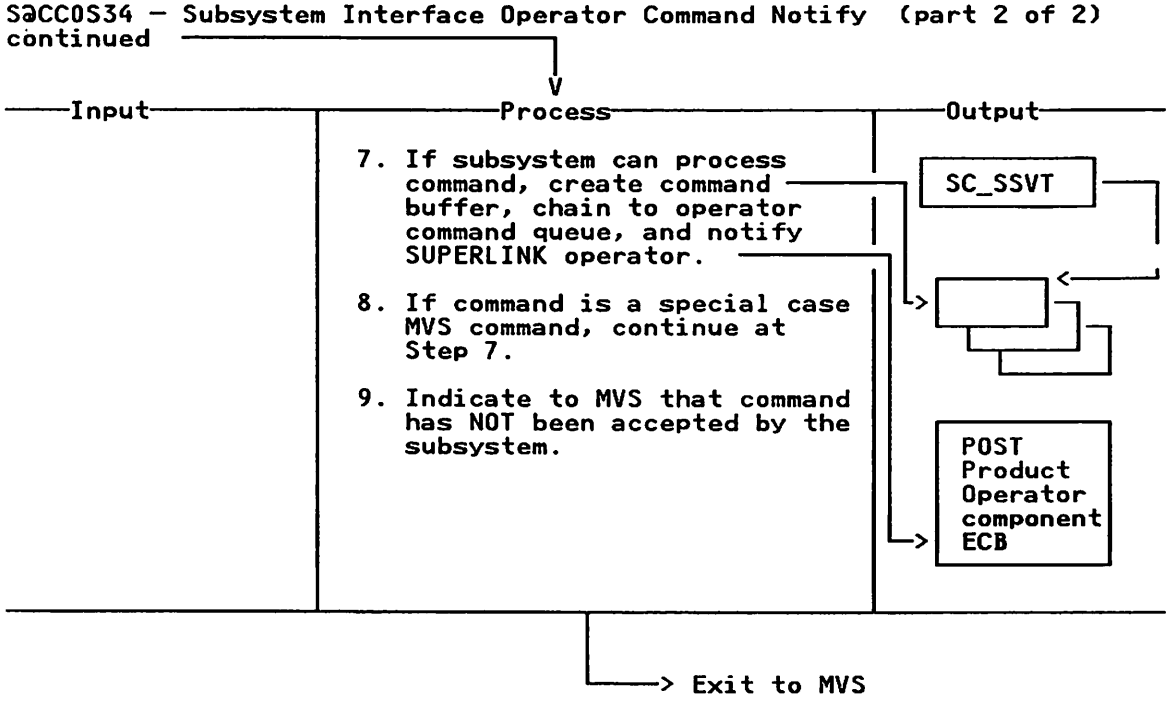
Diagram 2-11
S@CC0S34 - Subsystem Interface Support Routine (part 1 of 2)



Extended Description

Explanation	*Module*	*Label*
1. Upon entry, register 0 points to the SSCVT for this subsystem; the SC_SSVT is chained from the SSCVT. Register 1 points to the SSOB.	S@CC0S34	
2. The buffer containing the operator command and its length is chained from the SSOB extension (SSCM) for the Subsystem SVC34 operator command function call. The command is left-adjusted and padded on the right with 16 blanks.	S@CC0S34	
3. The name of the job being stopped is found by scanning the command buffer for a matching jobname.	S@CC0S34	
4. A subsystem must be in an active state to process an operator command. If the subsystem is still initializing or is terminating, the command is returned to MVS, indicating that it has not been accepted for processing. The shutdown type is normal, quick, or abort.	S@CC0S34	
5. A return code in the SSOB feedback code area, SSOBRETN, indicates to MVS that the command has been accepted by the subsystem. (EQUate SSCMSUBC is used.)	S@CC0S34	
6. The subsystem command character is installation-defined. The command character is available from the SC_SSVT_CMDCHAR field.	S@CC0S34	

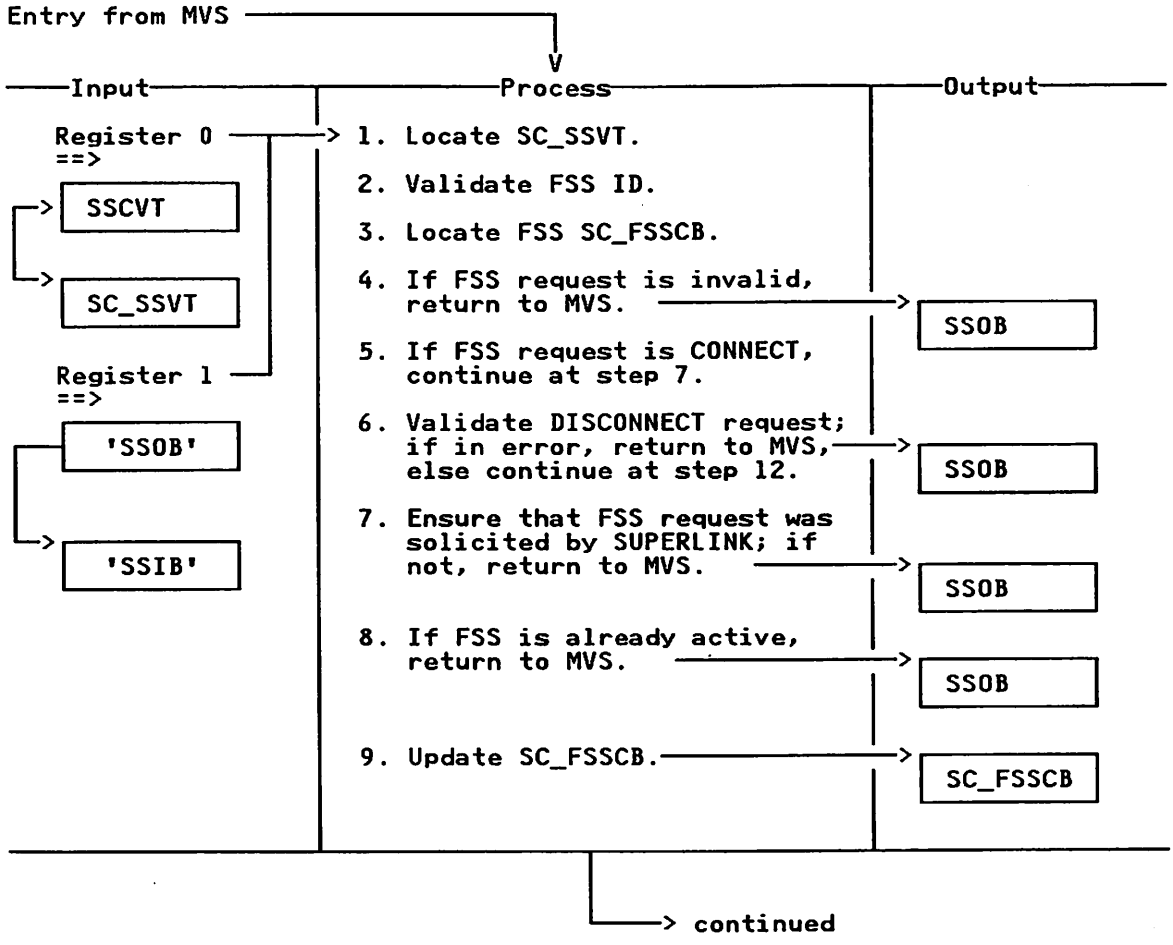
Diagram 2-12
S@CC0S34 - Subsystem Interface Support Routine (part 2 of 2)



Extended Description

Explanation	*Module*	*Label*
7. A subsystem must be in an active state to process an operator command. If the subsystem is either initializing or terminating and the Product Operator component is not available, the command is returned to MVS, indicating that the command has not been accepted for processing. The SUPERLINK/MVS Product Operator component manages all operator commands controlling SUPERLINK/MVS operation. A queue of commands to be processed is chained from the SC_SSVT. The operator processor is POSTed to perform the indicated actions.	S@CC0S34	
8. SUPERLINK/MVS may extend the MVS operator display command to provide a command interface familiar to the MVS operator. These commands are detected by the "Operator Command Notify" routine and chained to the operator command queue. The Product Operator processor is POSTed to perform the command.	S@CC0S34	
9. A return code in the SSOB feedback code area (SSOBRETN) indicates to MVS that the command has not been accepted by the subsystem. (SSCMSCMD is EQUed).	S@CC0S34	

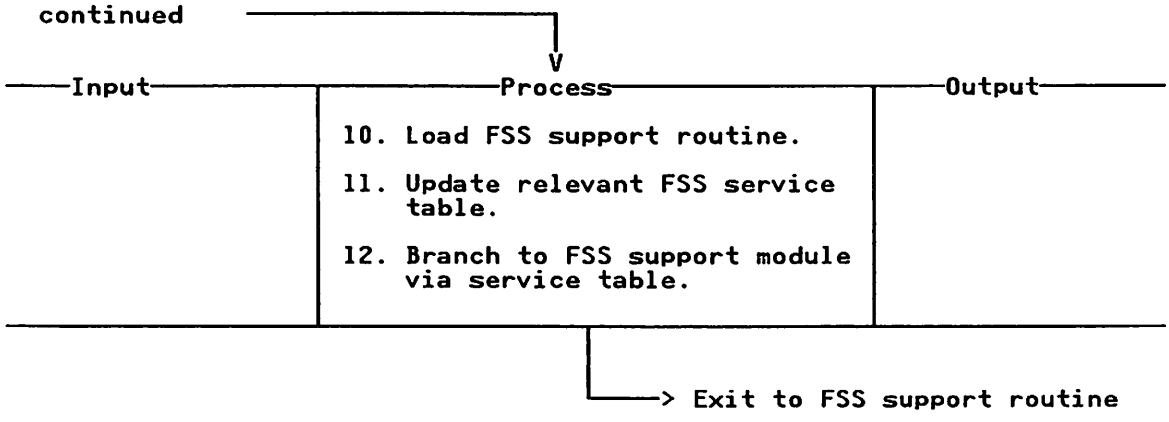
Diagram 2-13
S@CC0FSS - Subsystem Interface FSS CONNECT/DISCONNECT Routine (part 1 of 2)



Extended Description

Explanation	*Module*	*Label*
2. The MVS START command which initiates the FSS address space has a number of parameters. One parameter indicates the FSS ID of this FSS. The FSS-ID must be equal to or less than the maximum number of FSS address spaces supported by this FSS. This information is held in the SC_CIOT. The address spaces are allocated by routine SA@F0000. Check the pointer to see if S@CF0000 has allocated them.	S@CC0FSS	
3. All FSS control blocks are contiguous. The FSS-ID supplied with the START command locates the relevant SC_FSSCB.	S@CC0FSS	
4. Only FSS CONNECT or DISCONNECT requests are valid. Any other requests are treated as error conditions; in this case, the return code is set in the SC_SSOB and control is returned to MVS.	S@CC0FSS	
6. To ensure a valid FSS DISCONNECT request, the relevant FSS must be active, and the ASCB requested must be the ASCB supported by the SC_FSSCB. If either of these conditions is not satisfied, a return code is set in the SC_SSOB and control is returned to MVS.	S@CC0FSS	
7. The MVS START command which initiates the FSS address space also provides the originating subsystem name as a parameter. That name must agree with the subsystem name defined in the SC_SSVT, and the ASCB address of the subsystem must match the one stored in the SC_SSVT. If an error is detected, a return code is set in the SC_SSOB, and control is returned to MVS.	S@CC0FSS	
9. The subsystem address space and the FSS address space can update an SC_FSSCB concurrently. Therefore, care must be taken to use irreducible methods to update the SC_FSSCB whenever possible (for example, compare and swap logic). Updates to the SC_FSSCB include the ASCB address of the FSS and the TCB address of the mother task.	S@CC0FSS	

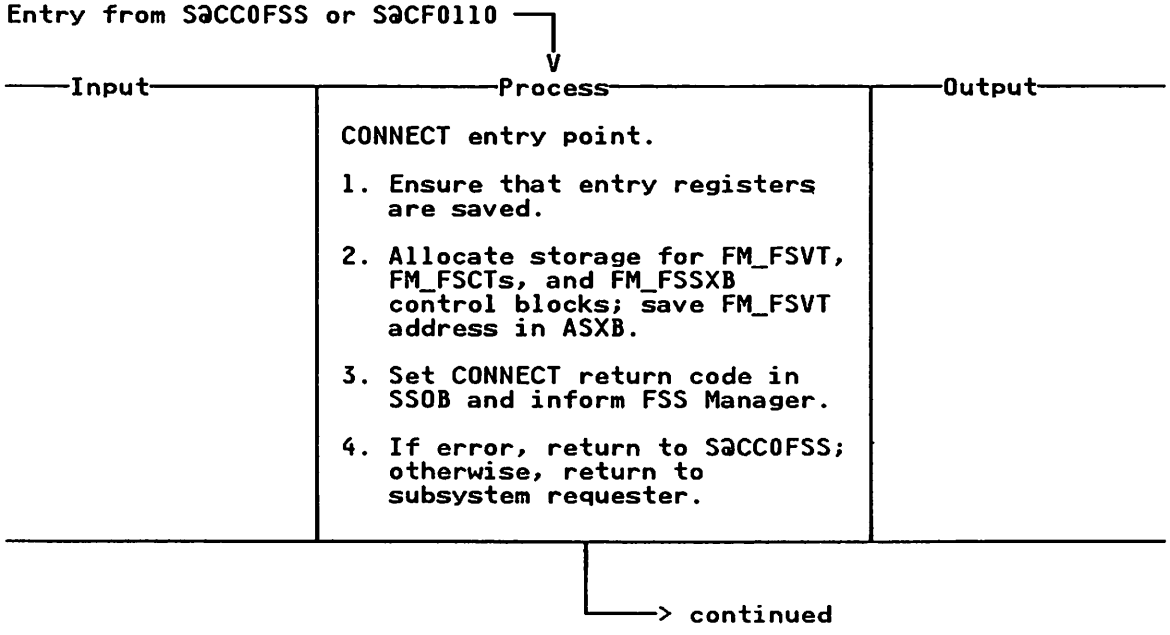
Diagram 2-14
S@CC0FSS - Subsystem Interface FSS CONNECT/DISCONNECT Routine (part 2 of 2)



Extended Description

Explanation	*Module*	*Label*
10. The FSS interface is supported through an FSS support module. Unlike the Subsystem Interface support module, which resides in common storage, the FSS support module resides in the private area of the FSS. The support module, named in the SC_CIOT, is loaded into LSQA. (The default FSS interface support module name is S@CC0FSI.) **Be careful to locate and load the correct module, since S@CC0FSS runs in the FSS address space.**	S@CC0FSS	
11. The addresses of the subroutines within the FSS support module are added to the "Common Service Table" for the FSS. Once all table updates have been made, control is given to the routine supporting either CONNECT or DISCONNECT processing.	S@CC0FSS	
12. The FSS support module completes FSS CONNECT or DISCONNECT processing.	S@CC0FSS	

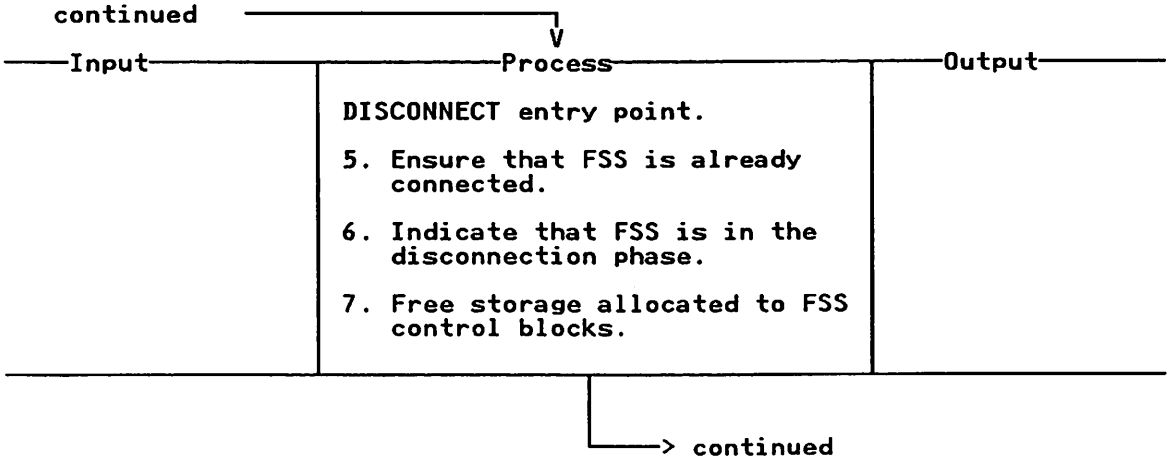
Diagram 2-15
S@CC0FSI - FSS Interface Support Routine (part 1 of 5)



Extended Description

Explanation	*Module*	*Label*
1. On return from this routine, control is regained not by the calling routine (S@C0FSS), but by the user requesting FSS CONNECT Subsystem Interface services. If an error condition is detected, control is returned to the Subsystem Interface routine, S@CC0FSS.	S@CC0FSI	
2. For an FSS CONNECT, the relevant control blocks to describe an FSS address space, (as determined by MVS), must be created and initialized. These control blocks are: • FM_FSVT - FSS Vector Table • FM_FSCT - FSS Control Table • FM_FSSXB - FSS extension control block Section "S@CF0000 - Data Areas" on page 4-3 provides a complete description of these control blocks.	S@CC0FSI	

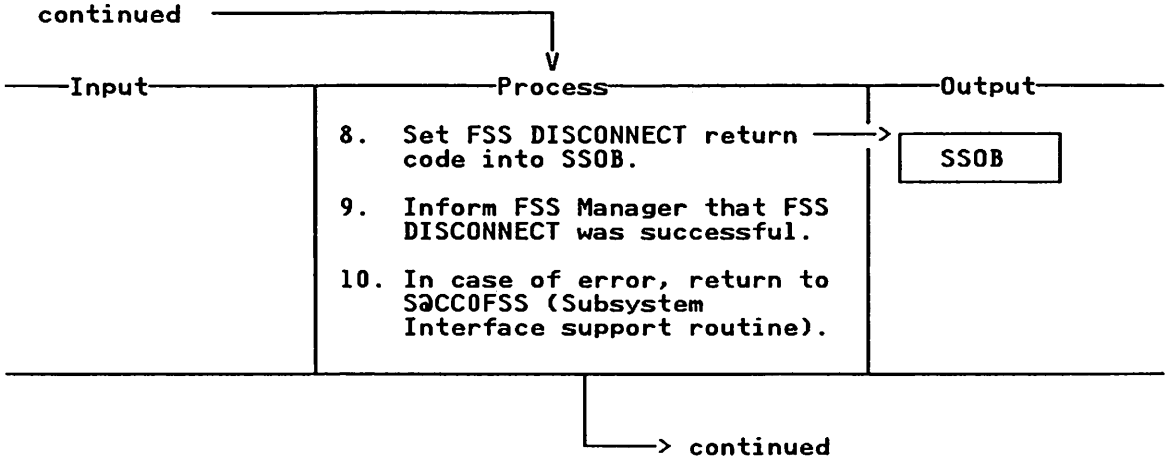
Diagram 2-16
S@CC0FSI - FSS Interface Support Routine (part 2 of 5)



Extended Description

Explanation	*Module*	*Label*
5. If a user makes an FSS DISCONNECT request, the FSS address space must have previously completed a successful FSS CONNECT. The SC_FSSCB data area includes a series of indicators describing the status of the relevant address space.	S@CC0FSI	
6. The storage allocated during CONNECT processing to hold the FM_FSVT, FM_FSCTs, and FM_FSSXB can now be released. The pointer to the FM_FSVT, held in the ASXB, must also be erased.	S@CC0FSI	

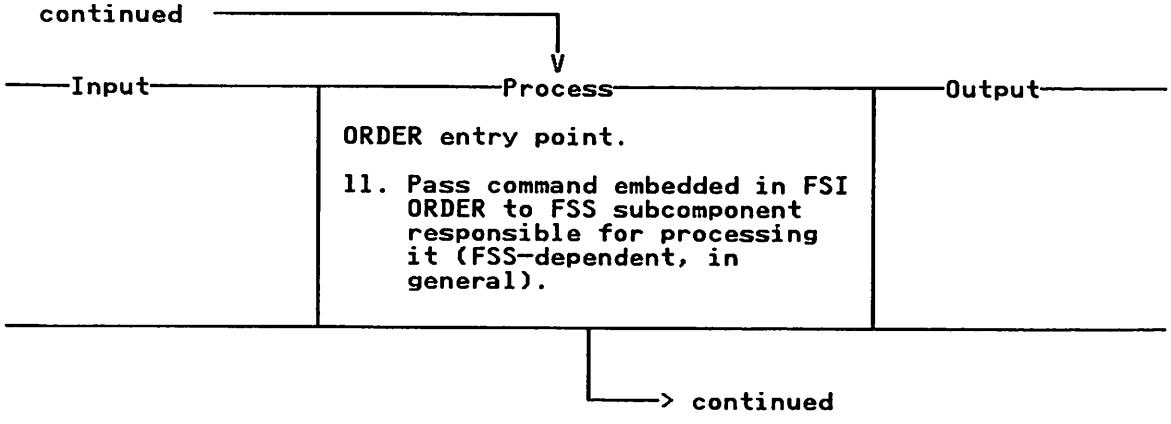
Diagram 2-17
S@CC0FSI - FSS Interface Support Routine (part 3 of 5)



Extended Description

Explanation	*Module*	*Label*
10. On entry to this routine, the Subsystem Interface support module registers were saved, and a pointer was located for the save area created on entry to the Subsystem Interface support routine. If an error is detected, control is returned to S@CC0FSS for further processing. During DISCONNECT, S@CC0FSI assumes responsibility for deleting the FSS interface routine from the system.	S@CC0FSI	

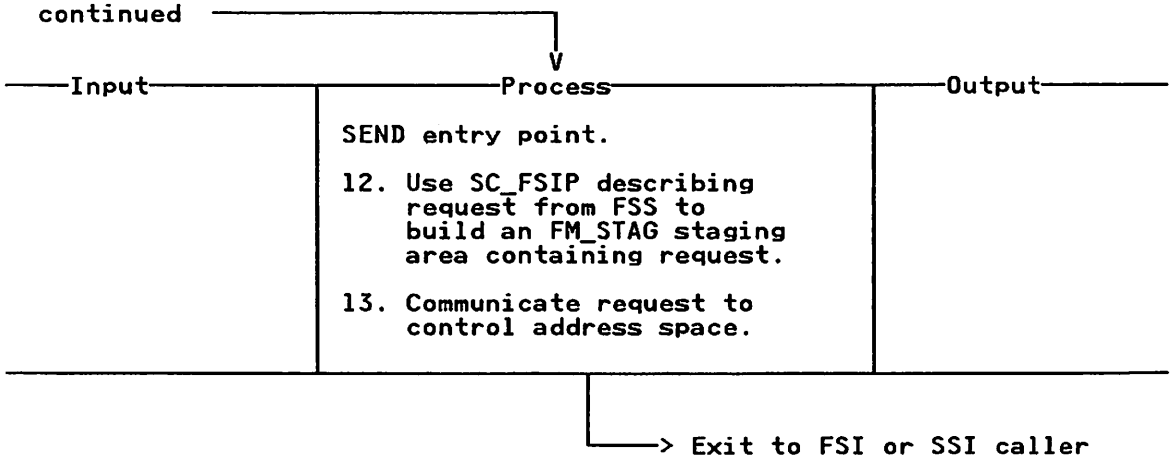
Diagram 2-18
S@CC0FSI - FSS interface Support Routine (part 4 of 5)



Extended Description

Explanation	*Module*	*Label*
11. Entry to the ORDER processing portion of the FSI routine is through routine S@CF0110, which was SCHEDULEd under an SRB from the control address space. S@CF0110 invoked this routine with a S@@FIREQ TYPE=ORDER macro call; therefore this code is effectively running in SRB mode. The SC_FSIP, from which the command being delivered to this FSS can be determined, is available to this routine. The actual mechanism of processing the command depends to a certain extent on the particular FSS; this routine simply places the command on a queue for processing. Replies are sent asynchronously from the FSS with the SEND communications primitive.	S@CC0FSI	

Diagram 2-19
S@CC0FSI - FSS Interface Support Routine (part 5 of 5)



Extended Description

Explanation	*Module*	*Label*
12. Entry is from an FSS-specific routine that issues an S@@FIREQ TYPE=SEND macro to send data to the control address space. The FSI parameter list SC_FSIP describing the request is available at this point.	S@CC0FSI	
13. This request is communicated to the control address space by embedding the request in a staging area in common storage (FM_STAG) and POSTing the "listen" task (S@CF0120) in the control address space.	S@CC0FSI	

Diagram 2-20
S@CC0TRT - Commonly Available ASCII/EBCDIC/ASCII Translate Tables

Input	Process	Output
	1. This is a non executable module containing two translate tables which are available for use throughout the SUPERLINK/MVS product. An EBCDIC to ASCII translate table is located at offset 0 into the the module. An ASCII to EBCDIC translate table is located at offset 256 into the module.	

Extended Description

Explanation	*Module*	*Label*
1. This module resides in common storage, is thus addressable from any address space and is available for use by any component of SUPERLINK/MVS which wishes to use ASCII/EBCDIC/ASCII translate tables. The address of the module may be found in the field SC_GST_TRANSLATE of the SC_GST (Global service table), which itself may be located from the field SC_SSVT_GST of the Subsystem Vector table (SC_SSVT).	S@CC0TRT	

3. SUPERLINK Options Processor Component

The SUPERLINK/MVS product is configured using initialization options supplied by the customer installation. These initialization options define the SUPERLINK Functional Units that must be active, and specify which components, resident in those Functional Units, are to be supported for this execution of SUPERLINK/MVS. The Options Processor component performs validity checks on all the parameters provided and forms initialization control blocks applicable to each Functional Unit.

This component is similar to the components in MVS and JES2/3 that are concerned with initialization. The systems programmer specifies the initialization options in a parameter library allocated in the SUPERLINK JCL with the DDNAME "SLPARM". These options are processed and then held in storage using an internal format.

The Options Processor component has a "backend" that is invoked during SUPERLINK/MVS termination. The "back-end" releases the Initialization Options Table (IOT) appendages obtained during this component's operation.

Options Processor Module Structure

The Options Processor component consists of the following modules:

Module	*Function*
S@C1000	Control module
S@C1010	Initialization; obtain work areas and validate parameter list.
S@C1020	Termination; release work areas and return diagnostic information.
S@C1030	Statement builder
S@C1040	Parameter scan; validate and set control block field(s).
S@C1050	Table of valid parameters, value ranges, types, conversions, and so on
S@C1GETM	Obtain storage for and anchor IOT appendage
S@C1DATA	Read in-stream data records; store in IOT appendage and IOT list.
S@C1FLAG	Set multiple flag values in single byte field
S@C1060	Backend; release all IOT appendages.

Figure 6 on page 3-2 shows the hierarchical structure of the modules within the Options Processor component.

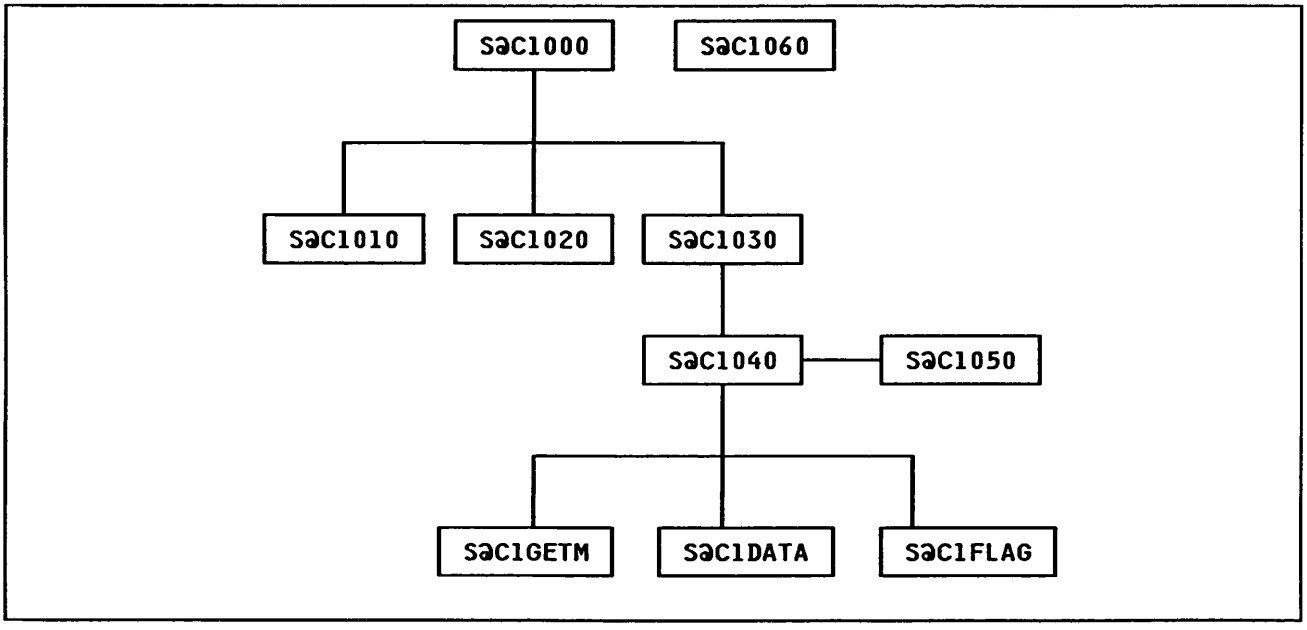


Figure 6. Module Structure of Options Processor Component

Options Processor Services

The Product Management component (S@CC0000) invokes the Options Processor component. S@CC0000 may request one of two functions: it asks the Options Processor component to provide either the complete set of options or the options for a specific Functional Unit. A parameter list is provided; this list gives the parameter requirements as well as the relevant output control blocks (see "Options Processor Data Areas" on page 3-3).

An optional DDNAME of SLLIST may be used to provide printed output of parameters and messages.

Options Processor Interfaces

The Options Processor component is either CALLed or ATTACHed and receives a parameter list on entry. The contents of the parameter list are as follows:

- An indication of the function for which options processing is required. An ALL indication must have been successfully processed before receipt of a specific indication.
- A pointer to the Control Initialization Options Table (SC_CIOT)
- The member name containing the options

  This parameter is mandatory for an ALL indication. For a specific indication, this value replaces the SC_CIOT value and causes the named member to be read. If this parameter is set to binary 0s or is omitted, the member named in the SC_CIOT is reused.
- An area in which to return error information

A combination of return codes and feed-back codes describe an error, as shown in Table 1 on page 3-3.

Table 1. Return Codes and Feed-back Codes for the Options Processor

Return Code	Feed-back Code	Meaning
00	Successful completion	
04	Successful completion; LIST output was requested but could not be produced.	
08	Processing terminated due to syntactical errors. For all feed-back codes, the address of the statement in error is returned. For feed-back codes 04 and 08, the address of the start of the parameter is returned. For feed-back code 12, the current address within the statement may be returned.	
	04	Scan error; an obsolete parameter was found.
	08	Scan error; an unsupported parameter was found.
	12	Scan error; an error was encountered during scan.
12	Processing terminated due to I/O failure. For feed-back code 16, the abend code is also provided; an indication of the attempted I/O operation is included if appropriate.	
	04	Member was not found in PARMLIB.
	08	Invalid data control block (DCB) for PARMLIB
	12	OPEN failed for DDNAME SLPARM (DD statement missing).
	16	An abend was intercepted by ESTAE processing.
16	Processing was not performed.	
	04	The member name was invalid/omitted.
	08	The control block address supplied was invalid.
	12	A "refresh" request for a specific options table was received before the complete options were built.
	16	Bad parameter list received

The backend (S@C1060) of the Options Processor component is called during SUPERLINK/MVS termination to release all IOT appendages. It receives the address of the SC_CIOT as a mandatory parameter.

Options Processor Data Areas

Each SUPERLINK/MVS Functional Unit has an associated data area containing its options. These data areas are allocated by S@CC0000, and their addresses are passed to each Functional Unit. The following is a list of these data areas:

Data Area	*Description*
SC_CIOT	Control Initialization Options Table (anchor for the NIOT)
SC_NIOT	Network Initialization Options Table

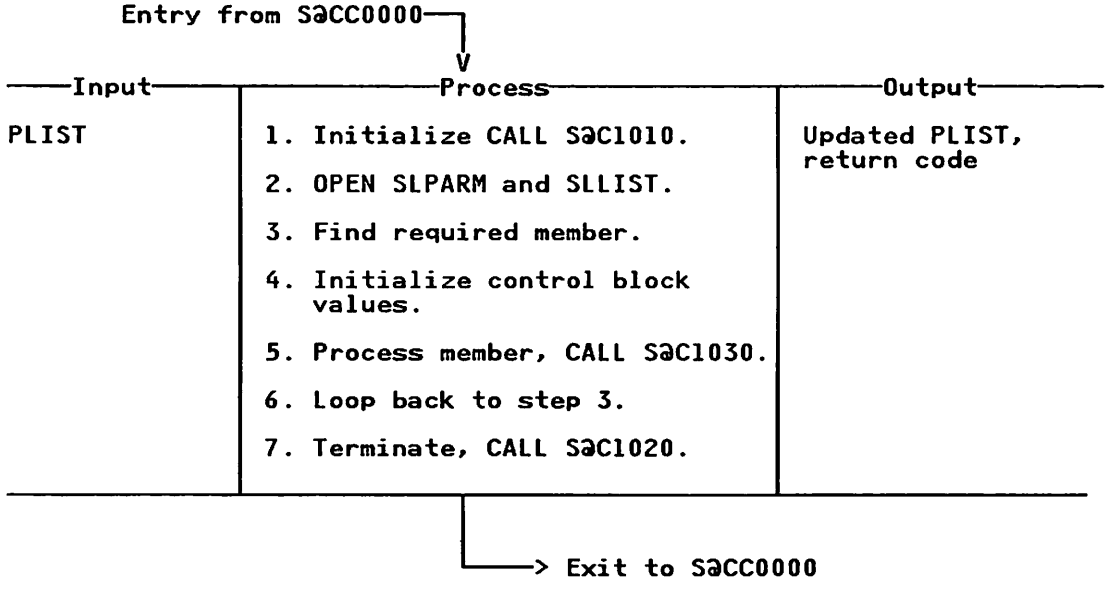
"Appendix A. Data Area Descriptions" on page A-1 provides a description of the previous data areas.

Options Processor Recovery

An ESTAE is provided to intercept abends. Retry processing is not performed; rather, diagnostic information in the form of feed-back codes is returned to the caller.

This page has been intentionally left blank.

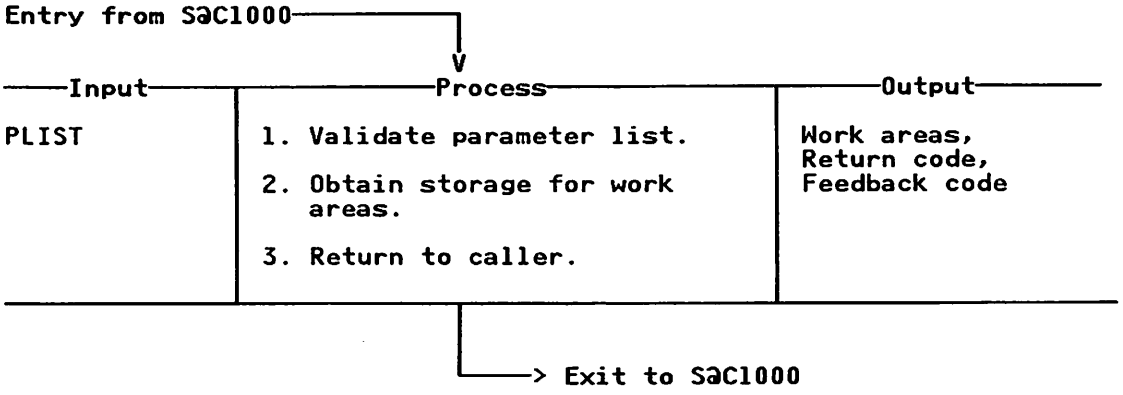
Diagram 3-1
S@C1000 - Control Module



Extended Description

Explanation	*Module*	*Label*
1. Module S@C1010 obtains work areas and validates the parameter list (PLIST). The return code from S@C1010 is checked, and processing is resumed at step 7 if errors occurred.	S@C1000	
2. The parameter library is opened and validated. Any errors result in a branch to step 7. The LIST dataset is validated and opened. Processing continues in "NOLIST" mode if there are any errors. Return code 04 is given on completion.	S@C1000	
3. Search the parameter library's directory for the required member name. If it is not found, branch to step 7.	S@C1000	
4. Module S@C0000 allocates the IOTs and places identifiers in them. The fields in the IOT are set to their starting values. For refresh, S@CC0000 may pass the address of the original IOT or may provide a new IOT. The member read for refresh must contain statements for every OPTION required (not just those to be amended).	S@C1000	
5. Once the required member is found, module S@C1030 is called once to process all records in the member. The return code from S@C1030 is checked, and a branch is made to step 7 if errors occurred.	S@C1000	
6. Loopback occurs only when the PLIST requested that a complete set of options be built. Each set of OPTIONS is contained in a separate member. The member names are contained in a root member for SLCN OPTIONS. Exit from the loop is taken when all named members have been processed.	S@C1000	
7. Module S@C1020 releases the work areas and updates the parameter list to reflect the processing done.	S@C1000	

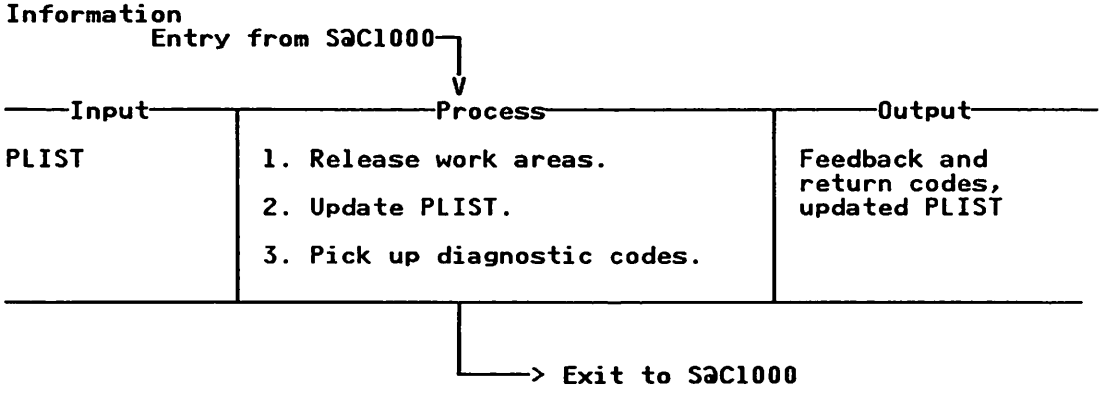
Diagram 3-2
S@C1010 - Initialization; Obtain Work Areas and Validate Parameter List



Extended Description

Explanation	*Module*	*Label*
1. The parameter list supplied by S@CC0000 to S@C1000 is passed to this module. Its contents are validated here. The return and feedback codes notify S@C1000 of errors. For more information about return and feed-back codes, see the SUPERLINK/MVS Logic Library Volume 1: Product and Component Descriptions, CRI publication SI-0181.	S@C1010	
2. The work areas must be used by S@C1030 and S@C1040. Module S@C1040 requires a work area stack of sufficient depth for the most complex statement syntax. For example, the statement "START1 PROC=a,PARAM=(b,c)" requires a depth of three (statement level, parameter level, subparameter level).	S@C1010	
3. Return and feedback codes indicate the processing done.	S@C1010	

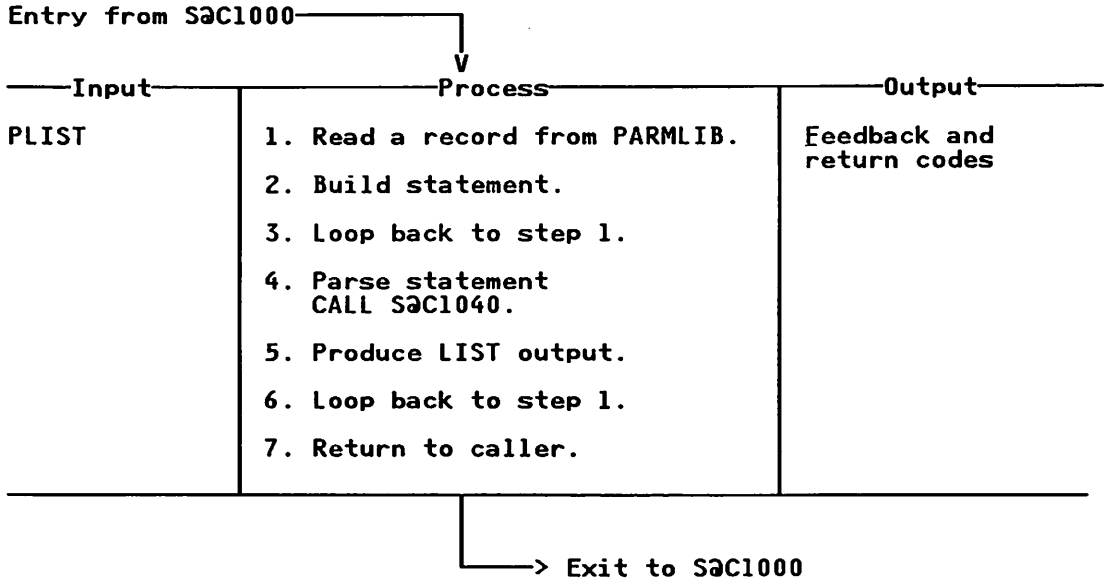
Diagram 3-3
S@C1020 - Termination; Release Work Areas; Return Diagnostic



Extended Description

Explanation	*Module*	*Label*
1. The work areas obtained by S@C1010 are released. However, if there are syntax errors, the offending statement is not freed, since it is required by S@CC0000.	S@C1020	
2. The PLIST-supplied S@CC0000 is updated with additional diagnostic information. Return codes for which there is no feedback code, and return codes for which the PLIST was in error are not updated.	S@C1020	
3. The return code and feedback codes are loaded and returned to the caller.	S@C1020	

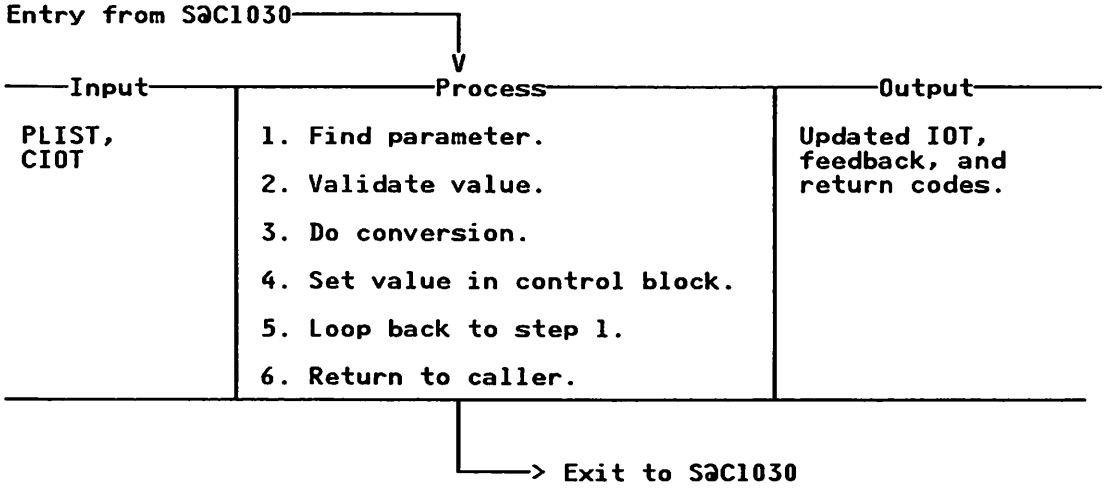
Diagram 3-4
S@C1030 - Statement Builder



Extended Description

Explanation	*Module*	*Label*
1. PARMLIB has been OPENed by S@C1000 and the PDS member found. The parameter list contains information (DCB address and so on) to enable this module to read records.	S@C1030	
2. The statement is formed from one or more records as a variable-length string. Leading and/or trailing blanks are stripped. A statement may contain a comment following the last parameter and may contain a continuation character in column 72.	S@C1030	
3. Loopback to step 1 occurs if the statement is continued onto another record.	S@C1030	
4. This step is bypassed if the statement is a comment (first byte = *). Module S@C1040 returns diagnostic information as a result of its processing. If there are errors, branch to step 7.	S@C1030	
5. The statement and any diagnostic information are written to SL2LIST, if present.	S@C1030	
6. Exit from the loop is taken at end-of-file. The dataset is not CLOSEd here since further member(s) may be required.	S@C1030	
7. Diagnostic information (feedback and return codes) is provided for the caller.	S@C1030	

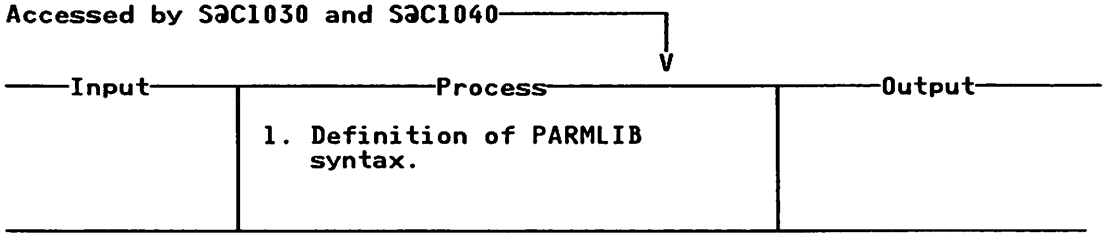
Diagram 3-5
S@C1040 - Parameter Scan, Validate and Set Control Block Field(s)



Extended Description

Explanation	*Module*	*Label*
1. The PLIST contains the statement to be parsed, the address of the table of valid parameters, and the address of the current work area in the stack. Work areas are chained, since this module is recursive for scanning subparameters. The top of the work area (parent) is pointed to on initial entry from module S@C1030.	S@C1040	
2. The value of the parameter is validated against the table in module S@C1050. Errors are indicated by a combination of return and feedback codes.	S@C1040	
3. The parameter is converted from its external representation (character, hex digits, and so on) to its internal representation (binary and so on), as indicated by module S@C1050.	S@C1040	
4. The value is stored in the CIOT, NIOT, or another table, as appropriate for the OPTIONS being built. S@C1050 holds addressing information, which enables the IOT field to be located for updating.	S@C1040	
5. Loopback occurs if there are more parameters or if there are subparameters.	S@C1040	
6. The caller is notified of the processing performed by return and feedback code settings.	S@C1040	

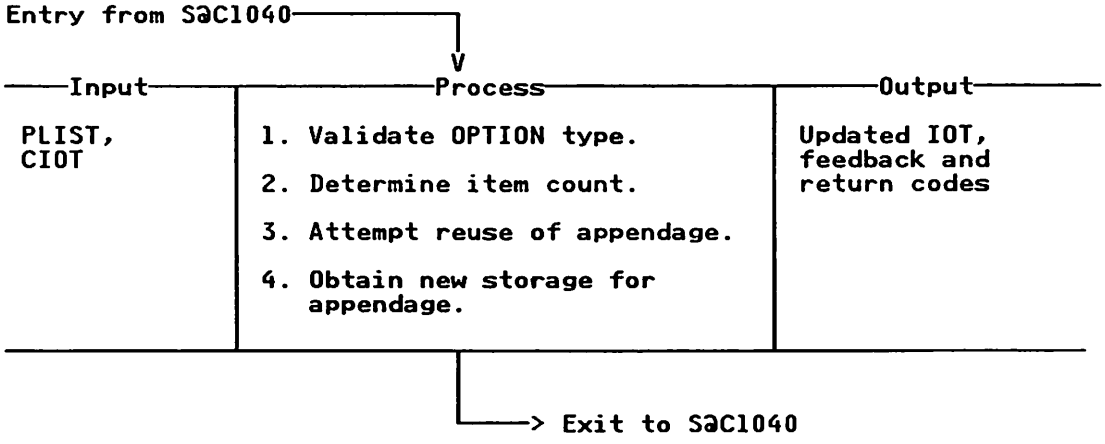
Diagram 3-6
S@C1050 - Table of Valid Parameters, Value Range, Type, Conversion



Extended Description

Explanation	*Module*	*Label*
1. This look-up table is a nonexecutable module. There are a number of entries that define the syntax, validation, conversion, and IOT control block fields for parameter library statements. Module S@C1030 passes a pointer to this module to S@C1040. Details are given on the S@C1SYNX macro (see "Appendix B. SLCN Macros" on page B-1).	S@C1050	
An entry for each statement is followed by a list of entries for each parameter. The parameter entry may be followed by a list of entries for each subparameter. Parameters may be keyword or positional and may be subscripted. Parameters that have nonstandard syntax are handled by specifying an internal exit. (For example, PARAM(a,b) on the START statement is required as a single variable-length string a,b).		
You can mark parameters that are superseded from one software version to the next as "obsolete." This fact is reported on the LIST dataset; S@CC0000 is also informed.		

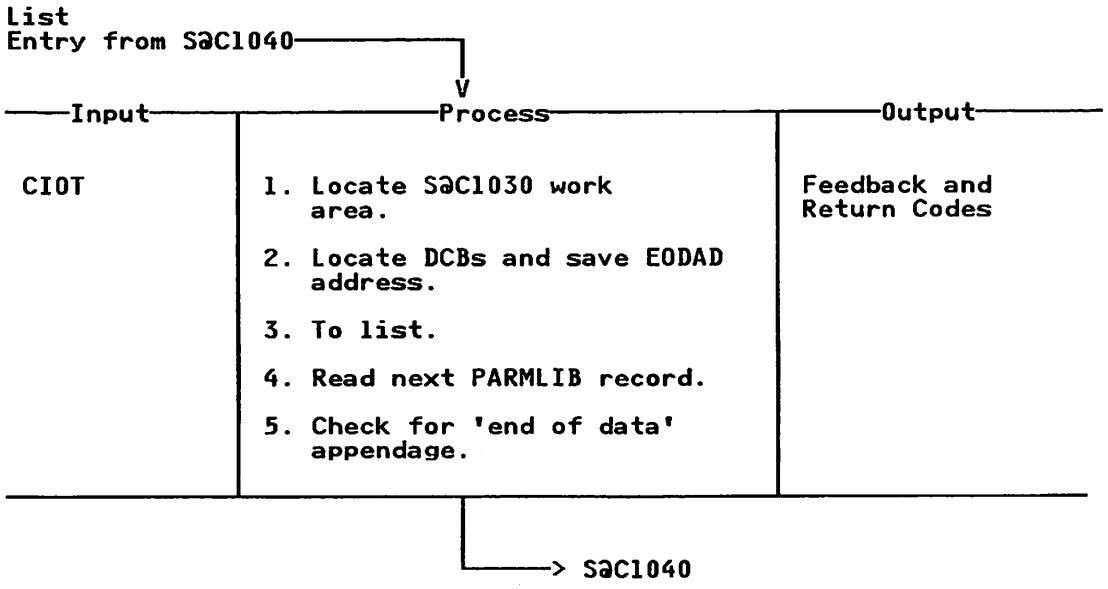
Diagram 3-7
S@C1GETM - Obtain Storage for and Anchor IOT Appendage



Extended Description

Explanation	*Module*	*Label*
1. The parameter list consists of two fullwords. The first is a pointer to the current work area in the stack. The second is a pointer to the following list of fullwords in S@C1050 (specified via AFT=(S@C1GETM,label of params). • Offset to anchor field in IOT • Length of an item (< 2**16) The current work area contains a pointer to the syntax table entry for the item being processed. The item must be defined as being numeric (CONV=NUM on S@C1SYNX) or an error message and return code result.	S@C1GETM	
2. The value of the parameter is validated against the table in module S@C1050. Errors are indicated by a combination of return and feedback codes. For more information about return and feedback codes, see CRI publication SI-0181, SUPERLINK/MVS Logic Library Volume 1: Product and Component Descriptions.	S@C1040	
3. This internal SUPERLINK/MVS Options exit is taken after the item has been validated and the CIOT updated. The syntax table definition for the item contains the offset to the item within the IOT. Note that the item may be 1,2 or 4 bytes long.	S@C1GETM	
4. The anchor field is checked to see if an appendage already exits. If not, then this step is bypassed. The count of items in the existing appendage is compared with the current item count. If the current item count is higher, then the existing appendage is released. Otherwise, the existing appendage header fields are reset and the next step is bypassed.	S@C1GETM	
5. The length of storage required for the appendage is calculated as follows: appendage header length + (item count * item length). The storage is obtained, the appendage header fields updated, and the appendage address stored in the IOT anchor.	S@C1GETM	

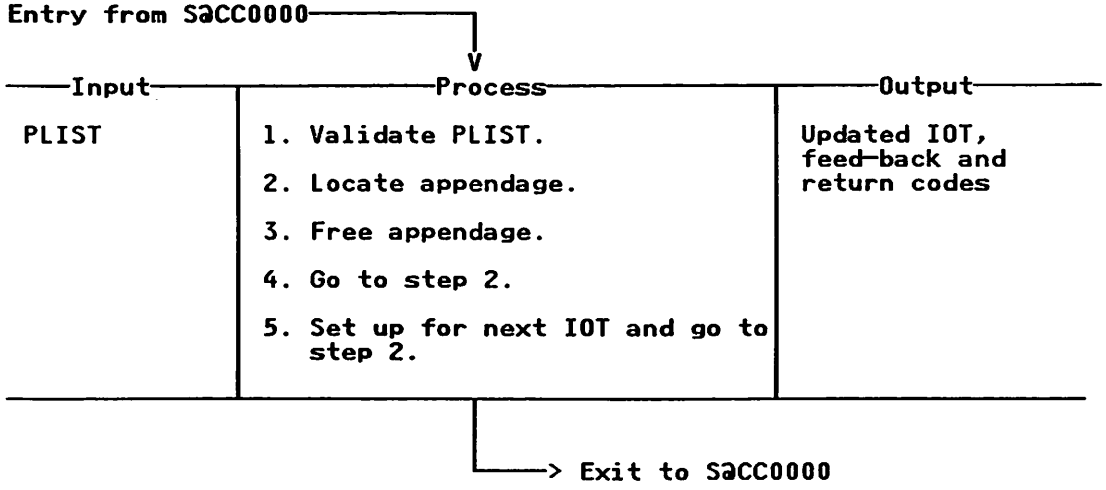
Diagram 3-8
S@C1DATA - Read Instream Data Records, Store in IOT/Appendage and



Extended Description

Explanation	*Module*	*Label*
1. The parameter list consists of two fullwords. The first is a pointer to the current work area in the stack. The second is a pointer to the following list of fullwords in S@C1050 (specified via PRE=(S@C1DATA,label of params). • Offset to anchor field in IOT • Pointer to syntax table entry for the item. This internal exit is called by S@C1040 before the current item on the Options statement has been processed. It is not known how many times S@C1040 has been invoked. However, the work area stack is contiguous with each element of a fixed size with a depth indicator. The S@C1030 work area is at the top of the stack.	S@C1DATA	
2. The DCB addresses are passed to S@C1030 in its parameter list from S@C1000. The address of this parameter list is held in S@C1030's work area. This module must deal with "end of file" during its execution but must restore the previous EODAD address on return.	S@C1DATA	
3. The statement/data record which is in the buffer is listed. The first time through the BEGIN statement that caused this internal exit to be invoke d is listed. On subsequent passes, the previously read "data record" is listed.	S@C1DATA	
4. The S@C1030 work area contains positioning information for reading PARMLIB records. This information is used to advance within a block or read a new block.	S@C1DATA	
5. "End of Data" is reached when the record read is an END statement. The name on the END statement is matched with the syntax table entry name for the BEGIN statement. An error message is returned if the match fails; otherwise, an exit from the list/read loop is taken. Reaching end of file results in the error message "end of file before end of data". The END statement is left in the buffer to be listed by S@C1030.	S@C1DATA	
6. The appendage header is used to check if there is room for the data record. If not, an error message is returned. The offset within the appendage for the star of the data record is computed using the appendage header fields.	S@C1DATA	

Diagram 3-9
S@C1060 - Back-end, Release All IOT Appendages



Extended Description

Explanation	*Module*	*Label*
1. The parameter list consists of a single fullword which contains the address of the CIOT. This parameter is mandatory; it is the starting point for appendage release. If the CIOT address is missing or invalid, return code 16 results.	S@C1060	
2. If the anchor field within the current IOT is non-zero, it contains the address of the appendage user data. If the anchor field is 0, step 3 is bypassed. The address of the appendage header is computed by decrementing the appendage user data address.	S@C1060	
3. The appendage header contains the length of storage occupied by the appendage.	S@C1060	
4. Exit from the loop is taken when all appendages in the current IOT have been released.	S@C1060	
5. The CIOT contains pointers to all the other IOTs. These are processed in the following sequence (providing the pointer is nonzero): NIOT, OIOT, VIOT, JIOT.	S@C1060	

4. SUPERLINK Functional Subsystem Manager Component

The FSS Manager component of SLCN is responsible for initialization, termination, and recovery of SUPERLINK Functional Units, as well as activation of SUPERLINK FSSs such as SLNET. FSSs can either be started automatically, under the control of SUPERLINK/MVS, or activated by an MCS operator command.

Functional Subsystem Subcomponents

The FSS Manager component consists of the following subcomponents:

Subcomponent	*Description*
S@CF0000	FSS Manager control subcomponent
S@CF0100	FSS Manager cross-memory communications subcomponent

S@CF0000 - FSS Manager Control Subcomponent

The FSS Manager control subcomponent is responsible for initiation, termination, and recovery of SUPERLINK/MVS FSSs.

S@CF0000 - Module Structure

The FSS Manager control subcomponent consists of the following modules:

Module	*Function*
S@CF0000	FSS Manager root module
S@CF0010	Cross-memory environment management initialization
S@CF0020	Cross-memory environment management termination
S@CF0030	MVS START command creation and issuance
S@CF0040	FSS Manager ESTAE processing

Figure 7 on page 4-2 shows the hierarchical structure of modules within the FSS Manager control subcomponent.

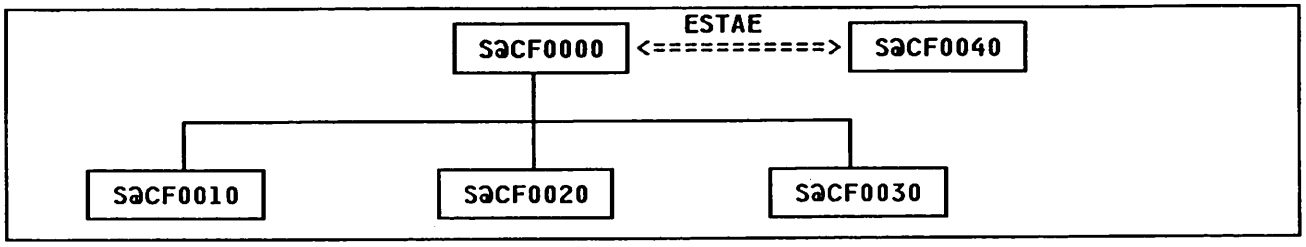


Figure 7. Module Structure of the FSS Manager Control Subcomponent

S@CF0000 - Services

The component managers resident within the FSS Manager control subcomponent modules are not controlled by the FSS Manager component; they are directly controlled from within the associated Functional Units. The FSS Manager control subcomponent controls the initialization and termination of subordinate Functional Units through the FSS ORDER/SEND mechanism.

S@CF0000 - Interfaces

The Functional Subsystem Interface (FSI), as defined by MVS/XA release 2.1.2 and later, is the standard interface adopted by the FSS Manager component.

Other SUPERLINK components may make requests of the FSS Manager component by building an FSS Request Element (FRQE) and adding it to the work queue for the FSS Manager component, which is anchored in the SUPERLINK SC_SSVT control block.

The FSS Manager component provides MVS-required support for configuration of a started task as a functional subsystem. It also enables the FSI to provide the multi-address space communication mechanism.

SLCN handles the initialization and termination of subordinate Functional Units with the FSS ORDER/SEND mechanism. The SUPERLINK/MVS ORDERS include the following:

ORDER	*Description*
INIT	Order issued to SLNET requesting allocation of link devices
MSG	Order requiring the specified Functional Unit to receive a message from SLCN
STATUS	Order requesting current status information for the specified Functional Unit
TERM	Order requesting an FSS to terminate its function. A subparameter of the TERM ORDER indicates whether the termination sequence is one of the following: • Component shutdown; requests termination of a specific component or all components resident within the specified Functional Unit. • Abort shutdown; requires termination regardless of session or data loss. • Quick shutdown; requests normal termination of all active sessions with no abort processing.

An INIT ORDER for FSS initialization is unnecessary. The INIT ORDER is implicit in the MVS START command issued by the controlling subsystem during the initialization sequence for the FSSs. A Functional Unit issues a subsystem request (FSS CONNECT/DISCONNECT) during initialization; this request allows the subsystem to acknowledge the Functional Unit as an FSS.

Note: RESTART is not a SUPERLINK/MVS ORDER. FSS RESTART is performed by a series of TERM ORDERs followed by MVS START commands. The MVS START commands perform the FSS reinitialization sequence.

S@CF0000 - Data Areas

Several data areas are required to support an FSS. These data areas contain a fixed header followed by component-specific fields. Figure 8 and Figure 9 illustrate the interaction between an FSS, the Subsystem Interface, and relevant MVS control blocks.

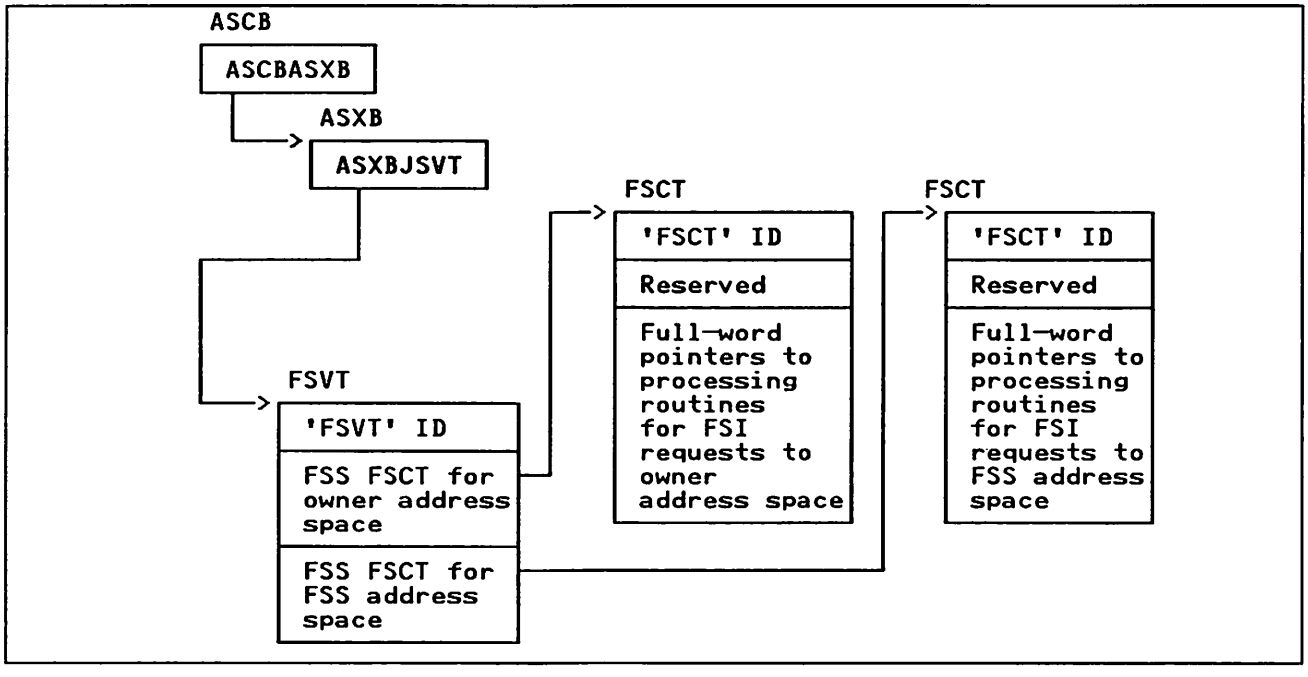


Figure 8. MVS-level Control Block Structure

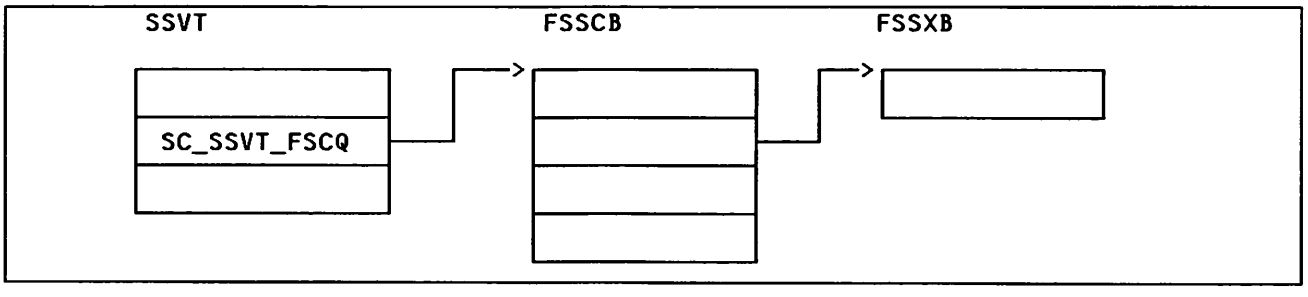


Figure 9. Subsystem Interface Level Control Block Structure

An SC_FSSCB exists for each FSS address space required and is resident in CSA. The SC_SSVT_FSCQ is an anchor point for the chain of SC_FSSCBs belonging to this subsystem. The FSSXB is an SC_FSSCB extension control block, resident in the FSS private area.

The FSS Manager component also makes use of the FSS Request Element (SC_FRQE), which describes a request for the performance of some function. It is built by the requester and chained off the SC_SSVT. "Appendix A. Data Area Descriptions" on page A-1 provides a description of these data areas.

S@CF0000 - Recovery

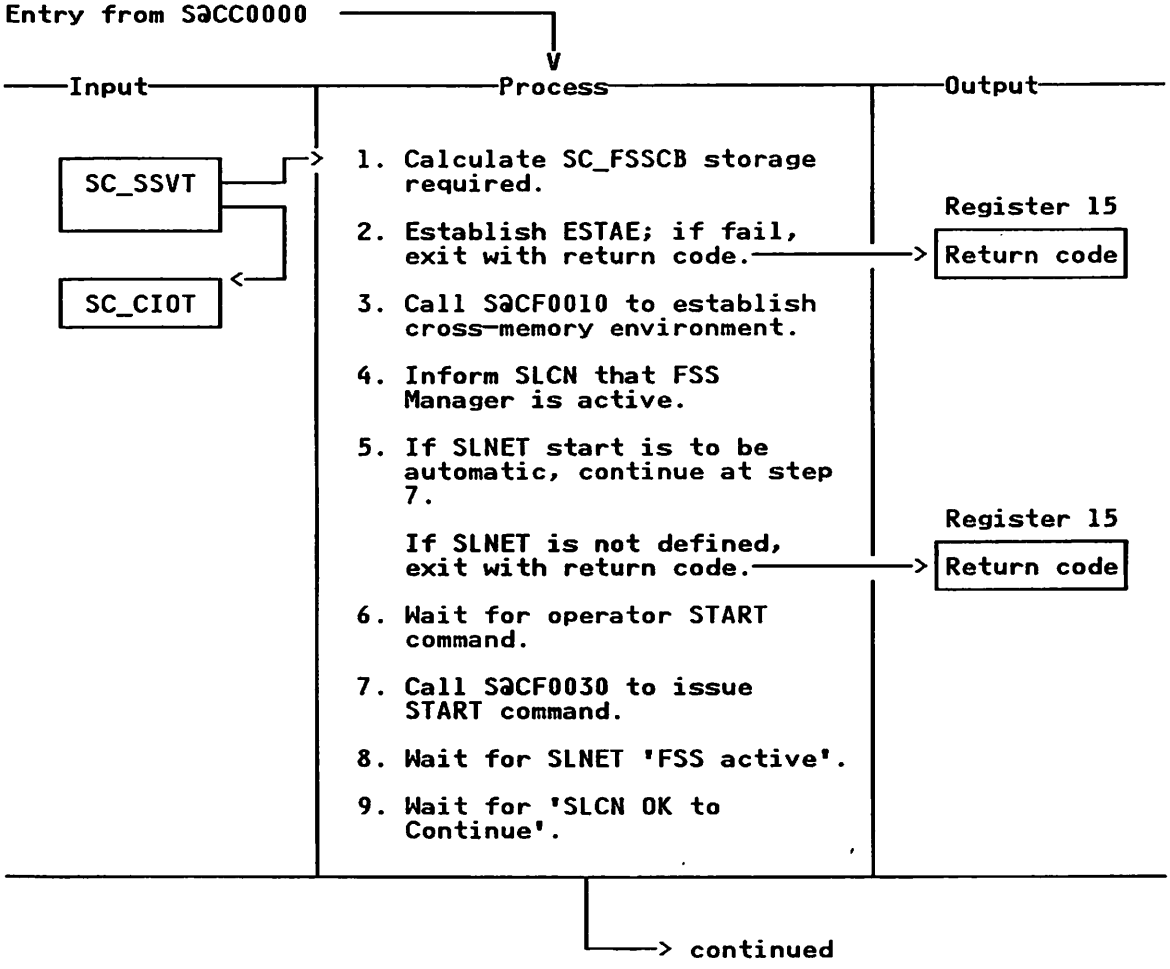
The FSS Manager runs as a subtask of the SLCN job step task. An ESTAE environment is established to intercept abends of this component.

When invoked by an abend, the ESTAE routine logs the error, gathers diagnostic information, and determines whether FSS Manager component recovery is possible. If recovery is possible, the SETRP macro is used to establish a retry routine. The ESTAE routine returns to the MVS Recovery Termination Manager, which causes resumption of the FSS Manager component at the retry routine.

If recovery is not possible, the environment is cleaned up, and the abend is percolated to the parent task (the SLCN job step task). The ESTAE detects the abend of the daughter task and determines whether or not to reestablish the FSS Manager component.

This page has been intentionally left blank.

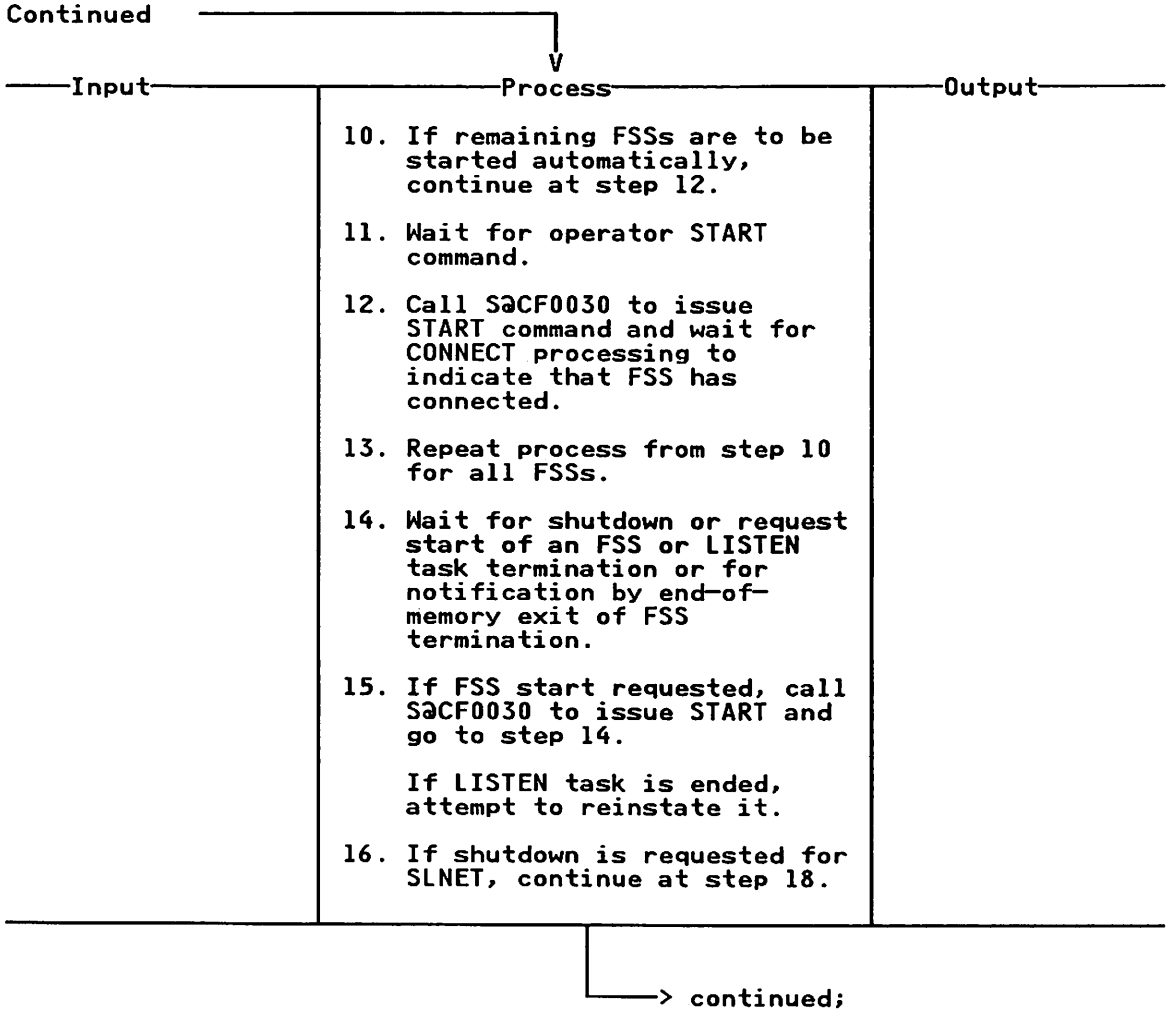
Diagram 4-1
S@CF0000 - FSS Manager Root Module (part 1 of 3)



Extended Description

Explanation	*Module*	*Label*
1. SC_FSSCBs are allocated in CSA in contiguous storage. The number of FSS address spaces supported by SUPERLINK/MVS is defined in the Initialization Options Table (SC_CIOT). An SC_FSSCB is located by its FSS-ID, which acts as an index into the SC_FSSCB storage area.	S@CF0000	
4. SLCN initialization must be informed that the FSS Manager component is active so that initialization processing can continue.	S@CF0000	
5. The FSS definition in SC_CIOT includes a parameter stating whether FSS initialization is to be automatically activated by SLCN, or is to be requested by an MVS MCS console operator command.	S@CF0000	
6. A subsystem START command is necessary before SLNET can be activated. All other START commands are invalid. The subsystem START command has the following format: #START FU=*nnnnnn,....* where # denotes the subsystem command character defined in the SC_CIOT.	S@CF0000	
8. CONNECT processing of the FSS POSTs this WAIT to indicate that SLNET is active.	S@CF0000	
9. Subsequent FSS address spaces cannot be activated until SLCN has completed its initialization sequence, which includes establishing the SUPERLINK/MVS Management Interface session with SLCN resident on COS.	S@CF0000	

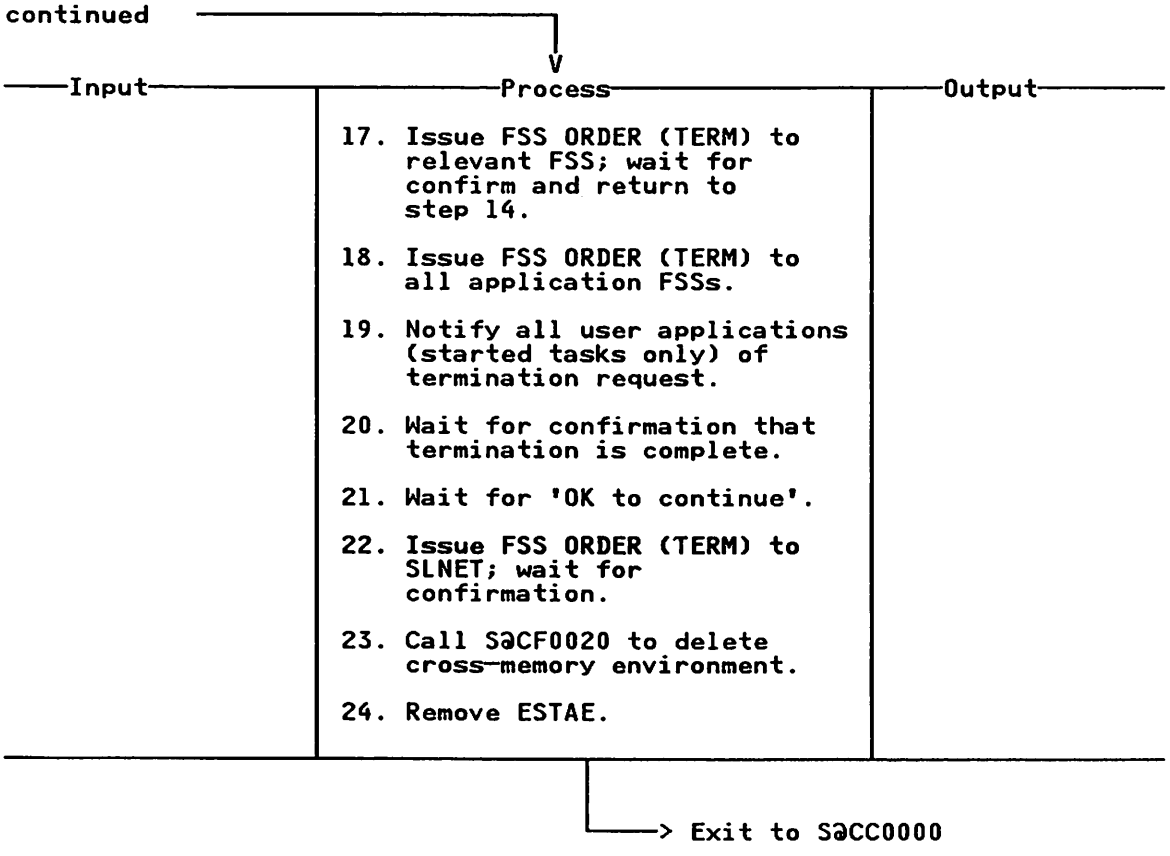
Diagram 4-2
S@CF0000 - FSS Manager Root Module (part 2 of 3)



Extended Description

Explanation	*Module*	*Label*
10. FSS address spaces can either be started automatically under the control of SUPERLINK/MVS or activated by MCS operator command.	S@CF0000	
14. Once all the FSS address spaces have been activated, the FSS Manager is required to act only under the following conditions: • Request for shutdown of a specific FSS • Request for shutdown of all FSSs • Request to start an FSS • Notification of FSS address space termination	S@CF0000	
15. If an FSS start is requested, call S@CF0030 to issue the MVS START command. If the cross-memory environment has terminated prematurely, attempt to reinstate it. If the reinstatement fails, POST the FSS Manager terminate ECB to terminate cleanly. In either case, return via step 14.	S@CF0000	
16. If an operator requests a shutdown of SLNET, all application FSSs must be closed down first.	S@CF0000	

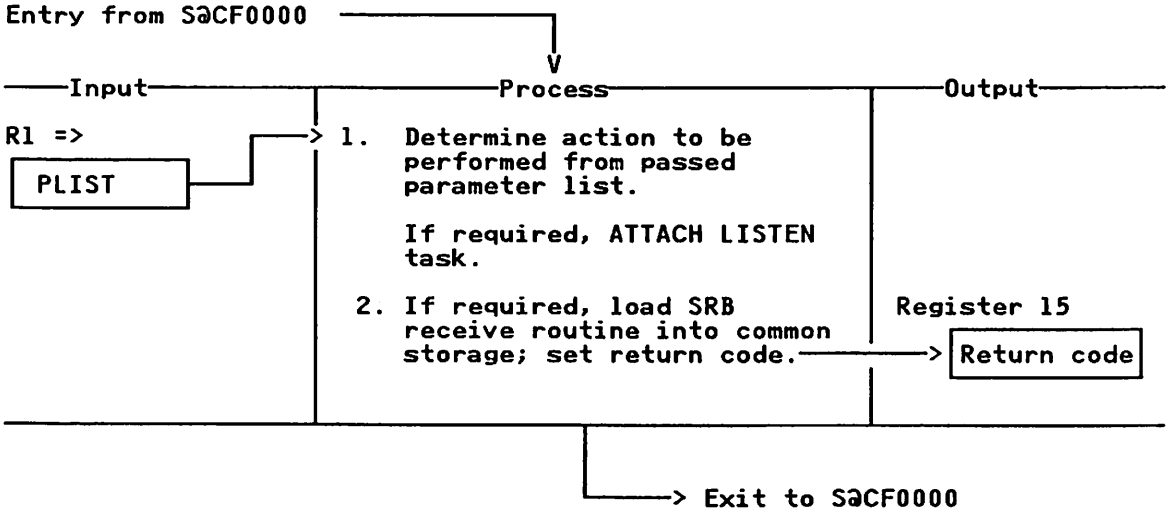
Diagram 4-3
S@CF0000 - FSS Manager Root Module (part 3 of 3)



Extended Description

Explanation	*Module*	*Label*
17. If a specific FSS shutdown has been requested, the FSS Manager issues an FSS ORDER to indicate that termination is required. This can be achieved by use of the S@@CSERV macro. A termination parameter determines whether the shutdown is to be immediate or sedate. *Note: Immediate shutdown is likely to cause loss of data on active sessions.* The relevant FSS response must confirm that FSS DISCONNECT is complete before the shutdown request can be considered as activated.	S@CF0000	
18. The processing for shutdown of the entire SUPERLINK/MVS product is as previously described, except that, initially, only the application FSS address spaces can be terminated. SLNET must remain active until the SUPERLINK Management Interface has been disabled.	S@CF0000	
19. User-written applications can be supported as started tasks. As SUPERLINK/MVS can initialize these applications, it must inform them when termination processing is required.	S@CF0000	
21. SLCN notifies the FSS Manager "OK to continue" once the Management Interface has been disabled.	S@CF0000	

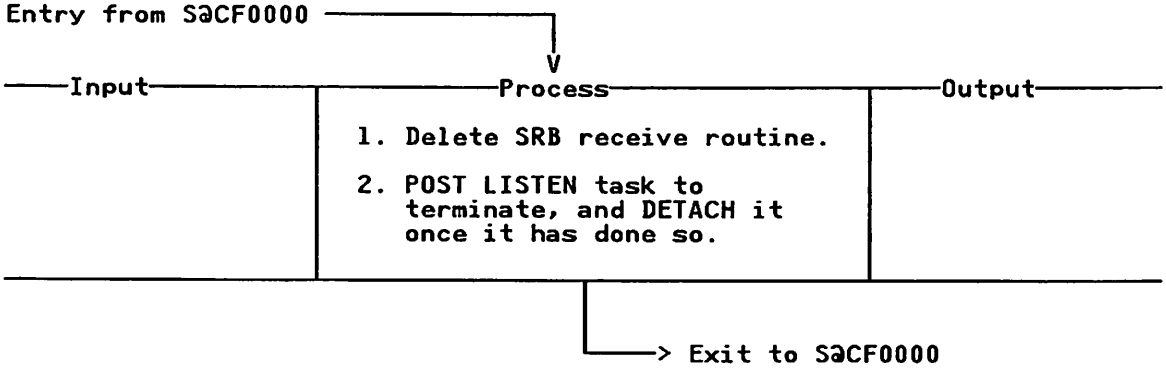
Diagram 4-4
S@CF00100 - Cross-Memory Environment Management Initialization



Extended Description

Explanation	*Module*	*Label*
1. ATTACH the "listen" task as a subtask of the FSS Manager (it must be up before any SRBs can be SCHEDULEd to the FSS address spaces).	S@CF0010	
2. A cross-memory receive routine, which is SCHEDULEd as an SRB in the target FSS address space, is required. This routine enables FSS ORDERs to be passed to the FSS Interface routines. These, in turn, invoke the routines that process the ORDERs.	S@CF0010	

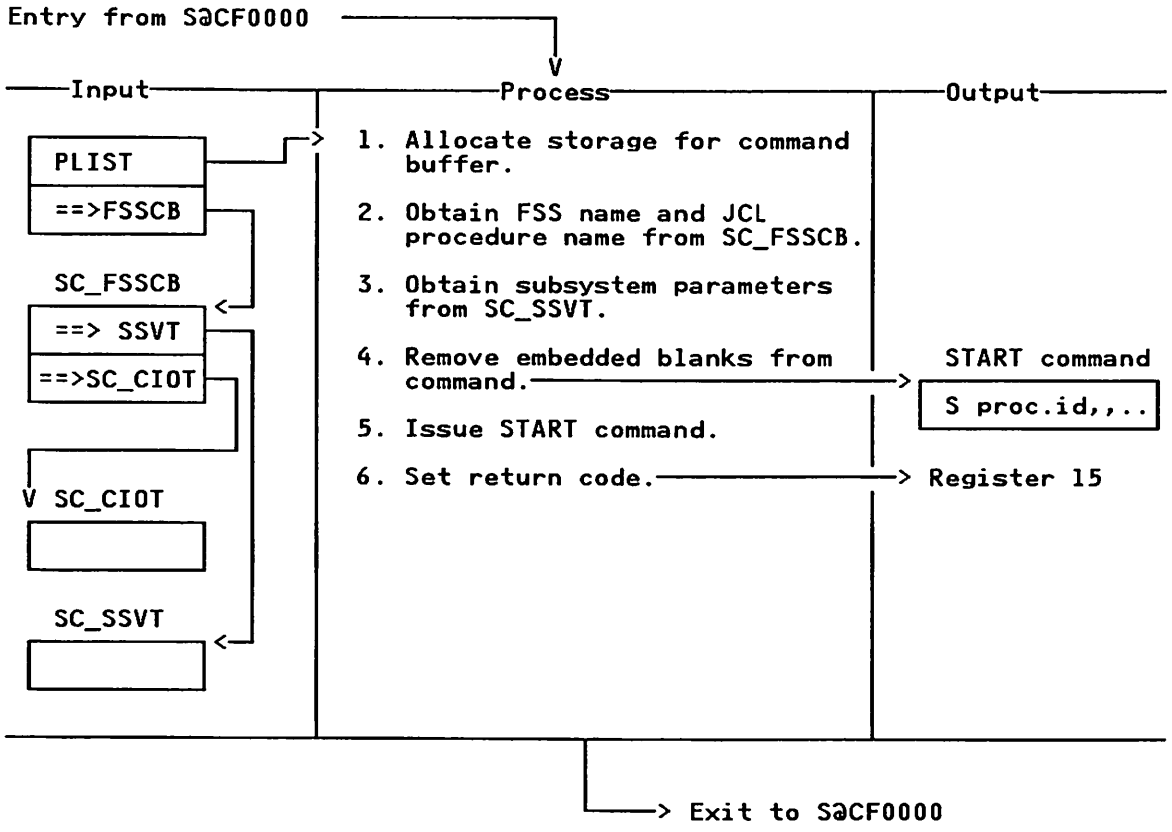
Diagram 4-5
S@CF0020 - Cross-Memory Environment Management - Termination



Extended Description

Explanation	*Module*	*Label*
1. FSS termination has completed; therefore, the SRB receive routine used to move data cross-memory from the SLCN address space is no longer required and is DELETEd.	S@CF0020	
2. Stop the "LISTEN" task by POSTing its "terminate" ECB. WAIT for it to terminate, then DETACH the task.	S@CF0020	

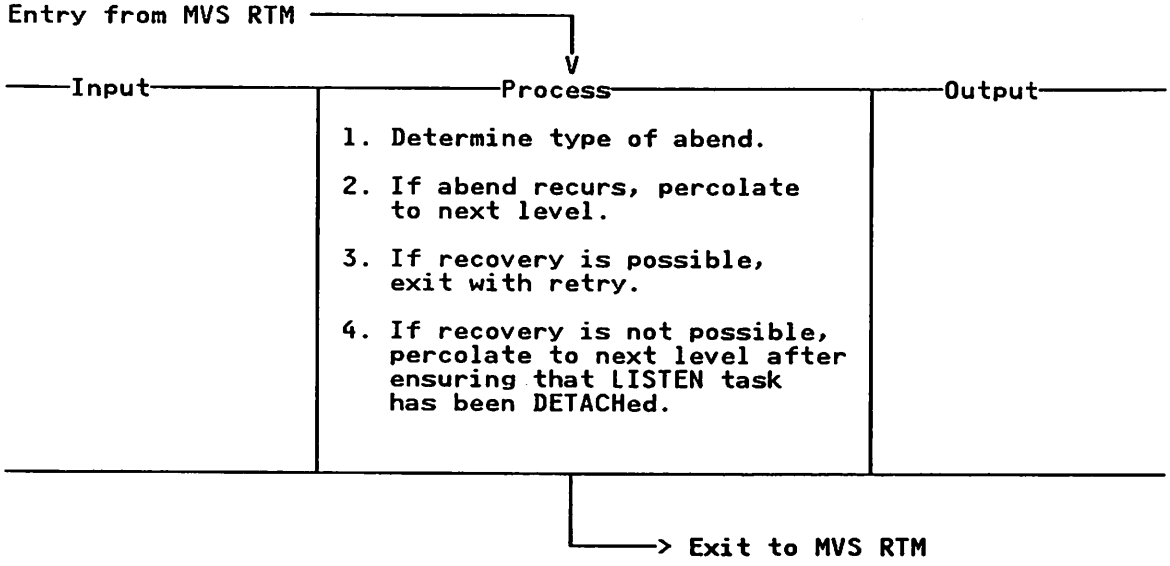
Diagram 4-6
S@CF0030 - MVS START Command Creation and Issuance



Extended Description

Explanation	*Module*	*Label*
1. The command buffer must be allocated in 24-bit addressable storage for the MGCR macro (it must be GETMAINed BELOW the line).	S@CF0030	
2. The JCL procedure name for the FSS is installation-defined. The SC_FSSCB for the FSS in question points to the FSS initialization parameters in which the JCL procedure name is found.	S@CF0030	
3. These START parameters must be included: • The FSS name of the FSS being initiated • A specification of whether the initialization parameters for this FSS are to be refreshed or revised from a specified parameter library member	S@CF0030	
4. The START command is created using fixed-length fields in the command buffer. These fields are embedded with blanks. Blanks must be removed so that the MVS START command processor does not ignore any important parameters. The general syntax of the START command is as follows: S *proc.id,,,(subsys-name,fss-id)*	S@CF0030	
5. The START command is issued with the MGCR macro.	S@CF0030	
6. The MGCR SVC indicates whether or not the START command was processed successfully. A return code of 8 indicates that the START command failed; a return code of 0 indicates that the START command was processed successfully, and register 15 contains the right-justified ASID of the started address space.	S@CF0030	

Diagram 4-7
S@CF0040 - FSS Manager ESTAE



Extended Description

Explanation	*Module*	*Label*
2. To prevent reentry in an abend-type loop, check for recursion. If this is a recursive abend, use the SDUMP routine to obtain a dump for diagnostic purposes. Percolate the abend condition to the next level, where the main-line code S@CC0000 detects the abend of the daughter task.	S@CF0040	
3. If recovery is possible, a retry routine can be specified on the SETRP macro. The routine gains control after this exit has returned control to RTM.	S@CF0040	
4. If recovery is not possible, use the SDUMP routine to obtain a dump for diagnostic purposes. Use the SETRP macro again to percolate the abend condition to the next highest level, where the main-line code S@CC0000 detects the abend of the daughter task. DETACH the "LISTEN" task.	S@CF0040	

S@CF0100 - Cross-Memory Communications Subcomponent

The cross-memory communications subcomponent of the FSS Manager component is primarily concerned with the control of communications between SLCN and the FSSs.

The FSS scheme provides the structure in which one or more address spaces run as subordinate address spaces to a controlling address space. This subsection describes how data is sent to a specified address space and how associated replies are received.

S@CF0100 - Module Structure

The FSS cross-memory communications subcomponent consists of the following modules:

Module	*Description*
S@CF0100	SCHEDULE the SRB receive routine in the target FSS
S@CF0110	The SRB receive routine
S@CF0120	"listen" task in the control address space
S@CF0130	Functional Recovery Routine (FRR) for the SRB receive routine, S@CF0110
S@CF0140	ESTAE routine for the "listen" task, S@CF0120

S@CF0100 - Services

The cross-memory communications subcomponent is used for sending commands from SLCN to the FSSs and returning associated replies to SLCN. This mechanism may also be used to notify SLCN about asynchronous events occurring in the FSSs.

The following FSS communications subcomponent services are used:

- ORDER enables data, typically commands, to be transferred from SLCN to an FSS.
- SEND enables an FSS to transfer data asynchronously to SLCN; typically this data consists of replies to commands, but it may also be notification of exceptional events in the FSS.

The underlying mechanisms for transferring ORDERs and SENDs between SLCN and the FSI and between the FSI and the FSS-specific routines differ as follows:

- ORDERs from SLCN to the FSI routines are moved between address spaces by using a service request block (SRB) routine scheduled in the FSS.
- The SRB receive routine retrieves an ORDER and transfers it to the FSS-specific routines through the FSI routines set up during FSS initialization.
- Like ORDERs from the FSI, SENDs from the FSS are passed to the FSI via the FSI routines.
- The SENDs passed from the FSS to the FSI are passed to SLCN via the cross-memory POST of a "listen" task in SLCN.

S@CF0100 - Interfaces

There are two major interfaces involved in the communication between SLCN and the target FSS:

- SLCN to the FSI routines that are part of the FSS
- FSI routines to the FSS

Figure 10 illustrates these communication interfaces and the mechanisms used to transfer data across them.

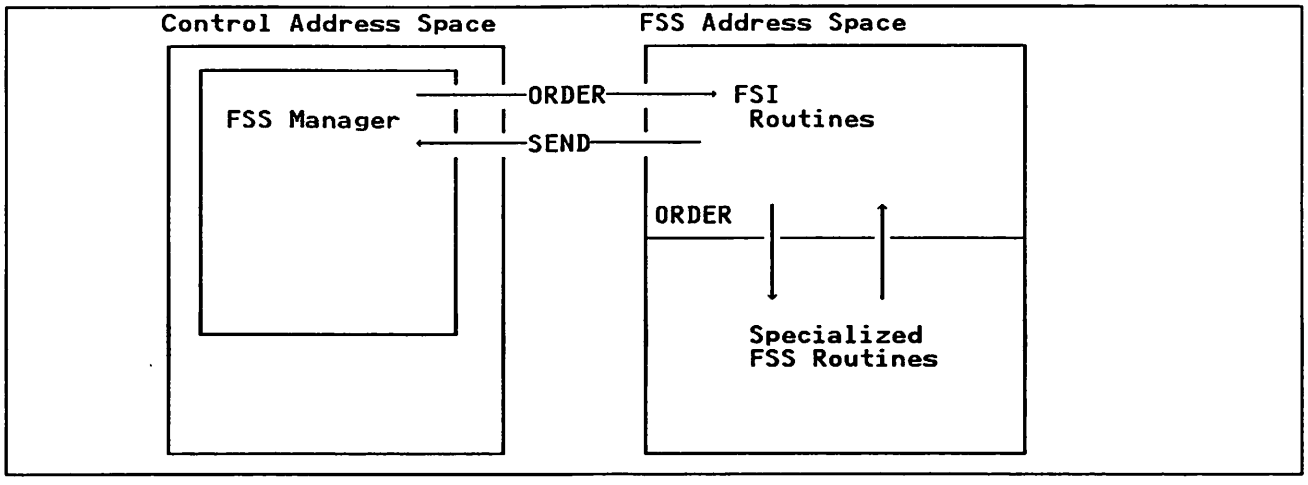


Figure 10. FSS Communication Channels

The control address space is notified of SENDs via cross-memory POST. ORDERs and SENDs are transferred via the FSI mechanism.

The following macros provide the interfaces:

- S@@CSERV

  This macro handles communications between SLCN and the FSI routines in SLCN. It SCHEDULEs the SRB to the FSS address space upon receipt of an ORDER and cross-memory POSTs the control address space when a SEND is requested from SLCN.

  A return code indicating the success or failure of the request is passed in register 15.

- S@@FIREQ

  This macro handles the communications between the FSI and the FSS-specific routines. It also allows the FSS to CONNECT and DISCONNECT itself from the controlling address space through Subsystem Interface function request 53.

"Appendix B. SLCN Macros" on page B-1 describes the syntax for these macros.

S@CF0100 - Data Areas

The following data areas correspond to the two main communication interfaces between SLCN and the target FSS and to the mechanisms used to carry ORDER and SEND requests:

Data Area	*Description*
SC_FSIP	FSI parameter list This data area describes requests passed over the FSI/FSS interface. Extensions to this structure are used to map different types of FSI requests. The extensions are used for CONNECT/DISCONNECT and SEND/ORDER processing; there are IBM mappings for CONNECT/DISCONNECT requests and for the fixed headers for ORDER and SEND requests contained in the IAZFSIP DSECT. Extensions to the ORDER and SEND mappings are provided to accommodate special SUPERLINK/MVS requirements within the context of requests passed via the ORDER and SEND primitives.
SC_SERV	The parameter list associated with the S@@CSERV macro This data area is used to pass requests between the requester of a service and the routine (S@CF0100) that SCHEDULEs the SRB in the target address space. SC_SERV is completed by the S@@CSERV macro.
FM_STAG	Staging area buffer This data area is the staging area buffer used to transport requests between two address spaces. Control information about the SRB used to convey the request is also maintained here.
FM_FSWQE	FSS work request element This data area is the cross-memory work request element for the FSS. It is queued from the SC_FSSCB when an ORDER request is sent to the FSS. The ORDER request, control information, and chaining pointers are embedded in the FM_FSWQE. The cross-memory SRB routine allocates FM_FSWQE in the FSS-private area.

"Appendix A. Data Area Descriptions" on page A-1 provides a format description of these data areas.

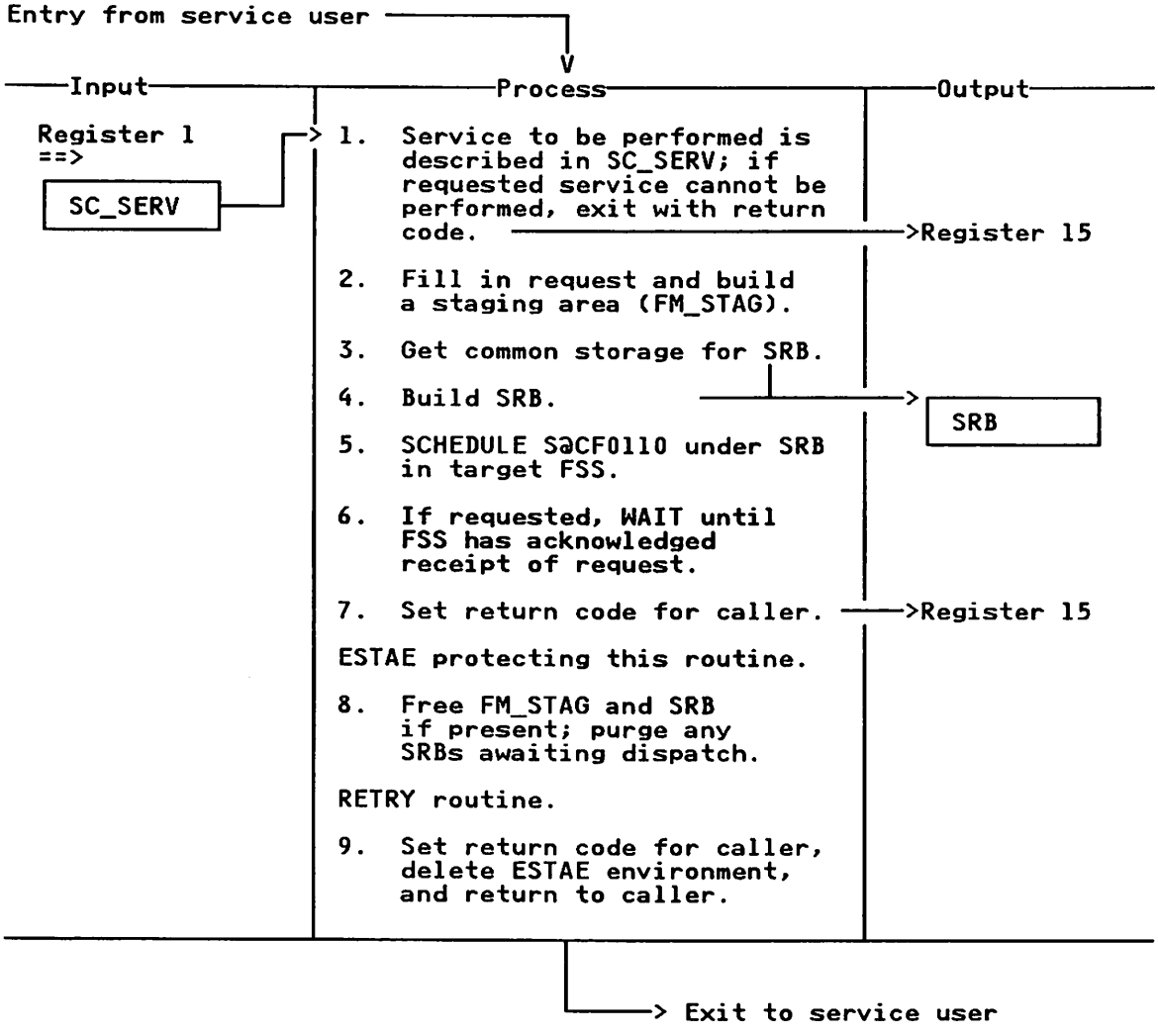
S@CF0100 - Recovery

The SRB "receive" routine (S@CF0110), which executes under an SRB in the target FSS, is covered by a Functional Recovery Routine (FRR). This routine traps abends in the FSI routines that are performing the receive processing of ORDERs.

The "listen" task, which runs in SLCN and waits to be POSTed by one of the FSSs upon SEND processing, is covered by an ESTAE. The ESTAE traps abends of this task and determines if successful recovery processing is possible.

This page has been intentionally left blank.

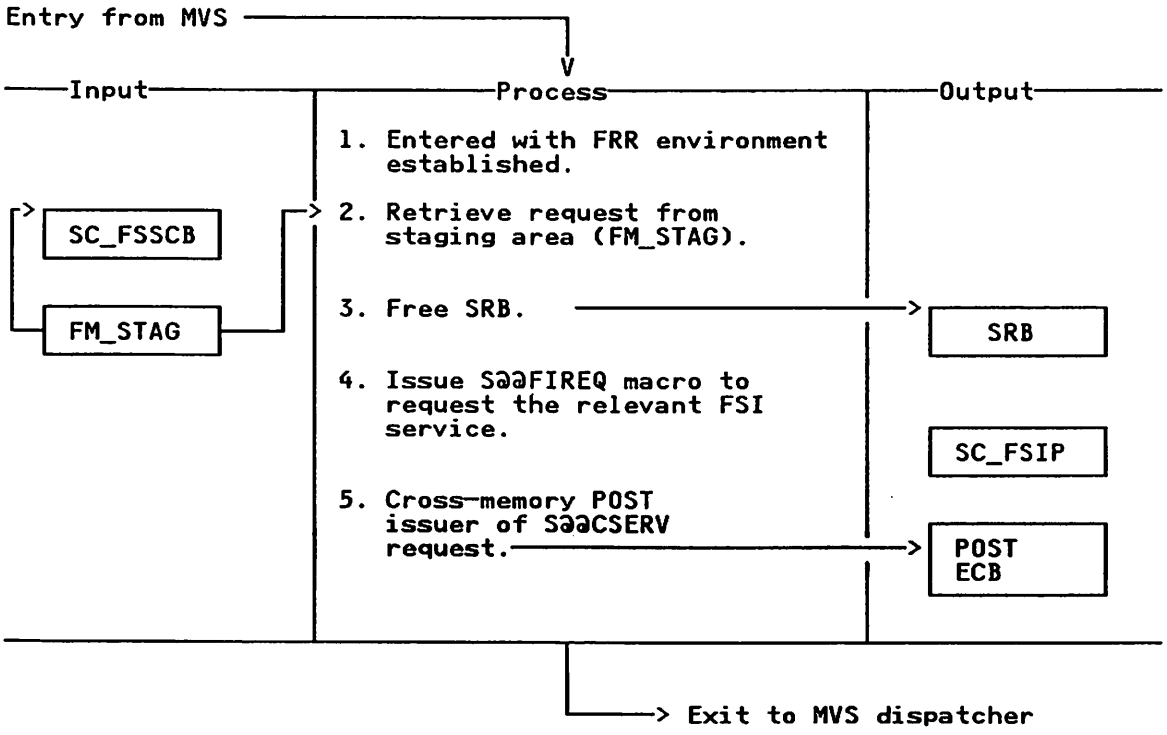
Diagram 4-8
S@CF0100 - SCHEDULE SRB Routine



Extended Description

Explanation	*Module*	*Label*
1. Register 1 points to the parameter list SC_SERV, which was built by the S@@CSERV macro in response to a cross-memory communications request. If the requested service is unavailable (for example, the request is invalid or the requested FSS is not available), an appropriate return code must be passed back to the caller.	S@CF0100	
3. The SRB used to dispatch the receive routine in the target FSS address space must be built in commonly addressable storage.	S@CF0100	
5. The SCHEDULE macro is used to place the newly built SRB on the SRB dispatch queue, which MVS will schedule later. The routine that will be dispatched is the "receive" routine, S@CF0110. It specifies that an FRR recovery environment will be established at the time the SRB is entered, and identifies this task as the associated task for the SRB.	S@CF0100	
6. The invoker of the S@@CSERV macro may have requested WAIT = YES or WAIT = NO on invocation. For WAIT = YES, this routine WAITs until POSTed by the target FSS. Otherwise, no WAIT is performed.	S@CF0100	
7. A return code passed in register 15 informs the caller of the success or failure of the request.	S@CF0100	
8. This ESTAE environment gains control when the FRR protecting the SRB routine (S@CF0130) continues with termination (percolates) rather than retrying. Any common storage not already freed (FM_STAG, SRB) is freed at this point, and any outstanding SRBs on the dispatch queue are purged with the PURGEDQ macro. This ESTAE retries to step 9.	S@CF0100	
9. The ESTAE in step 8 retries to this point, which sets a return code for the caller indicating that the SRB FRR gained control and that the staging area was not received by the target address space.	S@CF0100	

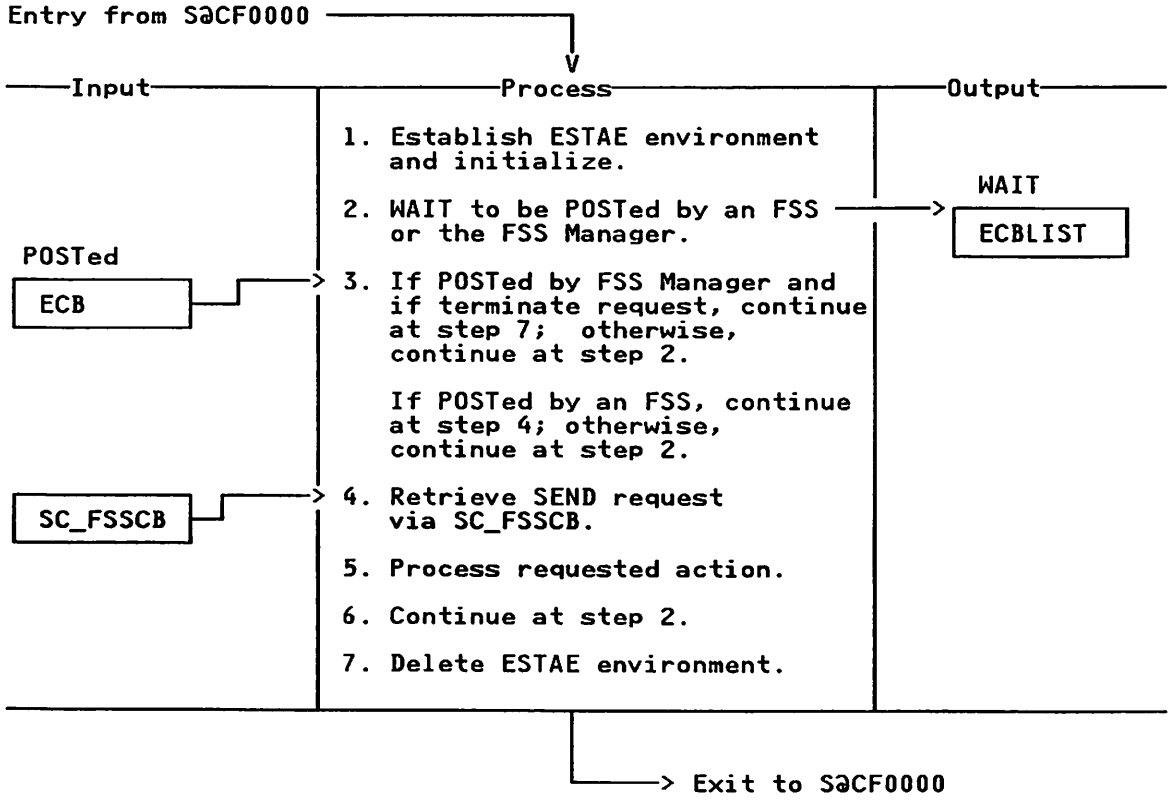
Diagram 4-9
S@CF0110 - SRB Receive Routine



Extended Description

Explanation	*Module*	*Label*
1. To effect recovery in case of an abend, this routine is entered with an FRR environment already defined. This was done when the SRB was SCHEDULEd from S@CF0100, which indicated that S@CF0130 was the FRR for this routine.	S@CF0110	
2. The request built by S@CF0100 in a staging area (FM_STAG) is retrieved to determine which request must be acted upon.	S@CF0110	
4. Once the requested action has been determined, the appropriate FSS Interface routine must be invoked to execute it. This is achieved by the F@@FIREQ macro, which finds the appropriate routine from the FM_FSVT and FM_FSCT and invokes it.	S@CF0110	
5. The control address space is cross-memory POSTed to notify the issuer of the S@@CSERV macro in the control address space of the receipt of the request.	S@CF0110	

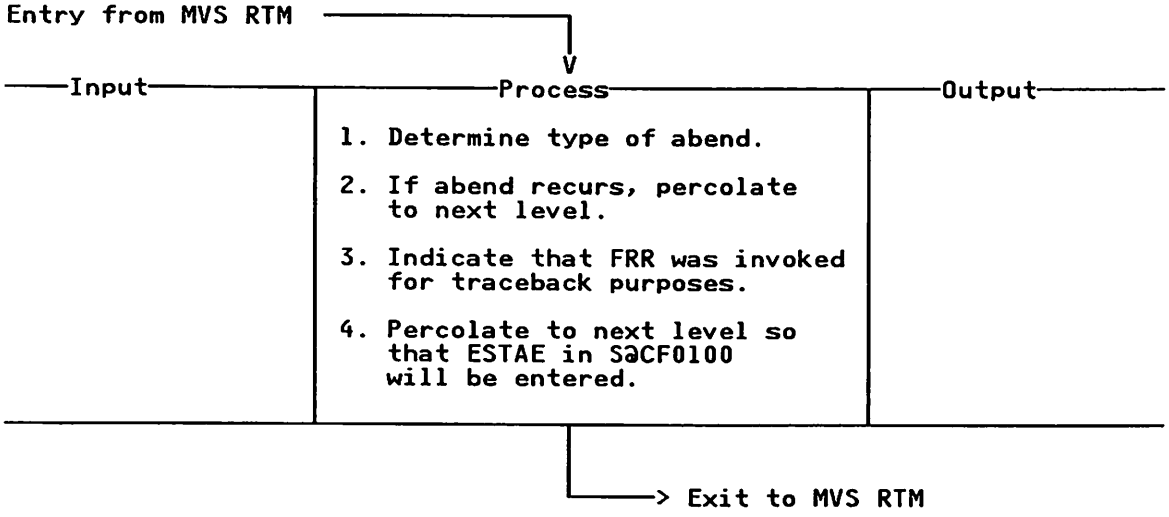
Diagram 4-10
S@CF0120 - Listen Task in Control Address Space



Extended Description

Explanation	*Module*	*Label*
1. To trap abends of this task and provide recovery facilities, an ESTAE environment is established. The ESTAE routine is module S@CF0140. Any initialization required is also handled at this point.	S@CF0120	
2. This is the main WAIT point of the whole module. Only two sources of wakeup are defined: • Wakeup by the FSS Manager • Wakeup by an FSS address space with data to pass	S@CF0120	
3. If a terminate request is received from the FSS Manager, the task should terminate cleanly. If there is work to be done in receiving data from the FSS, that unit of work should be processed. Any other source of stimulation results in the WAIT state being reentered.	S@CF0120	
4. The request from the FSS is retrieved from the SEND queue anchored from the SC_SSVT in staging area FM_STAG.	S@CF0120	
5. The request from the FSS is acted upon by passing the information received to the appropriate destination within the control address space. For example, if the request contained output from a previously-entered operator command, it should be sent to the component responsible for displaying the reply to the operator.	S@CF0120	
6. Reenter the WAIT state to wait for the next request.	S@CF0120	
7. Clear things up and terminate.	S@CF0120	

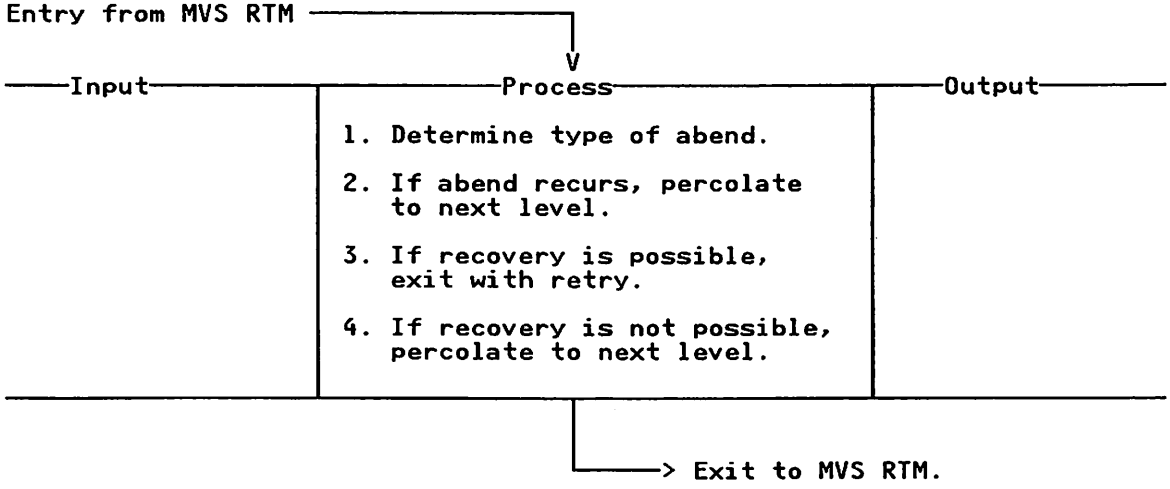
Diagram 4-11
S@CF0130 - Functional Recovery Routine for S@CF0110



Extended Description

Explanation	*Module*	*Label*
1. Routines running in SRB mode and requiring protection should be covered by an FRR. This routine protects SRB receive routine S@CF0110. FRR routines are always provided with an SDWA by RTM.	S@CF0130	
2. To prevent reentry in an abend-type loop, check for recursion. If this is a recursive abend, use the branch entry (since this is an FRR) to the SDUMP routine to obtain a dump for diagnostic purposes. The abend condition should be percolated to the next level.	S@CF0130	
4. Since this is an FRR, use the branch entry to the SDUMP routine to obtain a dump for diagnostic purposes. The abend condition should then be percolated to the next level using the SETRP macro. This causes the ESTAE in the associated task (S@CF0100) to gain control to perform more recovery in the originating address space.	S@CF0130	

Diagram 4-12
S@CF0140 - LISTEN Task ESTAE Routine



Extended Description

Explanation	*Module*	*Label*
2. To prevent reentry in an abend-type loop, check for recursion. If this is a recursive abend, use the SDUMP routine to obtain a dump for diagnostic purposes. Percolate the abend condition to the next level, where the FSS Manager detects this task's abend.	S@CF0140	
3. If recovery is possible, a retry routine can be specified on the SETRP macro, which gains control after this exit has returned control to RTM.	S@CF0140	
4. If recovery is not possible, use the SDUMP routine to obtain a dump for diagnostic purposes. Using the SETRP macro again, percolate the abend condition to the next level, where the FSS Manager detects this task's abend.	S@CF0140	

5. SUPERLINK Product Operator Component

The Product Operator component of SLCN controls the SUPERLINK/MVS product by processing operator commands entered from the MCS console interface.

Product Operator Module Structure

The Product Operator component consists of the following modules:

Module	*Description*
S@CO0000	Root module for the Product Operator component
S@CO0010	Product Operator initialization
S@CO0020	Parsing routine for all operator commands
S@CO0030	Parsing routine for all command operands
S@CO0040	Syntax graph describing all valid commands and their formats (this is a nonexecutable module)
S@CO0050	MCS console output processing routine
S@CO0060	Product Operator termination
S@CO0070	Product Operator ESTAE routine
S@CO0DIS	Root module processing routine for the DISPLAY command
S@COD010	Command processing routine for the DISPLAY SLUSERS command
S@COD020	Command processing routine for the DISPLAY NODES command
S@COD030	Command processing routine for the DISPLAY TABLES command
S@COD040	Command processing routine for the DISPLAY OFFERS command
S@COD050	Command processing routine for the DISPLAY LINKS command
S@COD060	Command processing routine for the DISPLAY AM command
S@COD070	Command processing routine for the DISPLAY SESSIONS command
S@COD080	Command processing routine for the DISPLAY STORAGE command
S@COD090	Command processing routine for the DISPLAY MIC command

Module	*Description*
S@COD100	Command processing routine for the DISPLAY FSS command
S@CO0SWT	Command processing routine for the SWITCH command
S@CO0SET	Root module processing routine for the SET command
S@COS010	Command processing routine for the SET SESSION command
S@COS020	Command processing routine for the SET CASE command
S@COS030	Command processing routine for the SET AM command
S@COS040	Command processing routine for the SET MI=XXXX command
S@CO0STR	Command processing routine for the START command
S@CO0STP	Command processing routine for the STOP command
S@CO0SND	Command processing routine for the SEND command
S@CO0MSG	Command processing routine for the MSG command

Figure 11 on page 5-3 shows the hierarchical structure of the modules within the Product Operator component.

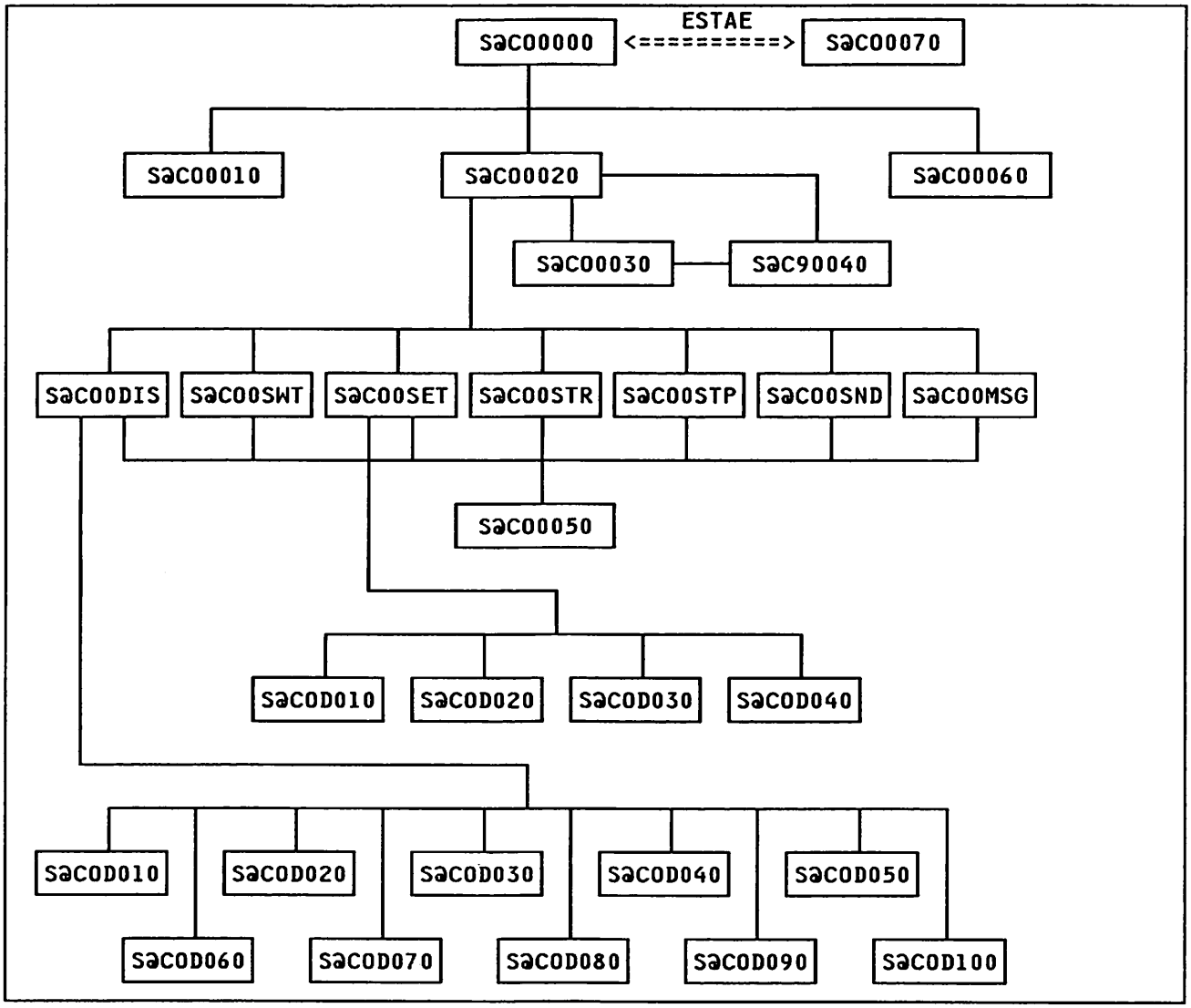


Figure 11. Module Structure of the Product Operator Component

Product Operator Services

The Product Operator component controls the SUPERLINK/MVS product by processing the following commands:

- DISPLAY
- MSG
- SEND
- SET
- START
- STOP
- SWITCH

The SUPERLINK/MVS Installation, Customization, and Tuning Guide, publication SI-0180, provides more detailed information about these commands.

Product Operator Command Definitions

Commands are defined to the Product Operator component by a syntax table (S@CO0040). This table is generated by a series of macro calls that define the attributes of each command and the syntax of their operands. The following macros are used:

Macro	*Description*
S@COADEF	This macro allows sets of alternate operands to be available on any command. Alternate operand entries may point to other alternate operand lists; in this way, complex syntaxes can be defined.
S@COCDEF	This macro defines the available commands and their attributes to the command parser and directs the parser to one or more operand entries that define the operands available on each command.
S@COKDEF	This macro allows definition of keyword operands. The attributes and values of the keywords are also defined.
S@COLDEF	This macro defines literal operands that can be used on a specific command.
S@COPDEF	This macro allows positional operands to be defined. The attributes and values of the operands are also defined.

"Appendix B. SLCN Macros" on page B-1 describes the syntax of these macros.

Product Operator Interfaces

The Product Operator component is activated by operator commands entered at an MCS console. The Subsystem Interface command processing routine, S@CC0S34, interrogates all commands presented to it by MVS. If a command other than a shutdown request is intended for SLCN, S@CC0S34 creates an operator command buffer, SC_OPCB, and queues it onto a chain anchored from the SC_SSVT. The Subsystem Interface cross-memory POSTs the Product Operator component to indicate that there is an operator command to be processed. The Product Operator component processes the commands in first-in-first-out (FIFO) order and continues processing until the queue is empty.

Product Operator Data Areas

The Product Operator component provides the following data areas:

Data Area	*Description*
SI_OPCT	Operator Control Table This is the major control block of the Product Operator component.

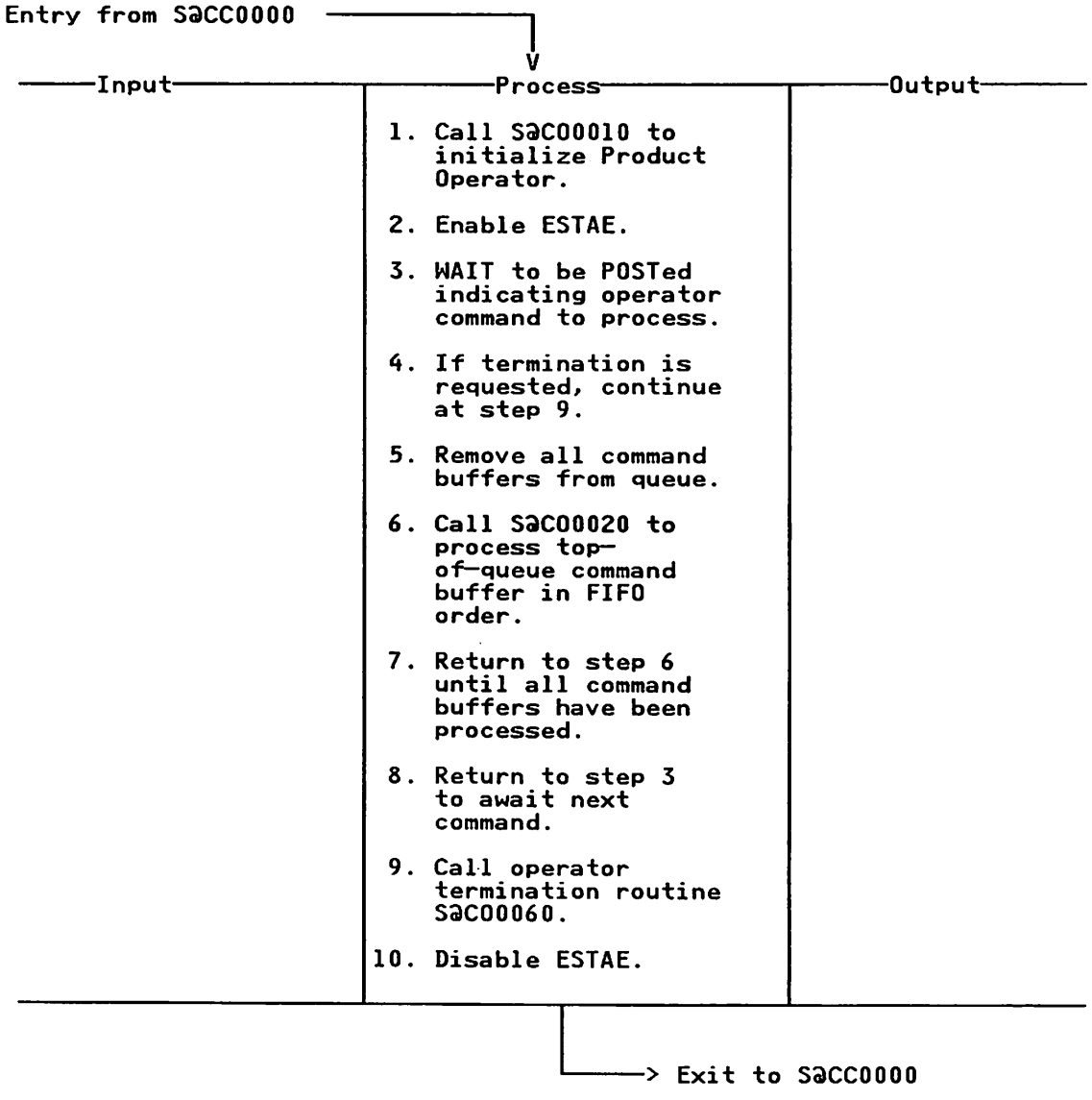
Data Area	*Description*
SC_OPCB	Operator Command Buffer The Subsystem Interface command processing routine, part of the Product Management component, creates the SC_OPCB control block to be used as input to the Product Operator component.
S@CO0040	The syntax table used to parse commands and their operands This syntax table consists of a number of entries mapped by the control block SI_CMD. Entries have slightly different mappings depending on whether they describe a command or a particular operand format. The highest-order index in the table consists of a list of command entries that are searched in linear order for a command verb match. Subordinate to each command entry is a syntax tree of the allowed operands for that command.

"Appendix A. Data Area Descriptions" on page A-1 provides format descriptions of these data areas.

Product Operator Recovery

The root module for the Product Operator component is protected by an ESTAE environment that traps ABENDs within the Product Operator component and attempts to recover from them whenever possible. If recovery is not possible, notification of the abend is percolated to the next level of the recovery environment. If an abend occurs during the processing of a command, the command is flushed.

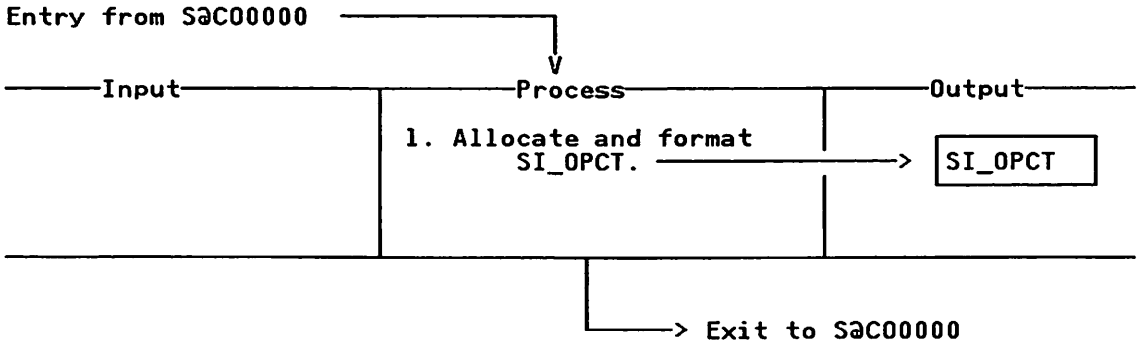
Diagram 5-1
S@CO0000 - SUPERLINK Product Operator Component Root Module



Extended Description

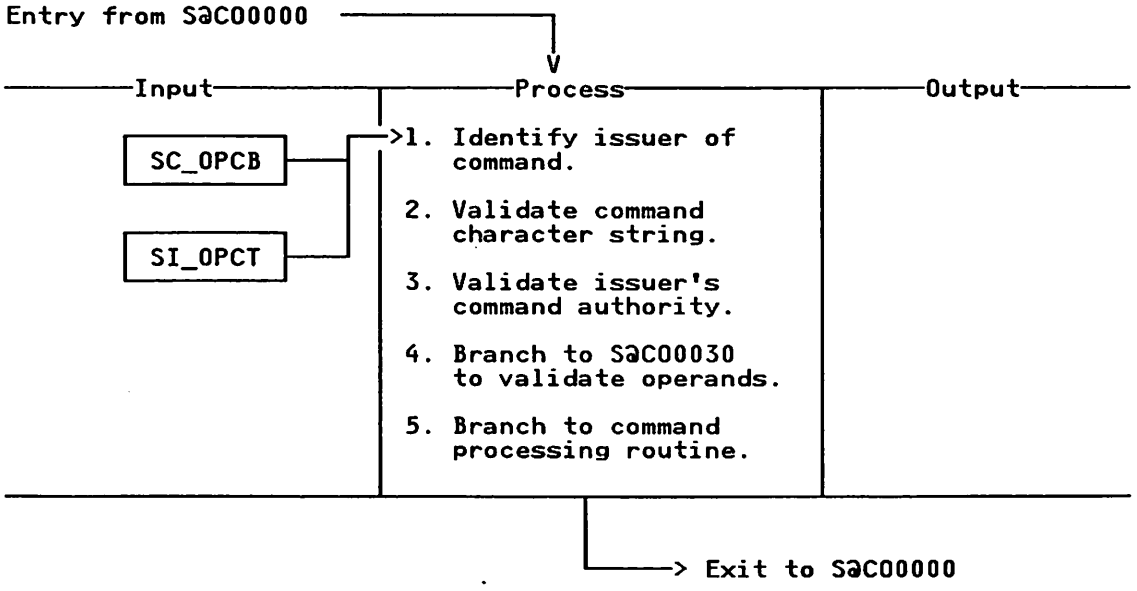
Explanation	*Module*	*Label*
5. The command buffer queue, anchored from the SC_SSVT, is in LIFO order. However, the operator commands must be processed in FIFO order. The complete queue of outstanding commands is now reorganized in preparation for processing.	S@CO0000	
7. The Product Operator component must ensure that all command buffers removed to form the FIFO queue are processed. These command buffers are processed before any commands that were chained to the LIFO queue in the SC_SSVT and after the command buffers were removed from the LIFO queue.	S@CO0000	

Diagram 5-2
S@CO0010 - Product Operator Initialization



This page has been intentionally left blank.

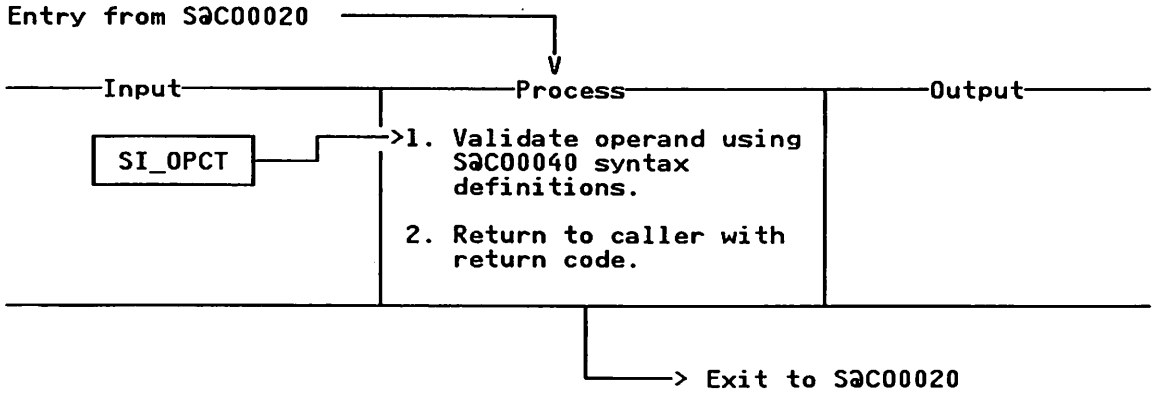
Diagram 5-3
S@CO0020 - SUPERLINK Product Operator Component Command Parser



Extended Description

Explanation	*Module*	*Label*
2. A syntax graph of valid operator commands and their operands is available from module S@CO0040. The syntax graph is searched for the relevant command entry, using the command character string input. If an entry is not located, the command is invalid, and an error message must be returned to the point of origin. In this case, an error message is formatted and the process branches to module S@CO0050, the Product Operator MCS console output processing routine. This routine issues the message to the appropriate MCS console.	S@CO0020	
3. The command issuer's authority must be checked before the command can be acknowledged as valid. If the issuer is not allowed to issue this command (authority levels are also defined within the syntax graph command definitions), an error message is formatted and a call is made to module S@CO0050, which outputs the message to the appropriate MCS console.	S@CO0020	
5. The syntax graph identifies the command processing module that supports the command which was requested by the operator. Processing branches to that routine, where the requested action is performed.	S@CO0020	

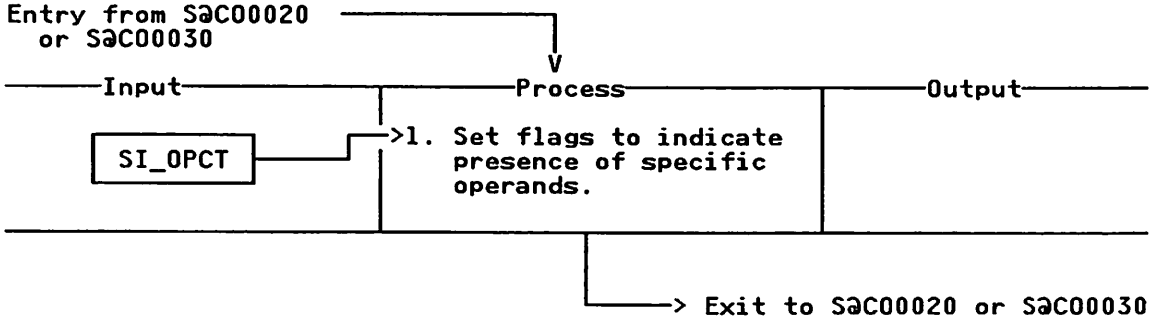
Diagram 5-4
S@CO0030 - Product Operator Operand Parser



Extended Description

Explanation	*Module*	*Label*
1. The operands are validated according to their syntax description, that is, as keywords, positional parameters, or literals. Indicators are set in the SI_OPCT to direct the command processing routines. If an operand is found to be invalid, an error message is formatted. A branch to module S@CO0050 causes the message to be output to the MCS console of origin.	S@CO0030	

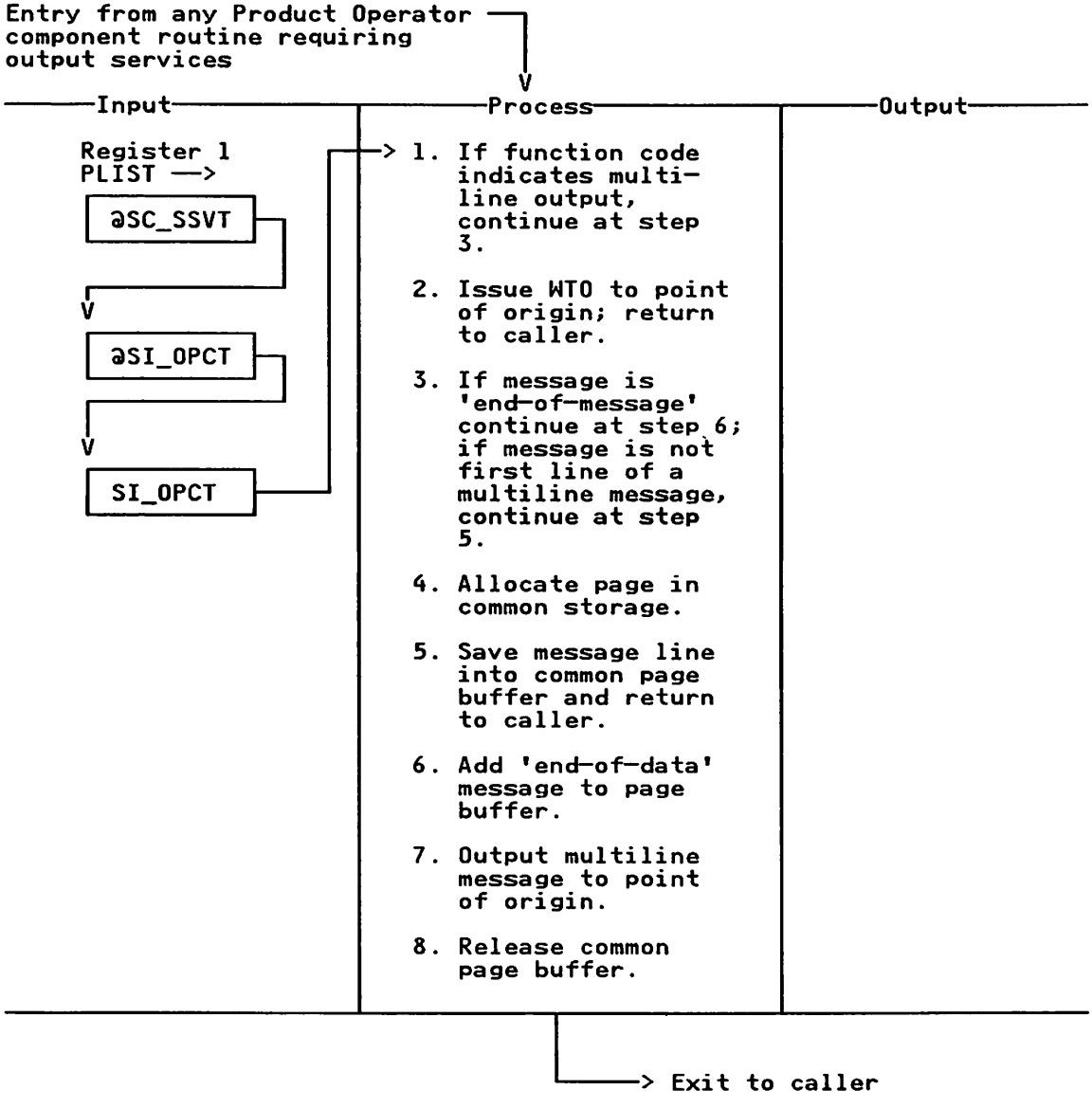
Diagram 5-5
S@CO0040 - Product Operator Syntax Graph



Extended Description

Explanation	*Module*	*Label*
1. A binary field is reserved in the SI_OPCT for each possible operand of a command. This field indicates the presence of the operand and, if necessary, its value.	S@CO0040	

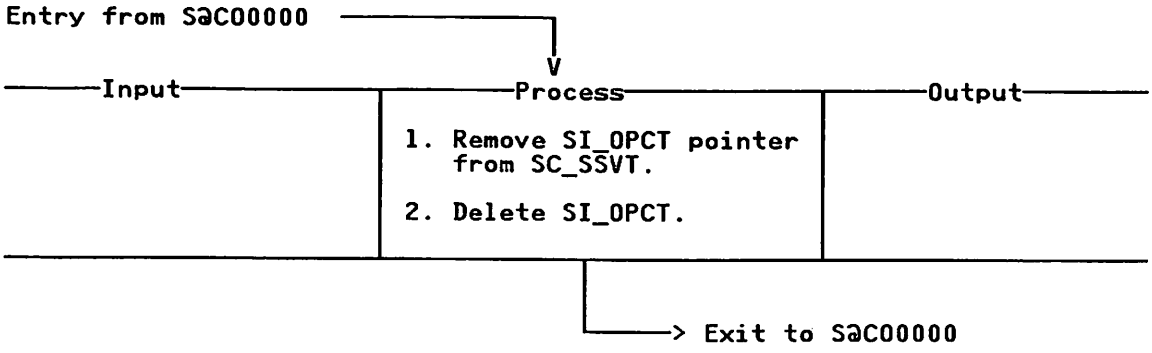
Diagram 5-6
S@CO0050 - SUPERLINK Product Operator Component Output Processor



Extended Description

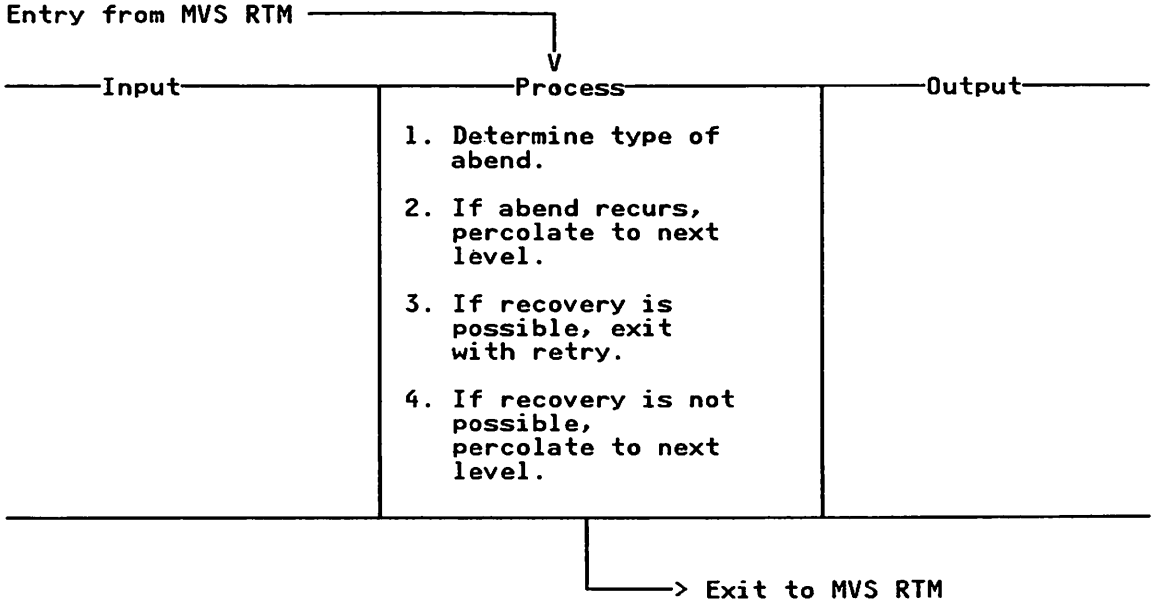
Explanation	*Module*	*Label*
1. On entry to module S@CO0050, register 1 points to a parameter list that contains a pointer to the SC_SSVT (and thence to the SI_OPCT) and a function code. This code indicates which type of processing the routine is to perform. The output may be a single line of data or part of a multiline message. A multiline message includes two types of data line: • **Control line** - First line of the message • **Data line** - All subsequent lines within the message To signify that all data has been transferred to the output processing routine, a further call, indicating "end-of-data", is made to S@CO0050. No data line processing is performed on this call.	S@CO0050	
2. The SI_OPCT control block holds the information concerning the console identifier of the console of origin.	S@CO0050	
5. The message for output may be longer than a 4K page of storage. If so, the data is displayed to the console 4K at a time. After one 4K page of data has been displayed, the 4K page is reused for the remainder of the message output.	S@CO0050	
6. An "end-of-data" message is appended to the page buffer, indicating to the operator that all data has been displayed.	S@CO0050	
7. Each message line from the buffer area is output to the point of origin; information concerning command origin is found in the SI_OPCT. The WTO service is used to output the data.	S@CO0050	

Diagram 5-7
S@CO0060 - SUPERLINK Product Operator Component Termination



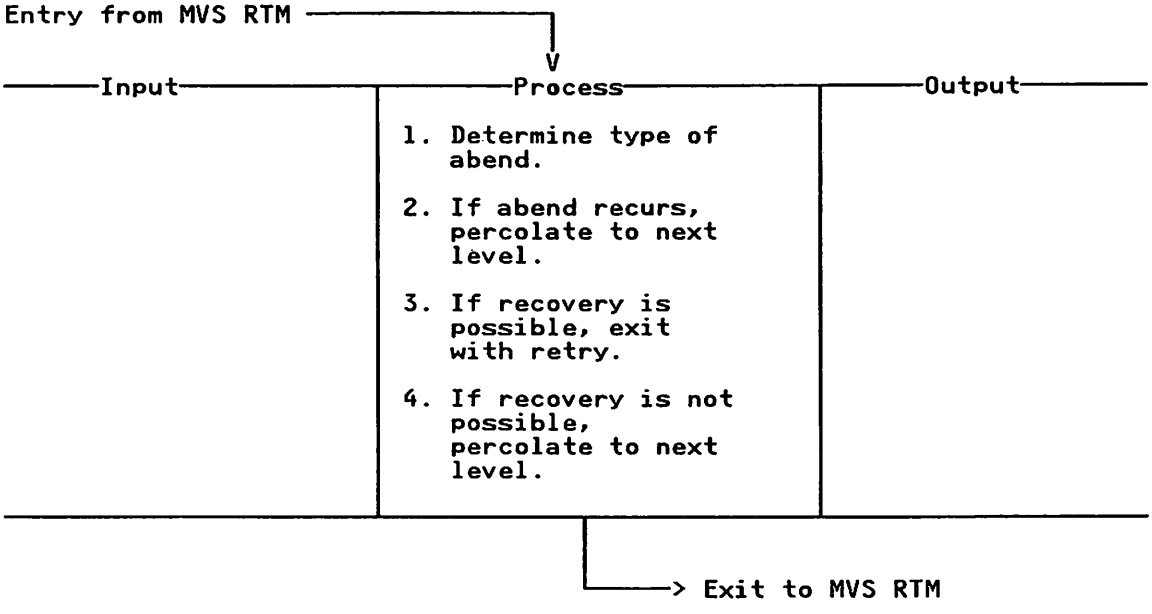
This page has been intentionally left blank.

Diagram 5-8
S@CO0070 - Product Operator ESTAE



This page has been intentionally left blank.

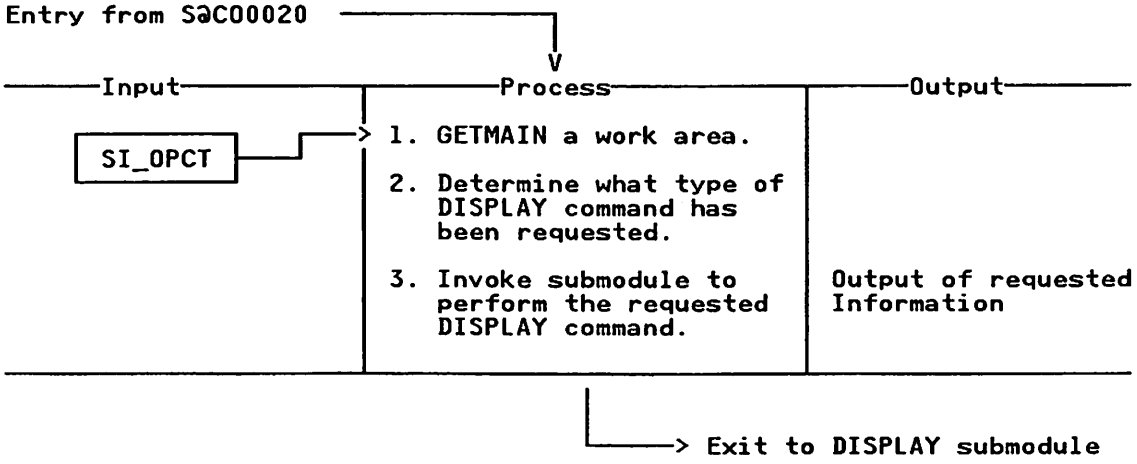
Diagram 5-8
S@CO0070 - Product Operator ESTAE



Extended Description

Explanation	*Module*	*Label*
1. No retry is attempted on X22-type abends, nor is a dump obtained.	S@CO0070	
2. To prevent reentry in an abend recursive loop, check for recursion. If this is a recursive abend, control should simply be percolated to the next level of recovery, where the SLCN control module detects the termination of the Product Operator component.	S@CO0070	
3. If recovery is possible for this abend and an SDWA is present, a retry routine is specified on the SETRP macro. This routine gains control after the ESTAE has returned control to RTM. A dump is also obtained for diagnostic purposes with the SDUMP routine. The retry routine attempts to recover the existing SI_OPCT or builds a new one if the old one has been lost. The operator task reinitializes itself and attempts to process the next command on the command queue.	S@CO0070	
4. If recovery is not possible, use the SDUMP routine to obtain a dump for diagnostic purposes. Using the SETRP macro again (if an SDWA is obtained), percolate the abend condition to the next level of recovery. At this level, the SLCN root module detects this task's termination.	S@CO0070	

Diagram 5-9
S@CO0DIS - DISPLAY Command Processing Routine

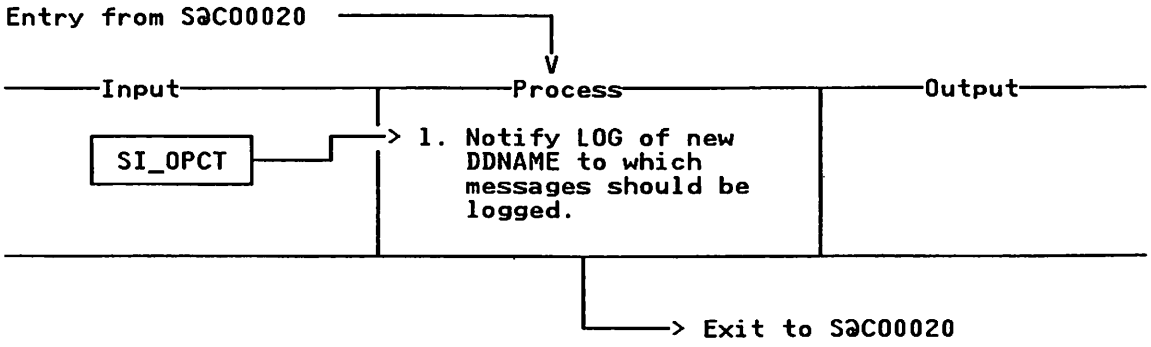


Extended Description

Explanation	*Module*	*Label*
2. As each operand on the DISPLAY command was validated, an indicator was set in the SI_OPCT control block. This routine must interrogate each of these indicators in turn to identify which operands have been input by the operator.	S@CO0DIS	
3. The appropriate sub-modules available are as follows:	S@CO0DIS	

- S@COD010 (DISPLAY SLUSERS) - Display a List of all SUPERLINK Users (using the SC_SLASVT)
- S@COD020 (DISPLAY NODES) - Display a list of network nodes (using the Link Table)
- S@COD030 (DISPLAY TABLES) - Display the address's of some SUPERLINK control blocks
- S@COD040 (DISPLAY OFFERS) - Display a list of all application titles on Offer (Using the ATE's)
- S@COD050 (DISPLAY LINKS) - Display a list of entries in the Link Table (Using Link Table entries)
- S@COD060 (DISPLAY AM) - Display Association Manager values and usage (Using AM Tables)
- S@COD070 (DISPLAY SESSIONS) - Display a list of connections (using the ATE's)
- S@COD080 (DISPLAY STOARGE) - Display SLNET buffer usage (Using the SMB)
- S@COD090 (DISPLAY MIC) - Display Management Interface connections (Using the MI_MACB)
- S@COD100 (DISPLAY FSS) - Display a list of Functional Subsystems (Using the FSSCB)

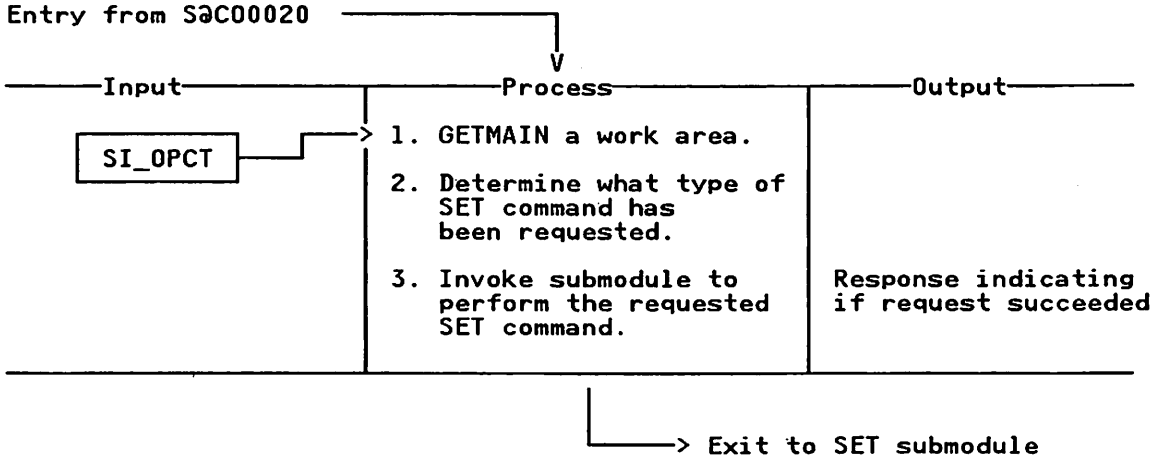
Diagram 5-10
S@CO0SWT - SWITCH Command Processing Routine



Extended Description

Explanation	*Module*	*Label*
1. As each operand on the SWITCH command was validated, an indicator was set in the SI_OPCT control block. This routine must interrogate each of these indicators in turn to identify which operands have been input by the operator. The DDNAME field is extracted. The LOG task is notified of the change in the logging DDNAME by completing the command field to be logged in the SC_SSVT and passing the new DDNAME across this interface.	S@CO0SWT	

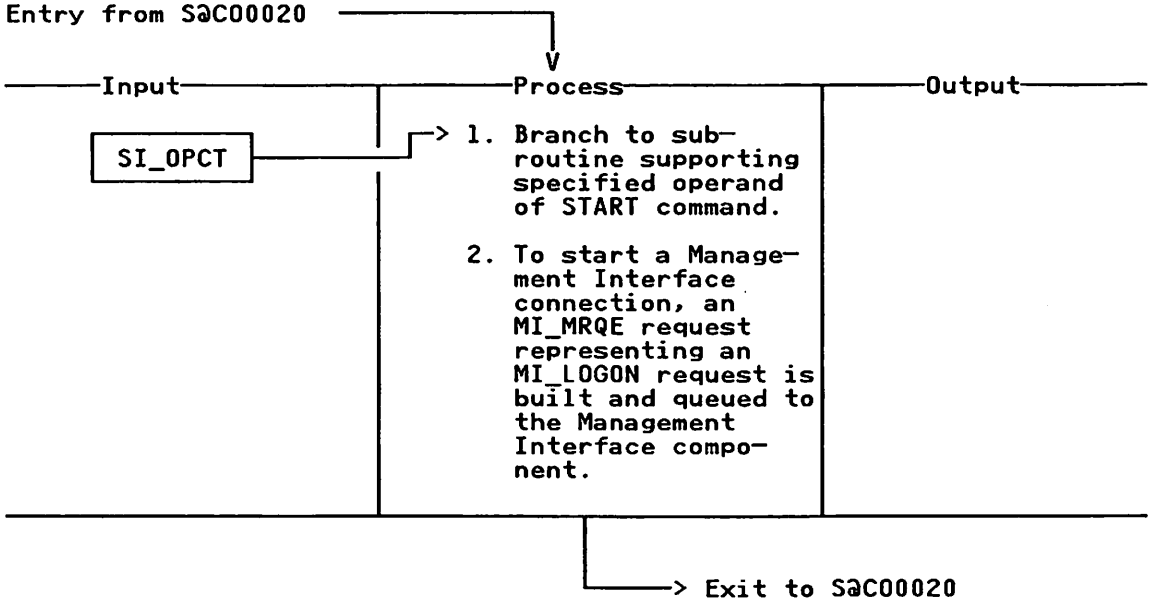
Diagram 5-11
S@CO0SET - SET Command Processing Routine



Extended Description

Explanation	*Module*	*Label*
2. As each operand on the SET command was validated, an indicator was set in the SI_OPCT control block. This routine must interrogate each of these indicators in turn to identify which operands have been input by the operator.	S@CO0SET	
3. The appropriate sub-modules available are as follows: • S@COS010 (SET SESSION) - Set the Sessions layer dump or trace options on or off • S@COS020 (SET CASE) - Set the ACSE layer dump or trace options on or off • S@COS030 (SET AM) - Modify Association Manager variables or set the dump option on or off • S@COS040 (SET MI=XXXXXX) - Set the Management Interface trace option on or off for a specific connection	S@CO0SET	

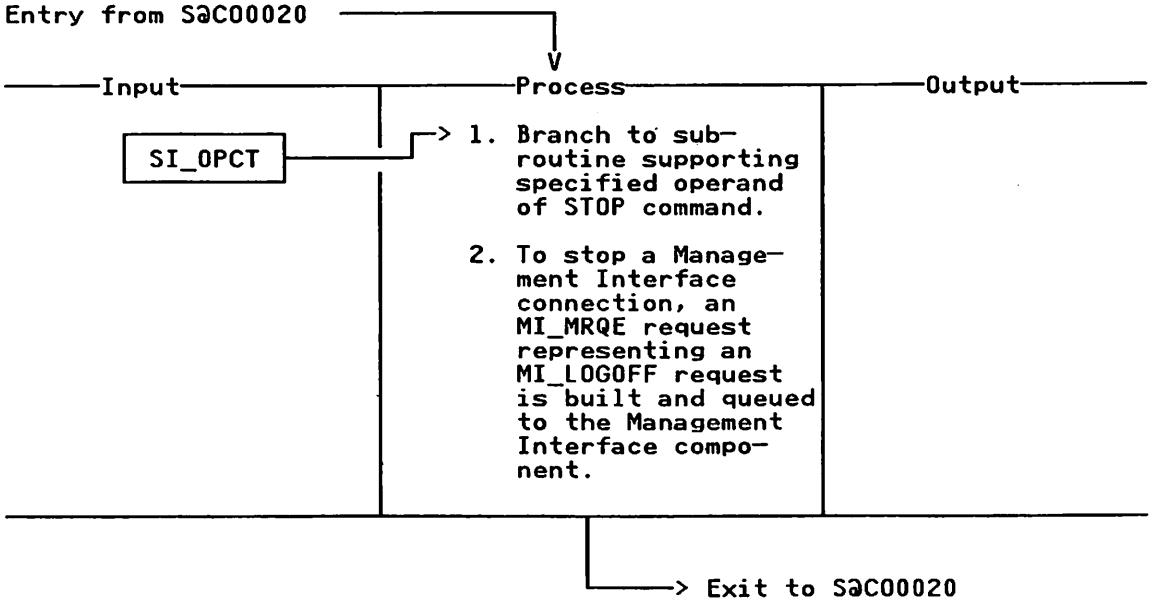
Diagram 5-12
S@CO0STR - START Command Processing Routine



Extended Description

Explanation	*Module*	*Label*
1. As each operand on the START command was validated, an indicator was set in the SI_OPCT control block. This routine must interrogate each of these indicators in turn to identify which operands have been input by the operator. A subroutine residing within S@CO0STR for each valid operand collects the relevant information and formats it into a message for output to the MVS console of origin. Routine S@CO0050 performs the output processing.	S@CO0STR	

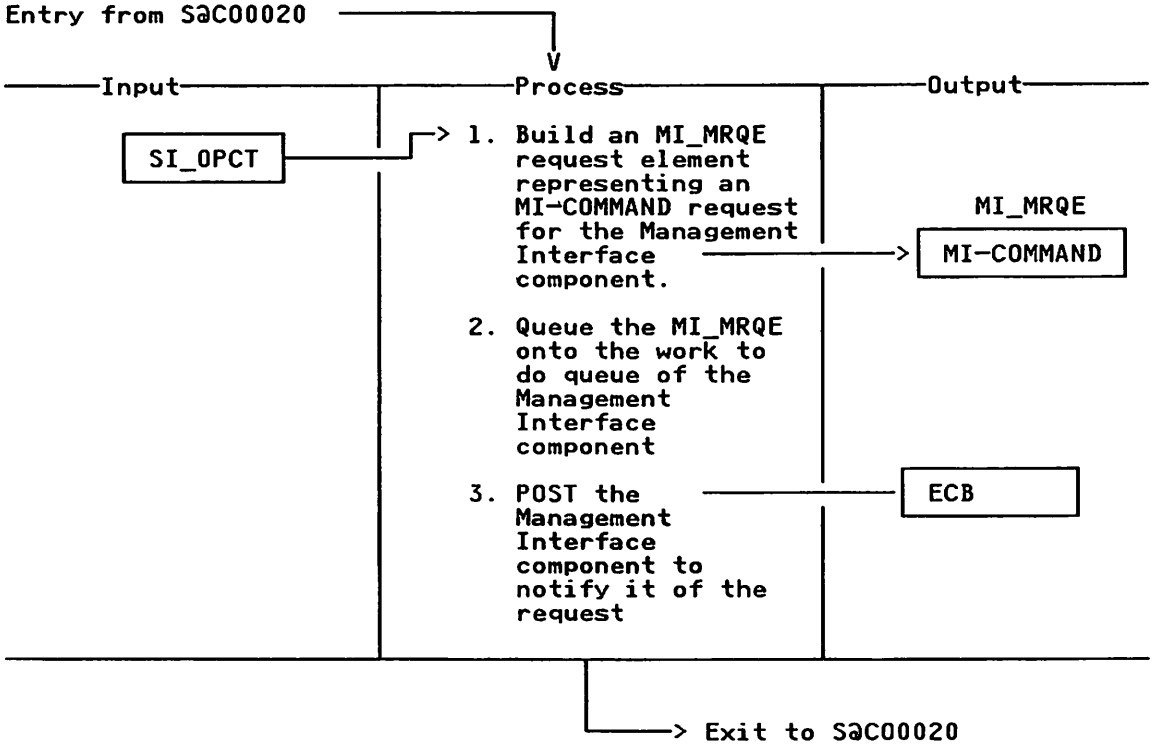
Diagram 5-13
S@CO0STP - STOP Command Processing Routine



Extended Description

Explanation	*Module*	*Label*
1. As each operand on the STOP command was validated, an indicator was set in the SI_OPCT control block. This routine must interrogate each of these indicators in turn to identify which operands have been input by the operator. A subroutine residing within S@CO0STP for each valid operand collects the relevant information and formats it into a message for output to the MVS console of origin. Routine S@CO0050 performs the output processing.	S@CO0STP	

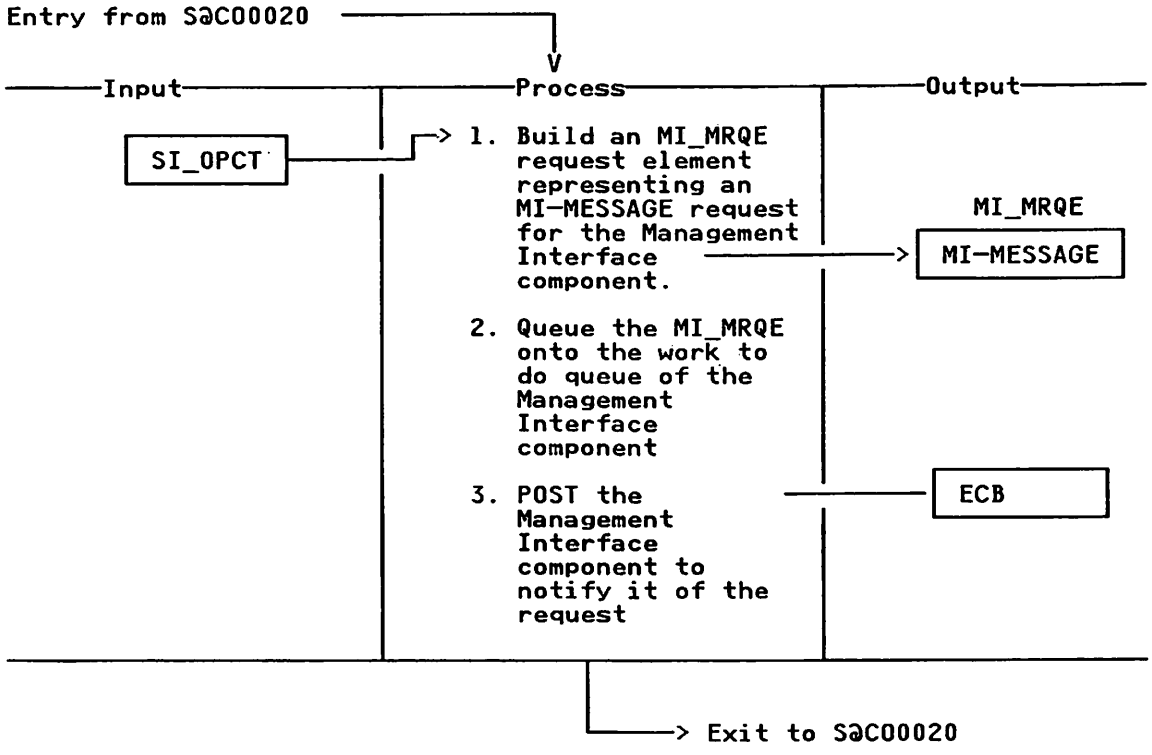
Diagram 5-14
S@CO0SND - SUPERLINK SEND Command Processing Routine



Extended Description

Explanation	*Module*	*Label*
1. As each operand on the SEND command was validated, an indicator was set in the SI_OPCT control block. This routine must interrogate each of these indicators in turn to identify which operands have been input by the operator. Since the operands on this command represent a command destined for sending over the Management Interface component, a request unit for the Management Interface is built containing the imbedded command for the target system. This request unit (MI_MRQE) represents an MI-COMMAND request on the target connection.	S@CO0SND	
2. The MI_MRQE, which is built, is placed on the Management Interface work to do queue which is anchored off the MI_MICT control block which in turn is anchored from the SC_SSVT.	S@CO0SND	

Diagram 5-15
S@CO0MSG - SUPERLINK MSG Command Processing Routine



Extended Description

Explanation	*Module*	*Label*
1. As each operand on the MSG command was validated, an indicator was set in the SI_OPCT control block. This routine must interrogate each of these indicators in turn to identify which operands have been input by the operator. Since the operands on this command represent a command destined for sending over the Management Interface component, a request unit for the Management Interface is built containing the imbedded command for the target system. This request unit (MI_MRQE) represents an MI-MESSAGE request on the target connection.	S@CO0MSG	
2. The MI_MRQE, which is built, is placed on the Management Interface work to do queue which is anchored off the MI_MICT control block which is in turn anchored from the SC_SSVT.	S@CO0MSG	

6. SUPERLINK LOG Processor Component

SUPERLINK components provide progress messages for normal events and indications of error conditions. These messages are queued by the relevant Functional Unit for subsequent processing by the LOG Processor component. The LOG processor component outputs the message data to a user-defined dataset.

LOG Processor Services

The LOG Processor component allows all other SUPERLINK components to write messages to the SUPERLINK LOG and/or the MVS system log. Messages may consist of a single one-line message, a single multiline message, or a number of messages output as a block. Each line of output is built from a LOG element (LOGE).

The SUPERLINK LOG is allocated to either a dataset or SYSOUT depending on the SLLOG DD statement coded in the SUPERLINK Control JCL procedure. Coding SLLOG DD DUMMY or DSN=NULLFILE results in output to the MVS system log. If the DD statement is absent, the SUPERLINK LOG is dynamically allocated to SYSOUT, class A. If SLLOG cannot be allocated, or an I/O failure occurs during processing, an attempt is made to use SYSOUT class A. If this attempt also fails, the MVS system log is used. The site systems programmer specifies the route code(s) used when outputting to the MVS system log on a SUPERLINK/MVS options statement.

The SWITCH operator command can be used to specify the DDNAME of a preallocated alternate log. This command allows the printing of a dataset log while SUPERLINK/MVS is running. When the SWITCH command is used, SUPERLINK/MVS closes the primary log and opens the alternate log. MVS utilities or TSO may be used to manipulate the primary log after the switch is complete.

The SUPERLINK/MVS Installation, Customization, and Tuning Guide, publication SI-0180, provides more detailed information about the SWITCH command.

LOG Processor Subcomponents

The LOG Processor component consists of the following subcomponents:

Subcomponent	*Description*
S@C2100	LOGE handler
S@C2200	Output of messages

S@C2100 - LOGE Handler

SUPERLINK/MVS components call the LOGE handler subcomponent to obtain storage and chain together a number of LOGEs. The LOGEs are formatted and then a call is made to queue the LOGEs.

S@C2100 - Services

The LOGE handler performs the following services:

- Checks to ensure that SUPERLINK/MVS logging is active
- Obtains and chains together requested LOGEs for a GET request
- Queues LOGEs for a QUEUE request
- Informs the caller of actions taken

S@C2100 - Interfaces

The addition of queue elements has two phases. The first phase returns a number of chained LOGEs. The second phase places the LOGEs on the queue after they have been formatted. A macro called S@@LOG is provided as the interface to the second phase.

The sequence of actions required to chain and queue LOGEs is as follows:

1. Call S@C2100 to obtain LOGEs.
2. Format LOGEs (message text, message ID, routing codes, and so on).
3. Call S@C2100 to queue LOGEs.

Components issuing S@@LOG do so in 31-bit addressing mode. Therefore, the address returned by S@@LOG is a 31-bit address. The message length in all returned LOGEs (after GET) is set to 0. "Appendix B. SLCN Macros" on page B-1 describes the syntax for the S@@LOG macro.

S@C2100 - Data Areas

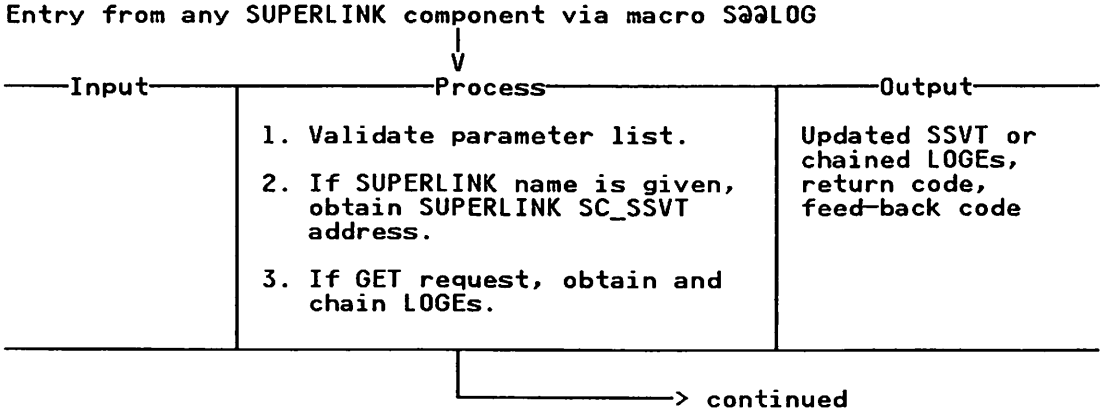
The LOGE is the major data area used.

S@C2100 - Recovery

There is no specific ESTAE for S@C2100. If the caller has an active ESTAE routine when S@C2100 is called, it gains control if an abend occurs.

This page has been intentionally left blank.

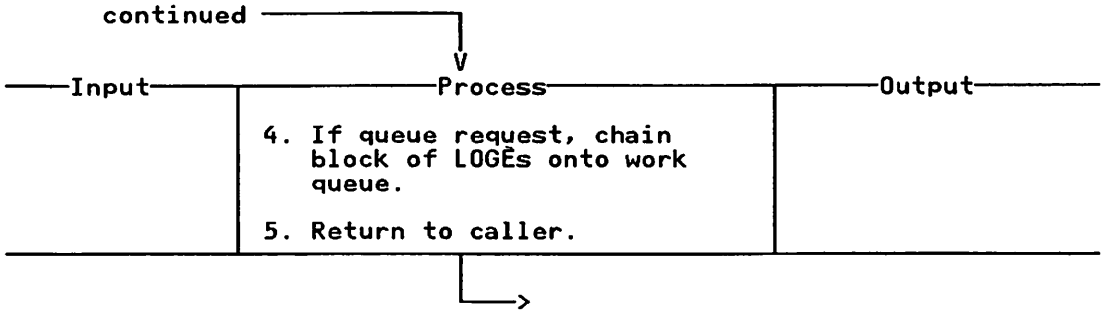
Diagram 6-1
S@C2100 - LOGE Handler (part 1 of 2)



Extended Description

Explanation	*Module*	*Label*
1. The parameter list consists of two fullwords. Register 0 describes the contents of the parameter list. The first fullword is an address, and the area pointed to is either the SUPERLINK/MVS SC_SSVT or the SUPERLINK/MVS name. The second fullword contains either a count of LOGEs to be obtained or the address of chained LOGEs to be queued. A return and feed-back code (see macro S@@LOG) are given identifying an invalid parameter and a branch to step 4 is taken. The SC_SSVT address may be 0 if the calling SUPERLINK/MVS component resides in the SLCN address space. In this case, the SC_SSVT address is found here; otherwise, an error is indicated.	S@C2100	
2. The macro S@@SUBSY is used to return the SUPERLINK/MVS SC_SSVT address given the SUPERLINK/MVS name. In the event of failure (name not known), a bad parameter indication is given (see step 1). The SC_SSVT is required in order to locate the queue anchor or pool address.	S@C2100	
3. Each LOGE is obtained individually (CPOOL, pool address in SC_SSVT) and chained to the previous one. The pointer to the next LOGE is set to 0. The message length is set to 0. On return to the caller, register 1 contains the address of the first LOGE. If there is insufficient storage to obtain an LOGE, the actions taken depend on the point at which the error occurred. If the first LOGE was obtained, a warning return code is set; register 0 is set to the count of LOGEs obtained; and a branch is taken to step 5. If the first LOGE was not obtained, an error return code is set, and a branch is taken to step 5. Control is returned to the caller.	S@C2100	

Diagram 6-2
S@C2100 - LOGE Handler (part 2 of 2)



Extended Description

Explanation	*Module*	*Label*
4. The work queue anchor is located in the SUPERLINK SC_SSVT. Using a compare double and swap instruction, the address of the chained LOGEs supplied replaces the head of the queue. The address of the previous LOGE chain is stored in the newly added LOGE chain. The work queue is in LIFO order. The message length in each LOGE is validated and, if outside the range (0 is valid), is replaced with the maximum length allowed. In the latter case, a warning return code is sent to the caller.	S@C2100	
5. The caller receives a return code and, optionally, a feedback code to indicate the processing performed.	S@C2100	

S@C2200 - Output of Messages Subcomponent

The output of messages subcomponent takes messages from the queue, writes them to the appropriate log(s), one at a time, and returns the LOGEs to the pool for reuse. This subcomponent must complete initialization before components are able to obtain LOGEs. Otherwise, components receive the following return code: SUPERLINK Logging Inactive.

S@C2200 - Module Structure

S@C2200 consists of the following modules:

Module	*Description*
S@C2200	Output of messages - control module
S@C2210	Output of messages - log file initialization
S@C2220	Output of messages - log file termination
S@C2230	Output of messages - queue swap and reorder
S@C2240	Output of messages - format and output a message from a LOGE
S@C2250	Output of messages - ESTAE

Figure 12 shows the hierarchical structure of modules within the output of messages subcomponent.

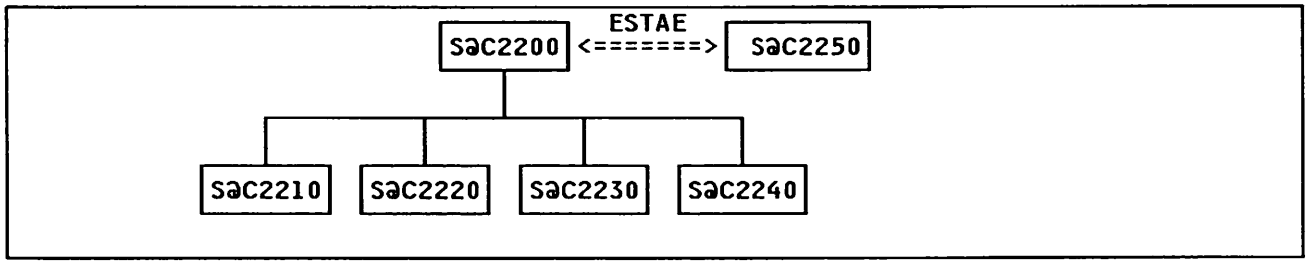


Figure 12. Module Structure of the Output of Messages Subcomponent

S@C2200 - Services

S@C2200 performs the following services:

- Informs the Product Management component (S@CC0000) of product initialization status
- Swaps the request and work queues to allow queuing to continue
- Outputs messages from the swapped queue

S@C2200 - Interfaces

The LOGE functions as the interface to this subcomponent and describes the messages to be written to a log. The LOGEs are queued for processing, and this queue is anchored from the SC_SSVT. -- Heading id 'cndarea' unknown -- provides a more detailed description of SC_SSVT.

S@C2200 - Data Areas

The major data area is the LOGE, which describes a message and specifies the log where it will be output. LOGEs are chained together for processing.

The LOG Processor component's queues, built in last-in, first-out (LIFO) order, are anchored in the SSVT. Each queue anchor is a doubleword. Thus, two queues are maintained: a work queue and a request queue. The request queue is updated by the addition of LOGEs; the work queue is null. The queues are swapped, the work queue (previously the request queue) is reversed (FIFO ordered), and individual LOGEs are written to the log(s) and released to the pool when the LOGEs are removed.

Storage for the queues is obtained and managed using the MVS service, CPOOL. The address of the pool is stored in the SSVT. Each LOGE's contents are mapped by the S@@LOG macro.

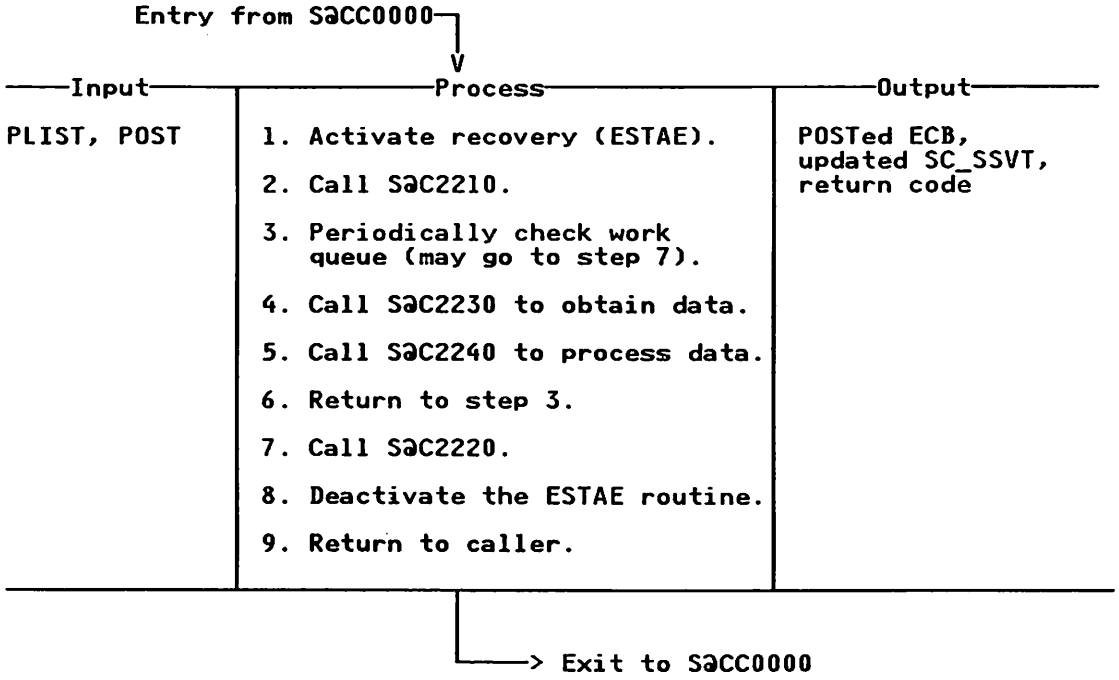
"Appendix B. SLCN Macros" on page B-1 describes the syntax of the S@@LOG macro.

S@C2200 - Recovery

The task that outputs messages has an ESTAE to intercept abends. In the case of I/O-related abends (for example, SB37), processing continues with messages written to SYSOUT or the MVS system log, as appropriate.

For other abends, the system log is closed, and diagnostic information is returned to the caller. Following LOG closure, users of S@@LOG receive return code 16, "SUPERLINK logging inactive".

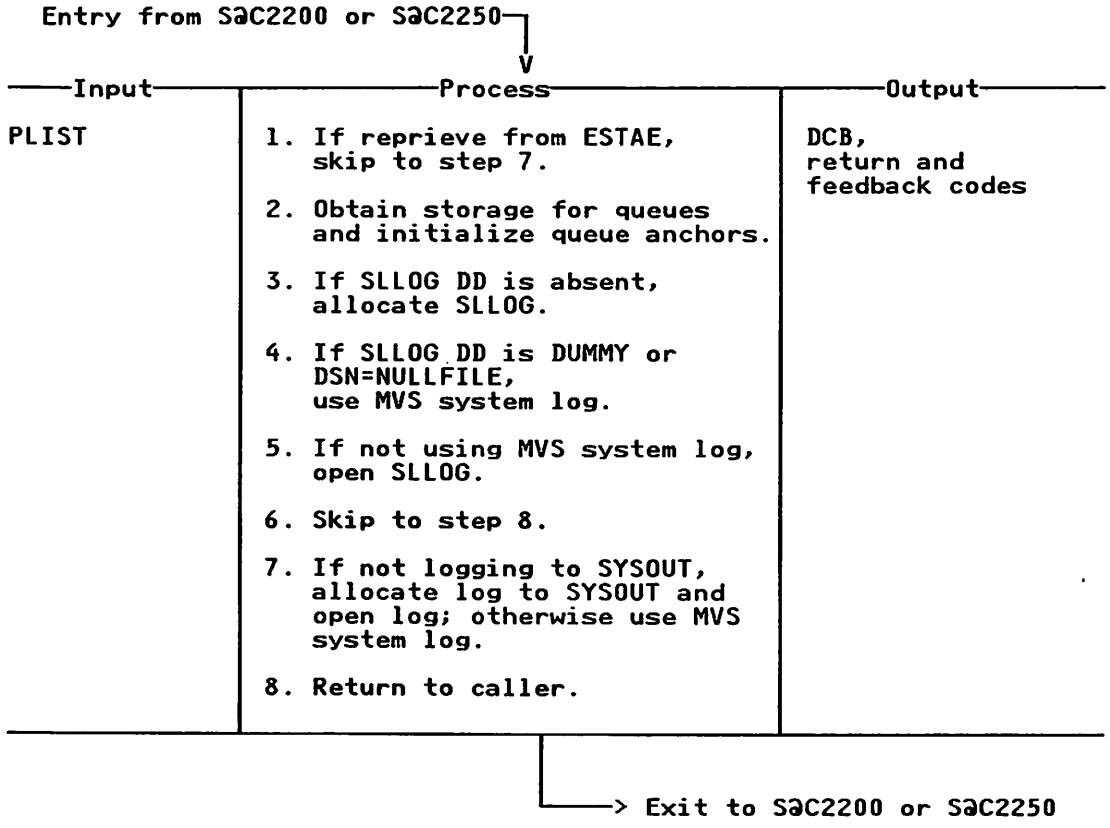
Diagram 6-3
S@C2200 - Output of Messages Subcomponent - Control Module



Extended Description

Explanation	*Module*	*Label*
1. The ESTAE is required to switch output to SYSOUT or the system log for I/O- related abends in an attempt to allow SUPERLINK/MVS to continue. For other abends, the LOG is terminated. The MVS operator receives messages indicating the type of error and proposed action(s) to be taken.	S@C2200	
2. Module S@C2210 initializes the LOG file. In the event of failure, control is returned to the caller with a bad return code. If initialization is successful, the caller is POSTed (ECB address in PLIST), and processing continues.	S@C2200	
3. This module is the head of a continuously-running task. A STIMER is used to wait when the work queue is empty. A field in the parameter list is checked to see if termination has been ordered by the caller. If ABORT termination is ordered, or the work queue is empty for NORMAL termination, a branch to step 7 is taken. The caller may request that a switch be made to an alternative preallocated log (DDNAME supplied). This is done by branching to step 7 and then to step 2 (not terminating).	S@C2200	
7. Module S@C2220 is called to terminate the LOG file.	S@C2200	
9. The caller receives return and feedback codes indicating the processing performed.	S@C2200	

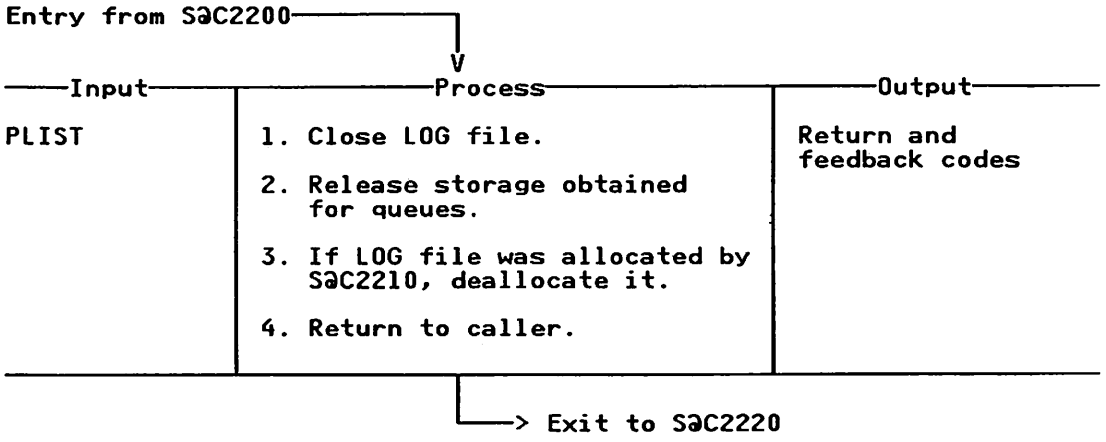
Diagram 6-4
S@C2210 - Output of Messages Subcomponent - LOG File Initialization



Extended Description

Explanation	*Module*	*Label*
1. The PLIST indicates whether normal processing or reprieve processing is required. The PLIST may contain a DDNAME; if so, branch to step 5.	S@C2210	
2. Storage is obtained using CPOOL. Each queue anchor consists of a doubleword. There are two queues currently in use: request and work. The free queue is managed by CPOOL and the anchor points to the pool. The queue anchors are located in the SC_SSVT.	S@C2210	
3. SLLOG DD should be allocated to SYSOUT=A and the caller informed that it was dynamically allocated (S@C2220 requires this information).	S@C2210	
5. Upon return to the caller, register 1 contains the address of the DCB for the log. If the system log is in use, register 1 is set to 0.	S@C2210	
7. An attempt is made to use a SYSOUT log in place of a dataset log. In the event of failure, or if SYSOUT was already in use, the system log is used. Allocation and open processing are as described in steps 2 and 4.	S@C2210	
8. The caller receives return and feedback codes indicating the processing performed.	S@C2210	

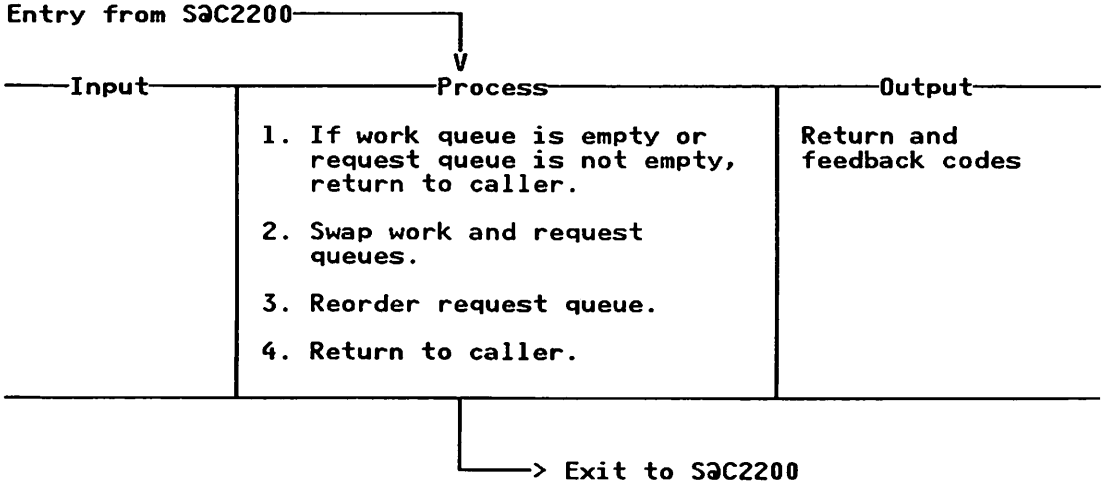
Diagram 6-5
S@C2220 - Output of Messages Subcomponent - LOG File Termination



Extended Description

Explanation	*Module*	*Label*
1. If the DCB address contained in the parameter list is 0, (system log in use), no action is taken by this module. All storage associated with the LOG is released.	S@C2220	
2. The address of the storage obtained is contained in the free queue anchor in the SC_SSVT. CPOOL is used to release it.	S@C2220	
3. The parameter list indicates whether or not module S@C2210 allocated the LOG. The DDNAME should be obtained from the DCB, since a system-generated DDNAME may be used following ESTAE reprieve.	S@C2220	
4. The caller receives return and feedback codes indicating the processing performed.	S@C2220	

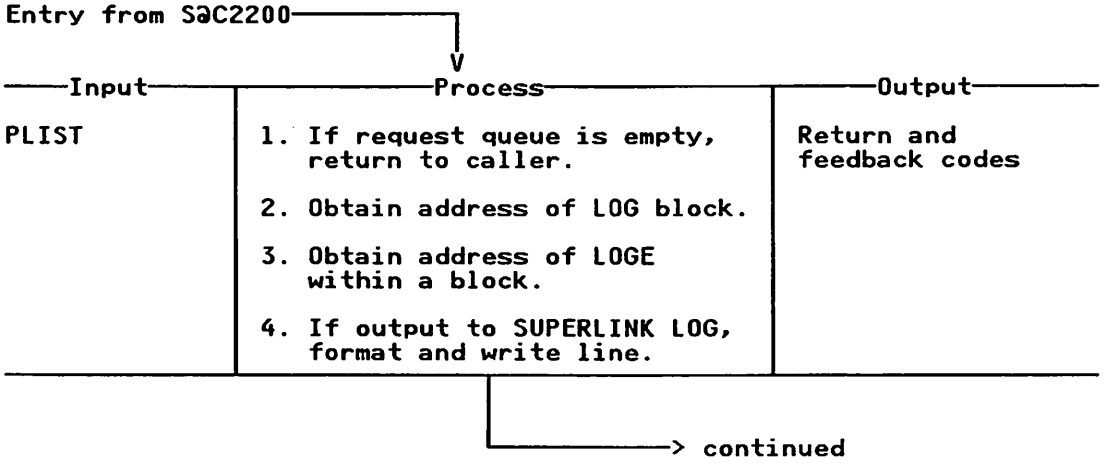
Diagram 6-6
S@C2230 - Output of Messages Subcomponent - Queue Swap and Reorder



Extended Description

Explanation	*Module*	*Label*
1. The queue anchors are held in the SC_SSVT as double words. A queue is empty if the anchor contains a value of 0's. The caller receives a warning return code if the work queue is empty. The caller receives an error return code and a message is written to the system log if the request queue is not empty, since module S@C2240 may not have processed the queue.	S@C2230	
2. The queues are swapped using a compare double and swap to ensure correct serialization.	S@C2230	
3. The work queue is a LIFO queue, and the request queue must be in FIFO order. This step is a loop to progress through the backward-chain pointers building forward-chain pointers to allow FIFO processing. See "Appendix A. Data Area Descriptions" on page A-1 for a description of the LOGE.	S@C2230	
4. The caller receives a return code indicating the processing performed.	S@C2230	

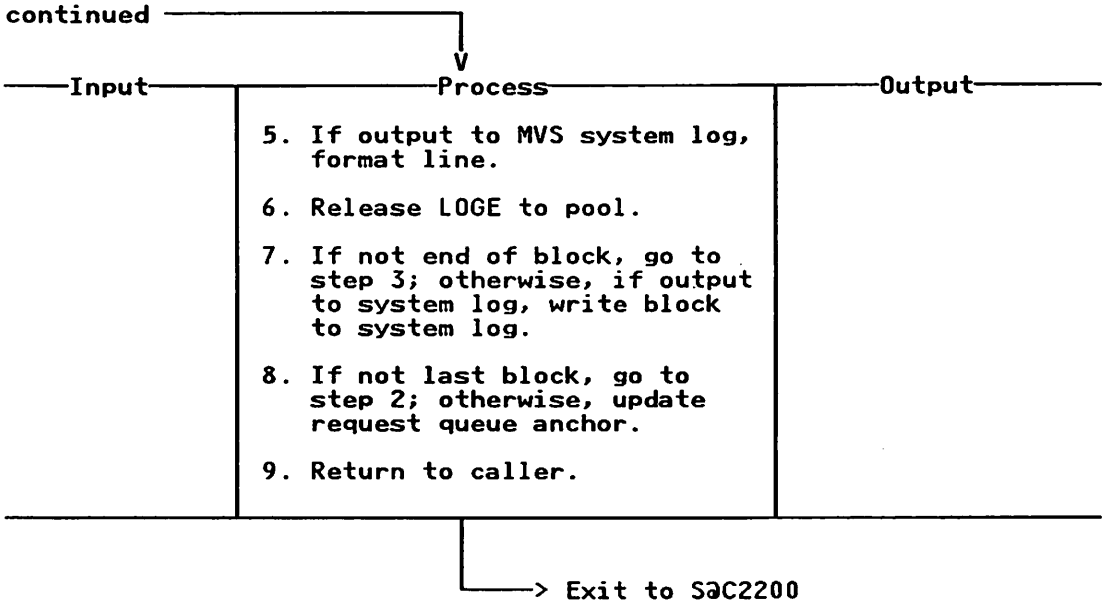
Diagram 6-7
S@C2240 - Output of Messages from LOGEs (part 1 of 2)



Extended Description

Explanation	*Module*	*Label*
1. A warning return code is given. Module S@C2240 is called following reprieve processing by the ESTAE. It should output the line in progress at the time of error to the new LOG (see steps 2 and 3).	S@C2240	
2. The request queue anchor block in the SC_SSVT contains the address of the first LOGE block. A LOGE block consists of one or more LOGEs chained by the user of the S@@LOG macro. For reprieve processing (for example, following an SB37 abend), the address of the next LOGE block is kept in the queue anchor head-of-queue pointer.	S@C2240	
3. Each LOGE contains the address of the next LOGE in a block; 0 indicates the last LOGE. The first LOGE address is the same as the LOGE Block address. For reprieve processing (for example, following an SB37 abend), the address of the current LOGE within a block is stored in the first fullword of the anchor block.	S@C2240	
4. The parameter list contains the address of the DCB (0 if using system log). This step is bypassed if the message length in the LOGE is 0. The LOGE indicates where the output should be routed (see macro S@@LOG found under "Appendix B. SLCN Macros" on page B-1). The print line is described by macro S@C2PLNE and is built by copying from the LOGE.	S@C2240	

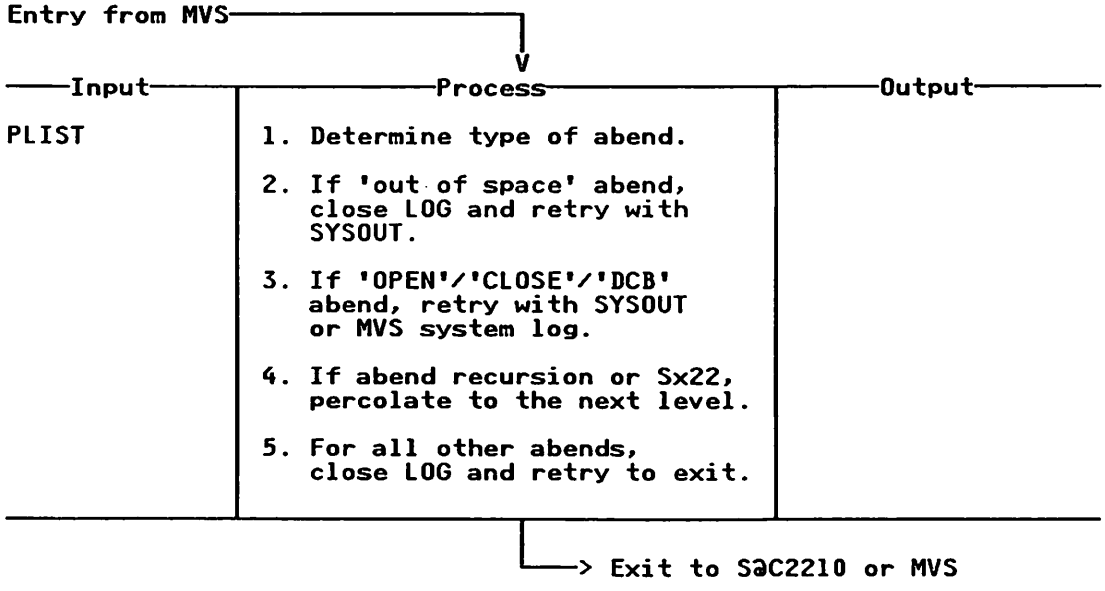
Diagram 6-8
S@C2240 - Output of Messages from LOGEs (part 2 of 2)



Extended Description

Explanation	*Module*	*Label*
5. The line for the system log output is the same as that for the SUPERLINK LOG except that the ASA character and time are not required.	S@C2240	
6. The LOGE just processed is returned to the pool using CPOOL. It then becomes available for use by issuers of macro S@@LOG.	S@C2240	
7. The chaining performed by macro S@@LOG ensures that message lines appear as an uninterrupted block. Thus, lines for the system log are collected at step 5 and output as a multiline WTO when the block is complete.	S@C2240	
8. The request queue anchor must show that the request queue is empty and is therefore set to 0's.	S@C2240	

Diagram 6-9
S@C2250 - Output of Messages Subcomponent - ESTAE Routine



Extended Description

Explanation	*Module*	*Label*
2. Typically, the abends are Sx37 abends. The LOG is closed by calling module S@C2220. The retry address is specified as S@C2210, and the PLIST specifies "reprieve." Module S@C2210 effects the LOG switch.	S@C2250	
3. Typically, the abends are Sx13/Sx14/S002 abends. The retry address is specified as module S@C2210, and the PLIST specifies "reprieve". Module S@C2210 effects the LOG switch.	S@C2250	
5. The LOG is closed by calling module S@C2220. The retry address is set so that module S@CC0000 receives return and feedback codes indicating that an abend occurred rather than the abend continuing.	S@C2250	

7. SUPERLINK Management Interface Component

Although SUPERLINK for MVS and SUPERLINK for COS must initialize independently, both MVS and COS management of the SUPERLINK/MVS product must be provided from a central point. In order to achieve this cooperation between the two systems, a permanent connection is established between the management functions on MVS (SLCN) and COS (SLMIR).

The Management Interface component, which takes the form of an OSI application, provides the permanent connection between SLCN and SLMIR. The component uses the services of the SUPERLINK Network Access Method (SLNET) to convey data between the management functions.

The Management Interface services enable two Management Interface components (typically one on MVS and one on COS) to interact and exchange "management" type information. This information typically includes commands and messages; messages may or may not be replies to commands.

Management Interface Module Structure

The Management Interface component consists of the following modules:

Module	*Function*
S@CI0000	Management Interface root module
S@CI0010	Management Interface initialization
S@CI0020	Management Interface termination
S@CI0030	Management Interface ESTAE
S@CI0040	Management Interface input queue server
S@CI0050	Management Interface connection protocol handler task
S@CI0060	Management Interface output queue server
S@CI0070	Management Interface connection protocol task ESTAE
S@CI0080	Management Interface protocol data unit (PDU) encoder/decoder

Figure 13 on page 7-2 shows the hierarchical structure of modules within the Management Interface component.

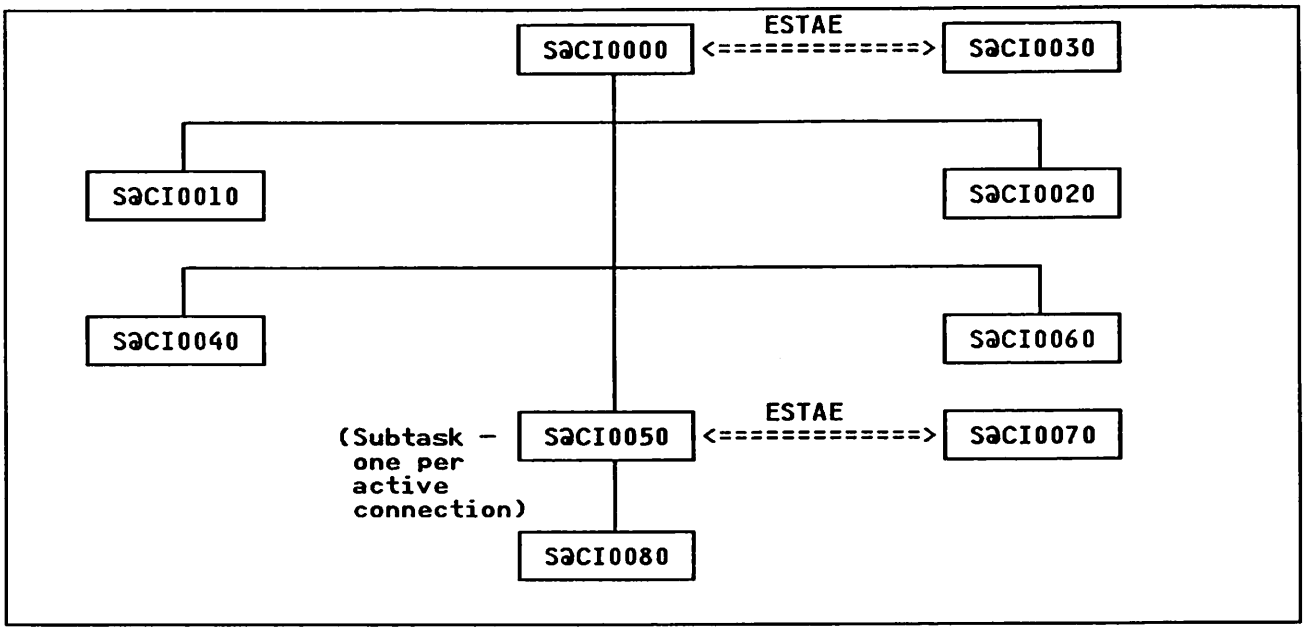


Figure 13. Module Structure of the Management Interface Component

Management Interface Services

In order to achieve cooperation between SLCN and SLMIR, a number of Management Interface component services have been defined as follows:

- Confirmed services - these services take the form of a two-way exchange of information between the participating entities
- Nonconfirmed services - these services take the form of a one-way transfer of information from one entity to the other
- Provider-initiated services - these services take the form of an indication presented by the service provider to the participating entities

The following subsections describe the basic service primitive types, Management Interface service primitives, and Management Interface local system primitives.

Service Primitive Types

Table 2 describes the service primitives that allow a service user to interact with a service provider.

Table 2. Basic Service Primitive Forms

Form	From	To	Function
REQUEST	Originating service user	Service provider	Activation of a particular service

Table 2. Basic Service Primitive Forms (continued)

Form	From	To	Function
INDICATION	Service provider	Service user	Provided at the destination end system to advise the service user of the activation of a particular service
RESPONSE	Service user	Service provider	Provided at the destination end system in response to an indication
CONFIRM	Service provider	Requesting service user	Complete a confirmed service

Management Interface Service Primitives

The following primitives are used to establish a Management Interface connection and to exchange data over that connection:

- MI-LOGON
- MI-LOGOFF
- MI-ABORT
- MI-P-ABORT
- MI-COMMAND
- MI-MESSAGE

The SUPERLINK Protocol Information Manual, publication SI-0175, provides more detailed information about these primitives.

Management Interface Local System Primitives

The following local system primitives are supported. The service user implements them to control the service being used.

- MI-LOGON-OFFER
- MI-RECEIVE

Management Interface Component Interfaces

The Management Interface component is required to convey configuration details between the COS and MVS Product Management components. The major source of input to this interface is from an operator's console. The types of commands and command responses flowing across this connection are typically a subset of the COS DSPL*xx* commands.

All commands processed by this mechanism are considered to be MVS Master Console-only requests, requiring the coordinated control that is achieved through the connection between SUPERLINK for MVS and SUPERLINK for COS.

Management Interface Data Areas

The Management Interface component has the following main data areas:

Data Areas	*Description*
MI_MACB	Management Interface connection manager control block Each Management Interface connection defined in the SUPERLINK/MVS initialization options statements is allocated a MI_MACB. This control block monitors the state of its associated connection for the entire session.
MI_MICT	Management Interface control table This is the major control block of the Management Interface task itself. The MI_MICT is anchored from the SC_SSVT and monitors information pertaining to the Management Interface component.
MI_MRQE	Management Interface request element Work from other tasks or the operator is queued to the Management Interface task by a work-to-do queue of MI_MRQEs. The elements are dequeued and processed one at a time, in FIFO order.

"Appendix A. Data Area Descriptions" on page A-1 provides descriptions of these data areas.

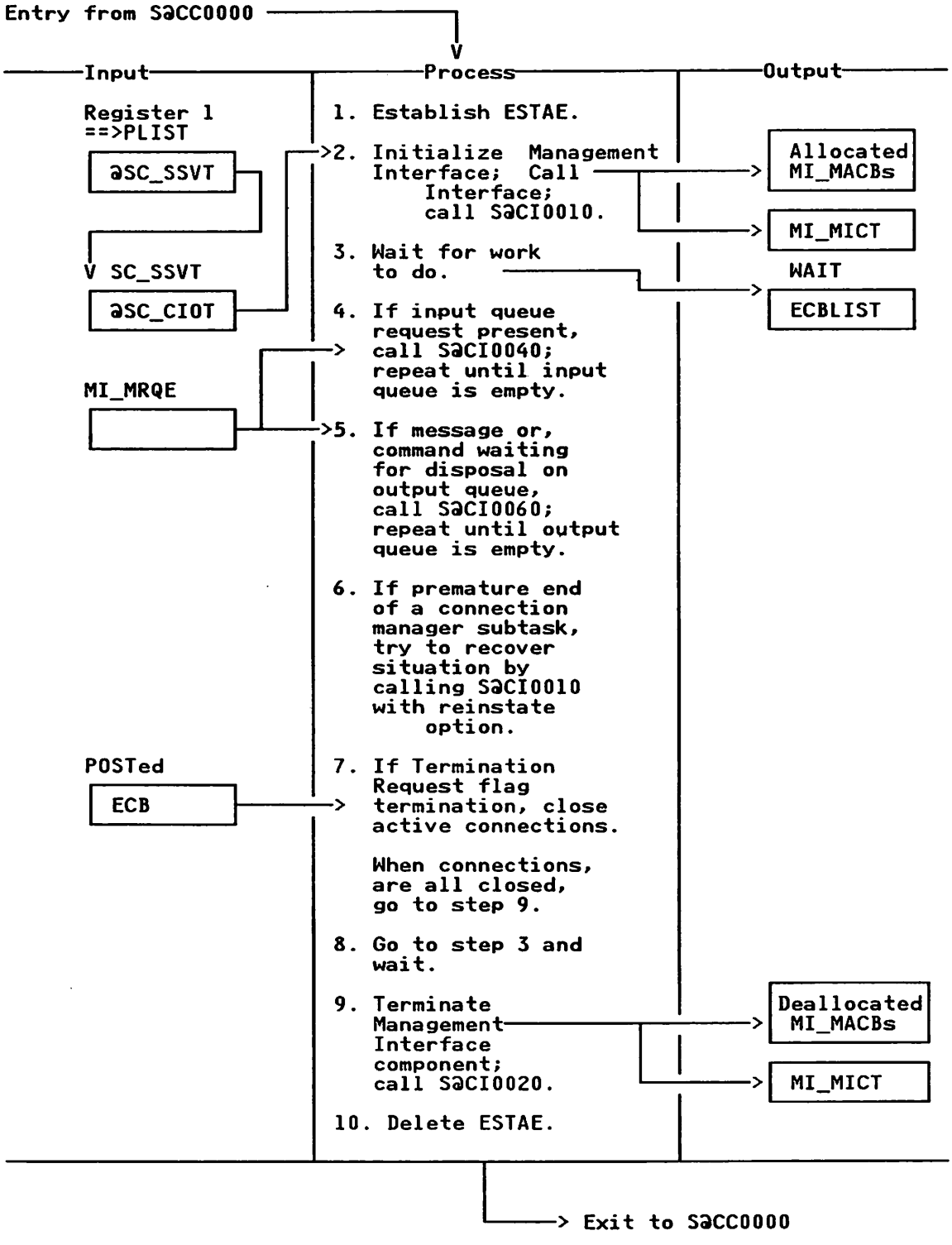
Management Interface Recovery

The Management Interface component runs as a subtask of the main root module of SLCN (S@CC0000). The root module of the Management Interface component (S@CI0000) enables an ESTAE environment to trap abends within the Management Interface subtask and attempts to recover from them whenever possible. If recovery is not possible, notification of the abend is percolated to the next level of the recovery environment.

Each Management Interface connection handler subtask is protected by an ESTAE environment which attempts to recover the connection. The SUPERLINK/MVS configuration options statements determine whether automatic restart of a connection is possible or operator intervention is necessary.

This page has been intentionally left blank.

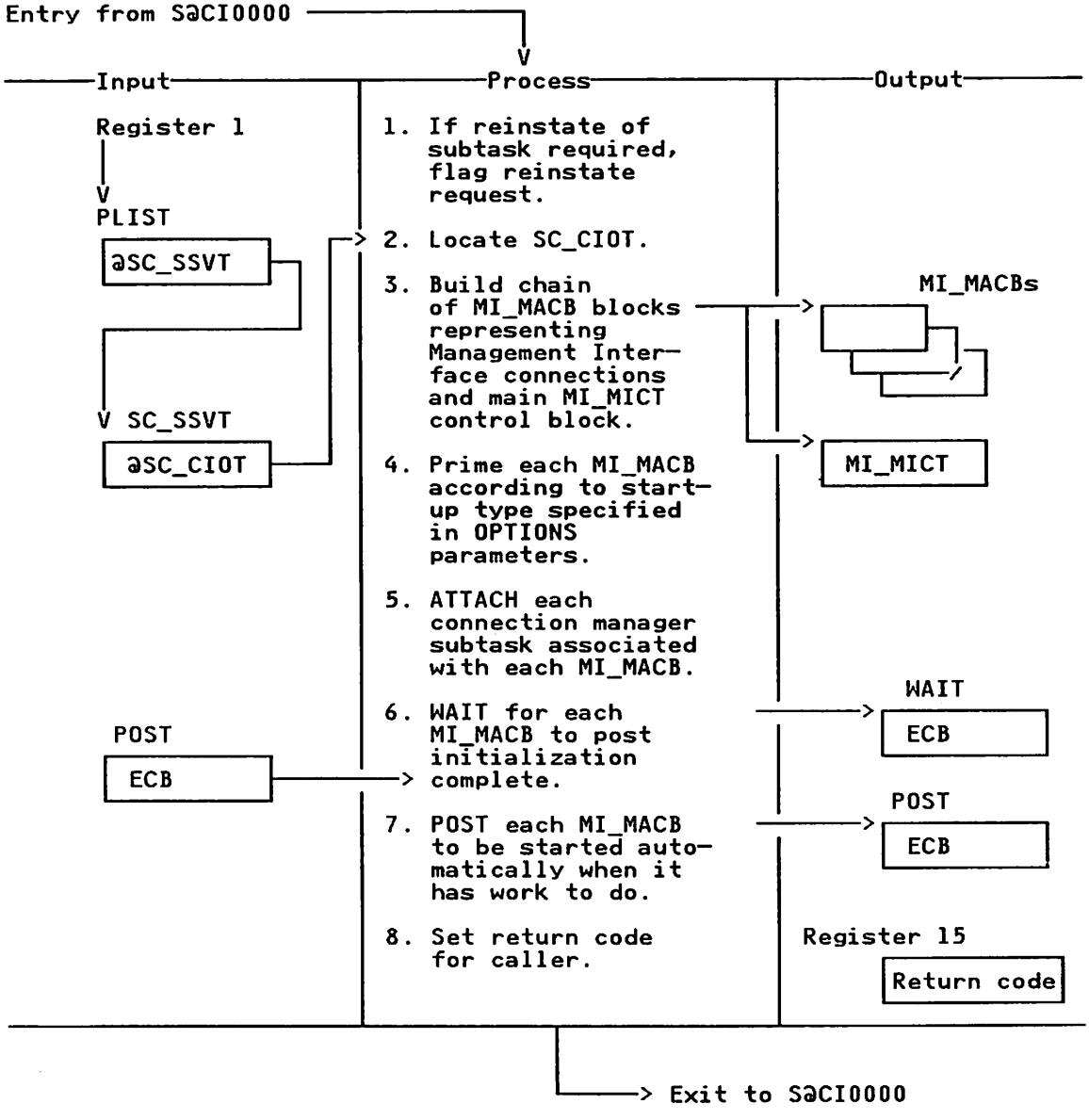
Diagram 7-1
S@CI0000 - Root Module of Management Interface Component



Extended Description

Explanation	*Module*	*Label*
1. The ESTAE routine is S@CI0030.	S@CI0000	
2. All the resources required to run the Management Interface component are obtained during this step. Particulars about the number and characteristics of the necessary management connections are defined in the SUPERLINK/MVS startup processor. These definitions are processed by the Options Processor and formatted into the SC_CIOT control block. Initialization uses this data to format the MI_MICT and MI_MACB control blocks. All the defined subtasks which are handled by the Management Interface connection are ATTACHed by routine S@CI0010. If initialization completes successfully, POST the main control task in SLCN; otherwise, branch to step 9 and return to the caller.	S@CI0000	
3. The Management Interface component waits for work-to-do requests to arrive from various sources (input queue, output queue, terminate request POSTed, or subtask termination).	S@CI0000	
4. Requests from the MVS operator, or other tasks or components of this task itself which require use of the Management Interface, arrive as MI_MRQEs on a queue hung from the SC_SSVT. If any requests are present on this queue, they are removed and processed, one at a time, by routine S@CI0040, until the queue is empty. These requests may be operator commands or messages to be sent on a particular management connection.	S@CI0000	
5. Messages and commands received on Management Interface connections must be disposed to their correct destinations (for example, messages to the SUPERLINK LOG or to the operator's console and commands to the correct component of SUPERLINK/MVS or MVS for execution). S@CI0060 performs this function on data that has been received and queued onto the output queue by instances of S@CI0050. S@CI0060 dequeues and processes each element until the queue is empty.	S@CI0000	
6. The premature termination of a connection manager subtask is detected here, and a decision is made to reinstate the task, to terminate, or to notify the operator for action.	S@CI0000	
7. If the main control task in SLCN has POSTed the Management Interface component to terminate ECB, this task cleans up and exits. This is done by determining the type of termination requested (NORMAL/QUICK/ABORT) and flagging all connections to terminate in that manner. Once all connections have terminated correctly, the routine branches to step 9 and exits.	S@CI0000	
9. The Management Interface component cleans up by freeing all the resouces it has obtained. Each connection manager subtask is DETACHed, having completed its own termination.	S@CI0000	

Diagram 7-2
S@CI0010 - Management Interface Initialization Routine



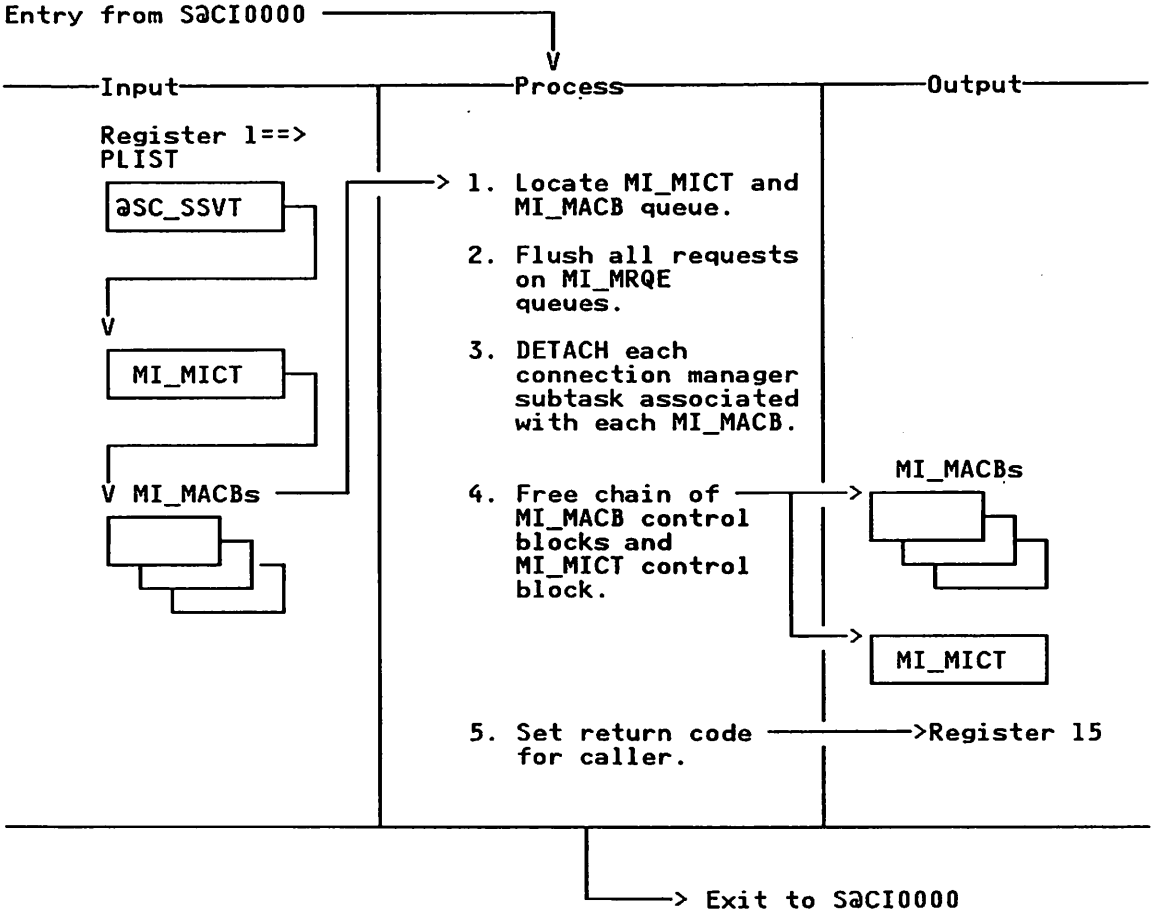
Extended Description

Explanation	*Module*	*Label*
1. A reinstate request is made after a connection manager subtask has failed and is being reinitialized. Thus the logic of this module is the same as that of S@CI0000, except that the MI_MICT and the MI_MACBs are already present, and only one connection manager subtask must be initialized.	S@CI0010	
2. The SC_CIOT is chained from the SC_SSVT.	S@CI0010	
3. The SUPERLINK/MVS configuration OPTIONS statements state how many Management Interface connections are defined and indicate the characteristics of each one. One MI_MACB per defined connection and one MI_MICT for the whole task are allocated. Configuration options indicate: whether the connection is to be started automatically or with an operator command; whether automatic restart is to take place on a failed connection or whether the operator initiates the restart sequence; and whether the connection is to come up as a primary end (issues MI-LOGON requests) or a secondary end (issues MI-LOGON-OFFER requests).	S@CI0010	
4. Each connection to be started automatically has a request queued to its MI_MACB indicating whether it is to start as a primary or a secondary end.	S@CI0010	
5. Each MI_MACB represents a Management Interface connection, and each is managed by an instance of the connection manager subtask (S@CI0050). Each connection defined by an MI_MACB has an instance of this task ATTACHed, and is passed the address of its own MI_MACB. End-of-task should be notified by specifying a common exit routine instead of using a list of ECBs, which proves more difficult to manage.	S@CI0010	
6. Each connection manager subtask must initialize itself. Processing must WAIT until each has done so.	S@CI0010	
7. Those subtasks to be automatically started are POSTed. The ECB to be POSTed is the same one the subtask uses as its "external cancel" ECB during A-RECEIVE processing; the subtasks also WAIT on this after initialization and before they start to perform any work. POSTing indicates that the task's MI_MACB has work to do. When dispatched, they find the MI_MRQE request element on the work queue and perform the initial action required, for example, MI-LOGON request or MI-LOGON-OFFER request (for testing).	S@CI0010	
8. A return code for caller is set indicating the state of initialization as follows: **00** Initialization was successful. **04** Initialization failure; able to proceed with reduced functionality. **08** Initialization failure; unable to proceed.	S@CI0010	

If reinstate of a subtask was requested, the return codes are as follows:

00 Reinstate was successful.
04 Reinstate failed.

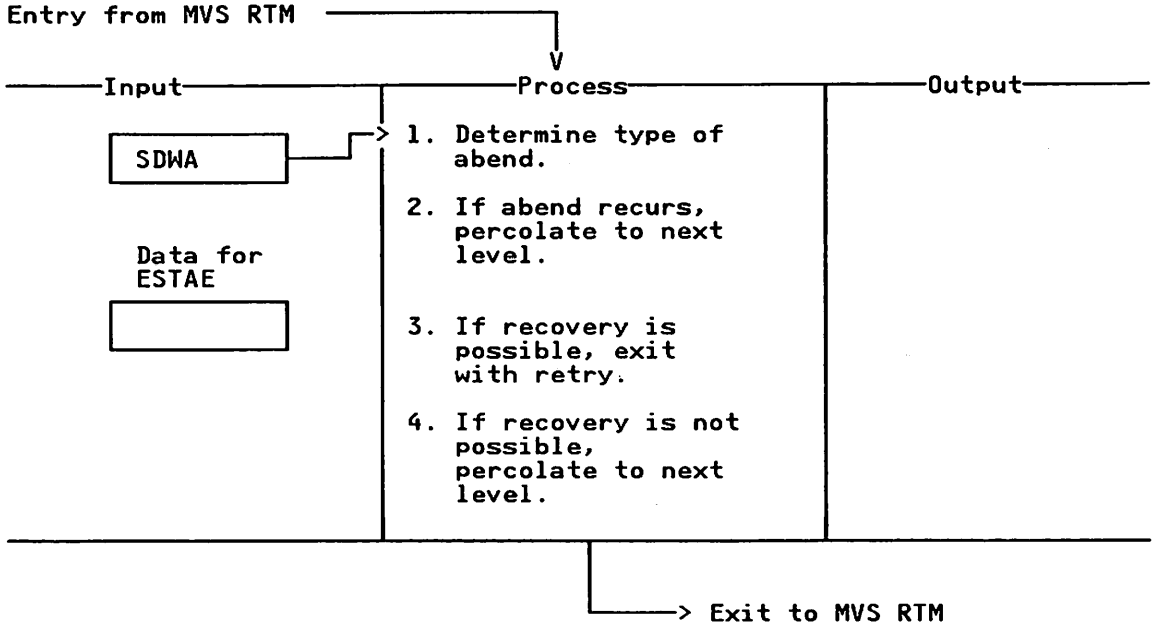
Diagram 7-3
S@CI0020 - Management Interface Termination Routine



Extended Description

Explanation	*Module*	*Label*
1. The MI_MICT and the MI_MACB queues are anchored from the SC_SSVT.	S@SCI0020	
2. Both input and output M_MRQE queues have all their outstanding MI_MRQE elements flushed (dequeued and FREEMAINed).	S@CI0020	
3. Each MI_MACB that has an ATTACHed connection manager subtask has that subtask DETACHed. The subtask will have already terminated due to processing performed when a termination request was recognized in the S@CI0000 module.	S@SCI0020	
5. The return codes sent to the caller are defined as follows: **00** Termination was successfully completed. **04** Termination processing encountered some error.	S@CI0020	

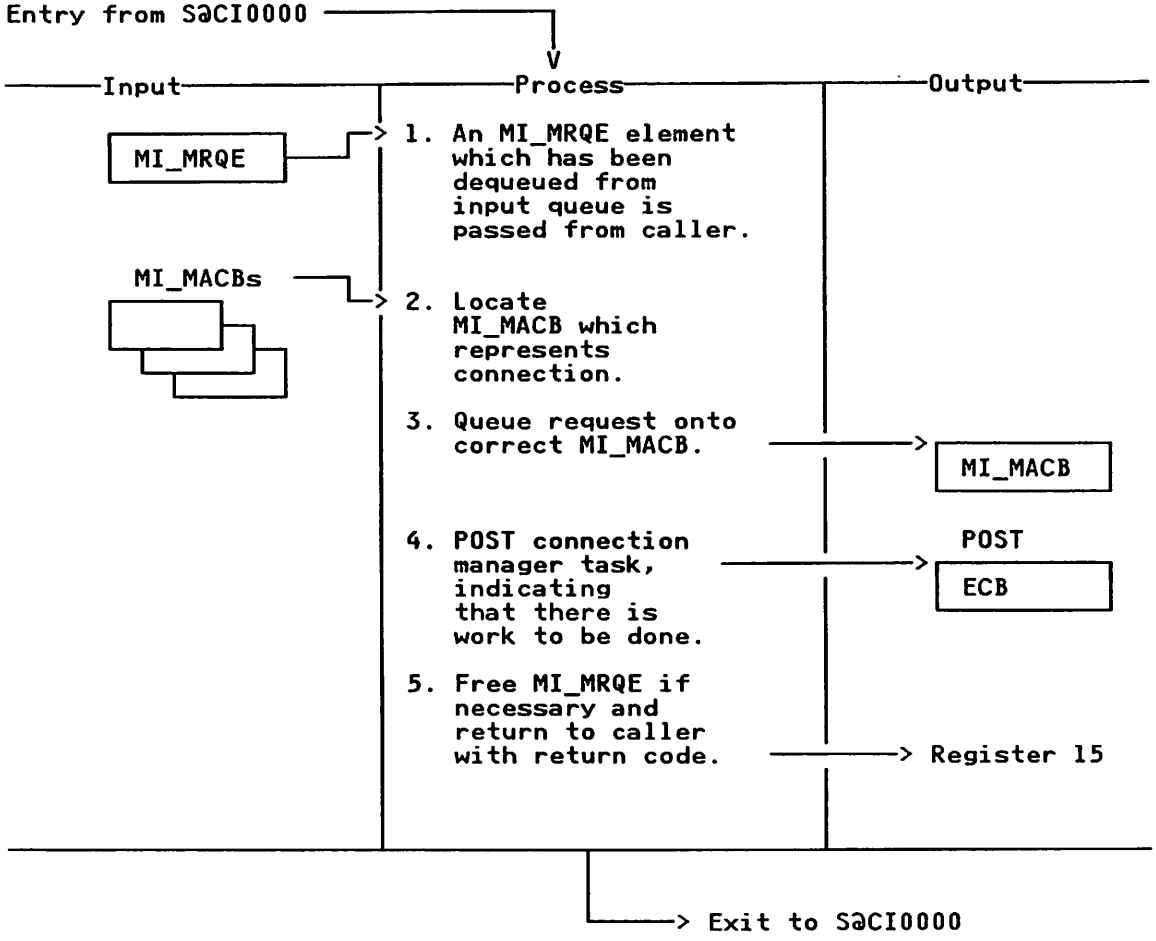
Diagram 7-4
S@CI0030 - Management Interface ESTAE Routine



Extended Description

Explanation	*Module*	*Label*
1. No retry is attempted on X22-type abends, nor is a dump obtained.	S@CI0030	
2. To prevent reentry in an abend recursive loop, check for recursion. If this is a recursive abend, control should simply be percolated to the next level of recovery, where the SLCN root module detects the termination of the Management Interface component.	S@CI0030	
3. If recovery is possible for the type of abend encountered, a retry routine is specified on the SETRP macro (if an SDWA is present). This routine gains control after the the ESTAE has returned control to RTM. Use the SDUMP routine to obtain a dump for diagnostic purposes.	S@CI0030	
4. If recovery is not possible, use the SDUMP routine to obtain a dump for diagnostic purposes. Using the SETRP macro (if SDWA is present) again, percolate the abend condition to the next level of recovery, where the SLCN root module detects the termination of this task.	S@CI0030	

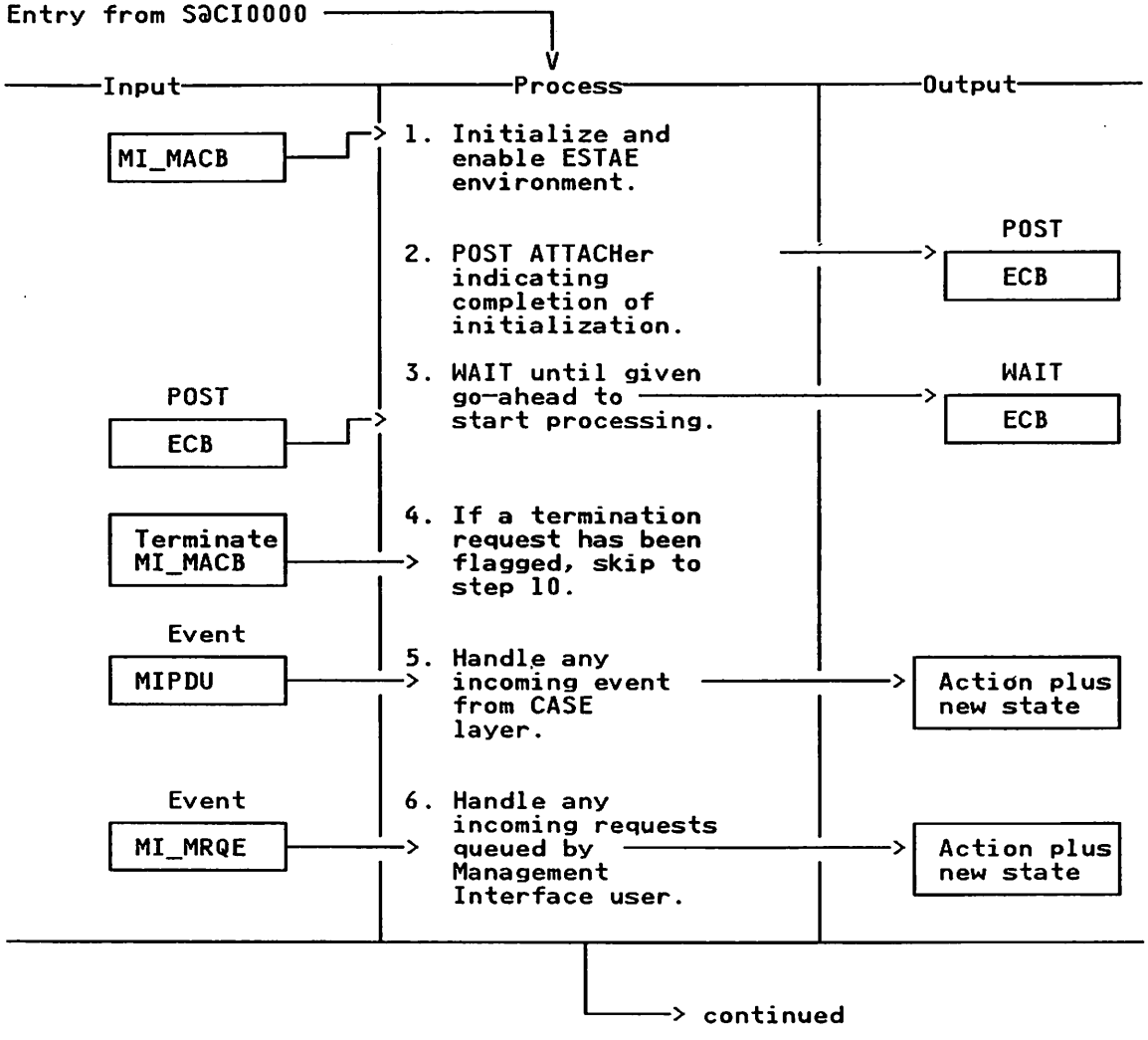
Diagram 7-5
S@CI0040 - Management Interface Input Queue Server



Extended Description

Explanation	*Module*	*Label*
1. MI_MRQE elements queued may be destined for a remote management component via MI-COMMAND or MI-MESSAGE requests, or for action on the local system, such as a command to stop or start a particular connection.	S@CI0040	
5. If an MI_MRQE request element is to be freed, free it; otherwise, do nothing. The return code set for the caller is as follows: **00** Request processed successfully; MI_MRQE not freed. **04** Request processed successfully; MI_MRQE freed. **08** Request not processed successfully; MI_MRQE freed.	S@CI0040	

Diagram 7-6
S@CI0050 - Management Interface Protocol Event Routine (part 1 of 2)



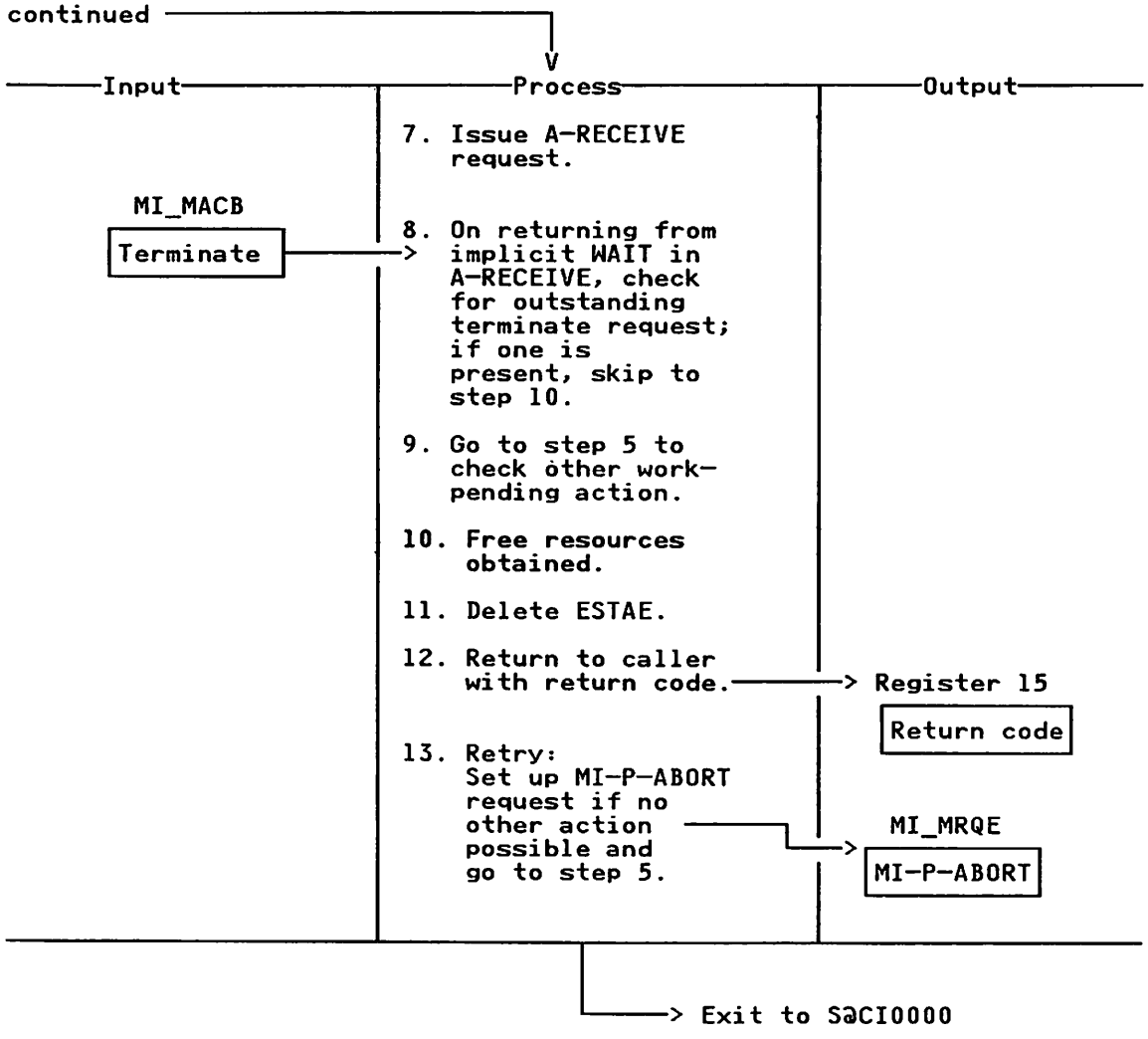
Extended Description

Explanation	*Module*	*Label*
1. This module must be reentrant, because there are multiple instances of it (one per active connection manager subtask). Enable the ESTAE environment (module S@CI0070) and initialize.	S@CI0050	
2. Once initialization has completed successfully, POST S@CI0000 that this subtask has correctly initialized.	S@CI0050	
3. WAIT for the ECB that will also be used as the A-RECEIVE "external cancel" ECB for S@CI0000 to indicate that the work-to-do queue may now be processed.	S@CI0050	
4. If a Terminate Request flag was set in MI_MACB, go to step 10.	S@CI0050	
5. Handle any incoming events, using the State/Event table to determine the appropriate action.	S@CI0050	
6. Handle any requests queued via MI_MRQE elements, using the state/event table to determine the appropriate action.	S@CI0050	

```
If (A-RECEIVE data pending) then do
extract-event-just-received;
perform-action(current-state,received-event);
end-if
```

```
While (request queue is not empty)
dequeue-MI_MRQE-event;
perform-action(current-state,dequeued-event);
end-while
```

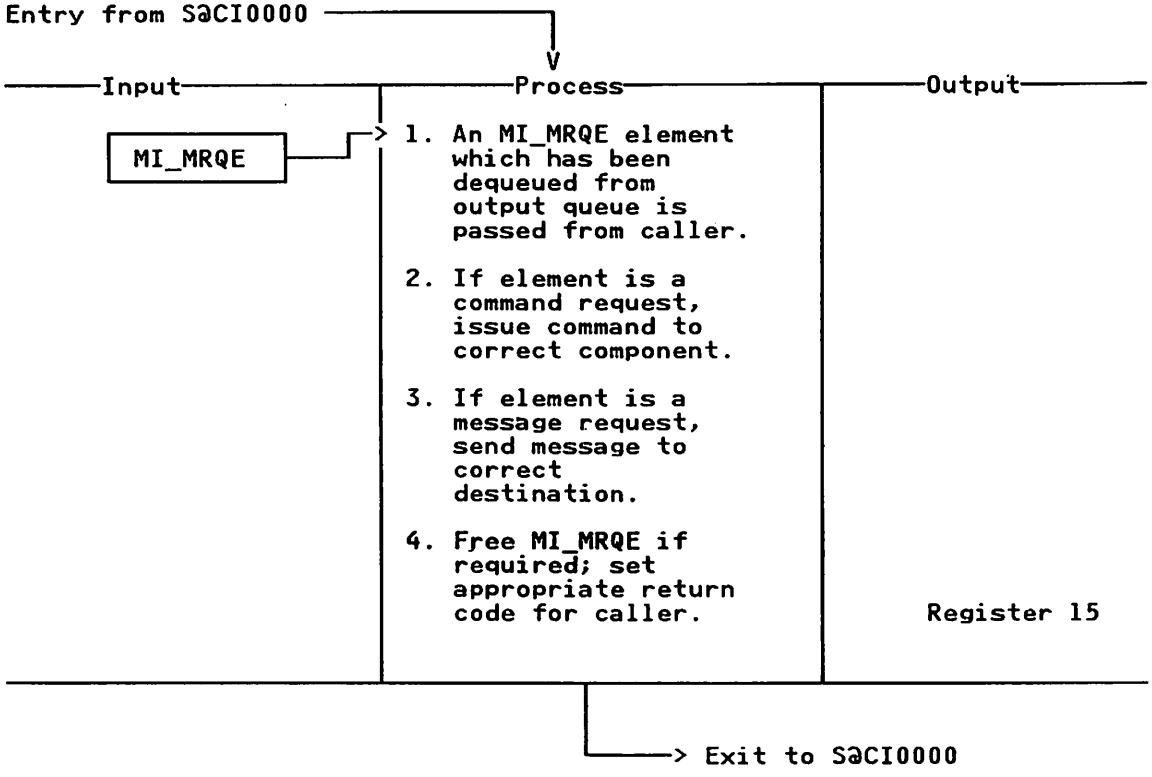
Diagram 7-7
S@CI0050 - Management Interface Protocol Event Routine (part 2 of 2)



Extended Description

Explanation	*Module*	*Label*
7. Issue an A-RECEIVE request (implicit WAIT on "external cancel" ECB).	S@CI0050	
8. When data is received on the outstanding A-RECEIVE or the "external cancel" ECB is POSTed (indicating that the queue contains work to do), check whether a termination request has been made. If the Termination Request flag is set in the MI_MACB, go to step 10.	S@CI0050	
9. Check what sort of work is to be done by going to step 5.	S@CI0050	
10. Clean up any resources obtained before terminating.	S@CI0050	
11. Delete the ESTAE environment.	S@CI0050	
12. Return to caller with return code indicating success of termination.	S@CI0050	
13. On retry processing from the ESTAE routine (S@CI0070), this routine is entered at this point. If there is no way to recover the existing connection, an MI-P-ABORT request is placed on the work queue of this connection manager subtask and branches to step 5 so the request can be acted upon according to the State/Event table.	S@CI0050	

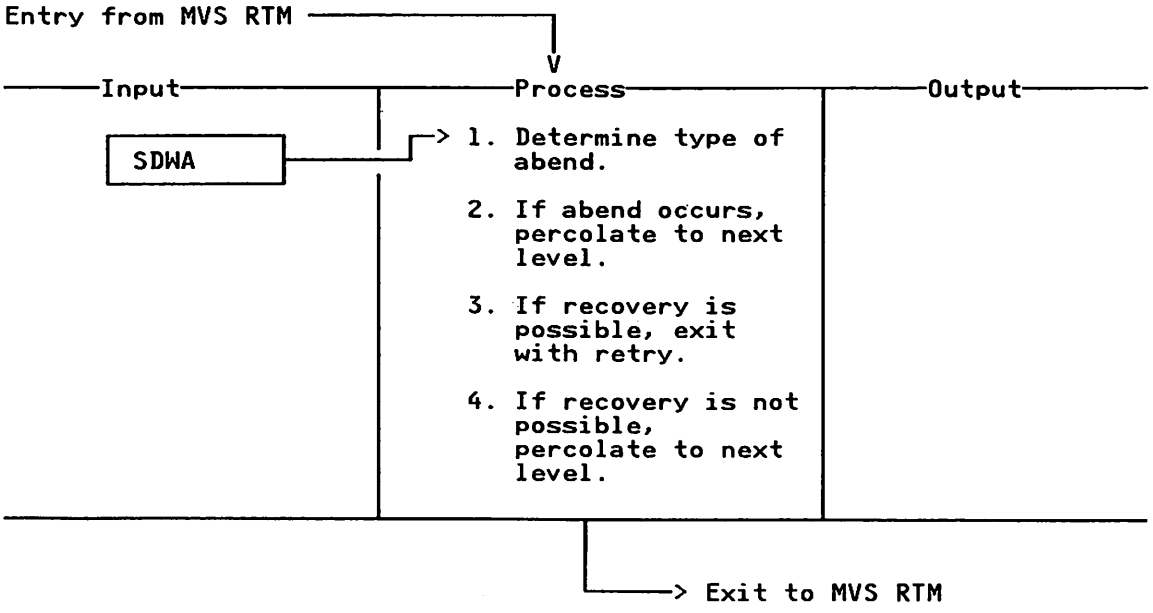
Diagram 7-8
S@CI0060 - Management Interface Output Queue Server



Extended Description

Explanation	*Module*	*Label*
1. These MI_MRQE elements may have originated from incoming MI-COMMAND or MI-MESSAGE indications within a connection manager subtask.	S@CI0060	
4. If it is necessary to free the MI_MRQE element, do so; otherwise, leave it as it is. Set a return code for the caller as follows: **00** Request processed successfully; MI_MRQE not freed. **04** Request processed successfully; MI_MRQE freed. **08** Request not processed successfully; MI_MRQE freed.	S@CI0060	

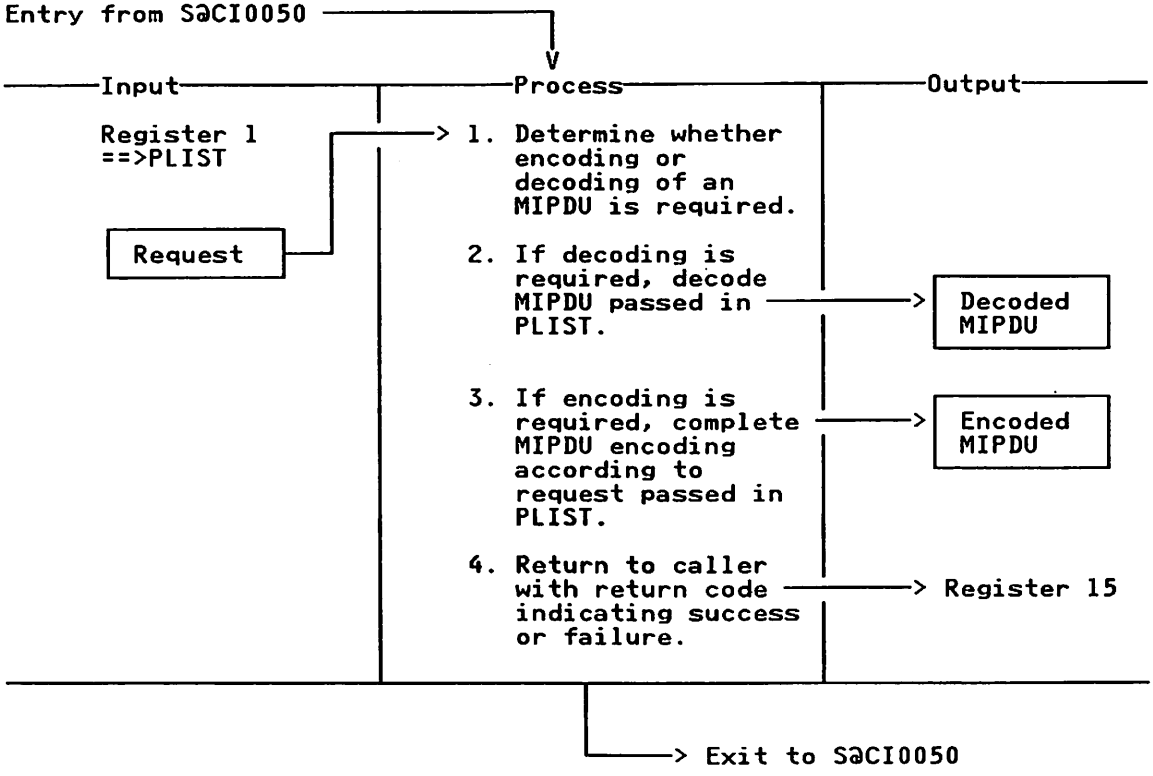
Diagram 7-9
S@CI0070 - Management Interface Connection Protocol Task ESTAE



Extended Description

Explanation	*Module*	*Label*
1. This routine must be reentrant, since multiple instances (one per Management Interface connection) exist at one time. No retry is attempted on X22-type abends, nor is a dump obtained.	S@CI0070	
2. To prevent reentry in an abend recursive loop, check for recursion. If this is a recursive abend, control should be percolated to the next level of recovery, where the Management Interface root control module (S@CI0000) detects the termination of the connection management subtask.	S@CI0070	
3. If recovery is possible for the type of abend encountered, a retry routine is specified on the SETRP macro (if an SDWA is present). This routine gains control after the the ESTAE has returned control to RTM. The recovery routine resides in the S@CI0050 module. A dump is not obtained at this level; however, a diagnostic message indicating the source of the problem is written to the SUPERLINK LOG.	S@CI0070	
4. If recovery is not possible, the abend condition is percolated to the next level of recovery using the SETRP macro again (if an SDWA is present). The Management Interface root control module (S@CI0000) detects the termination of this task. A diagnostic message indicating the source of the problem is written to the SUPERLINK LOG.	S@CI0070	

Diagram 7-10
S@CI0080 - Management Interface PDU Encoder/Decoder



Extended Description

Explanation	*Module*	*Label*
2. Decoding of PDUs is performed according to the PDU ASN.1 definitions and using the ASN.1 Basic Encoding Rules.	S@CI0080	
3. Encoding of PDUs is performed according to the PDU ASN.1 definitions and using the ASN.1 Basic Encoding Rules.	S@CI0080	

8. SUPERLINK Association Manager Component

The function of the SUPERLINK/MVS Association Manager is to issue A-OFFER service request primitives for all specified application titles (using the ATITLE statement within Options) and to process all incoming A-ASSOCIATE indications for those application titles. The SUPERLINK/MVS Installation, Tuning, and Customization Guide, publication SI-0180, describes the ATITLE statement.

Association Manager Subcomponents

The Association Manager component consists of the following subcomponents, which are described later in this subsection:

Subcomponent	*Description*
S@C9000	Association Manager controller (AM_controller)
S@C9100	Association Manager processor (AM_processor)
S@C9200	Association Manager interface (AM_interface)
S@C9300	Association Manager timer services (AM_timer)
S@C9UXAM	Association Manager User Exit Handler (AM_exits)

Association Manager Subcomponent Flow

Figure 14 on page 8-2 shows the subcomponent flow for the Association Manager component.

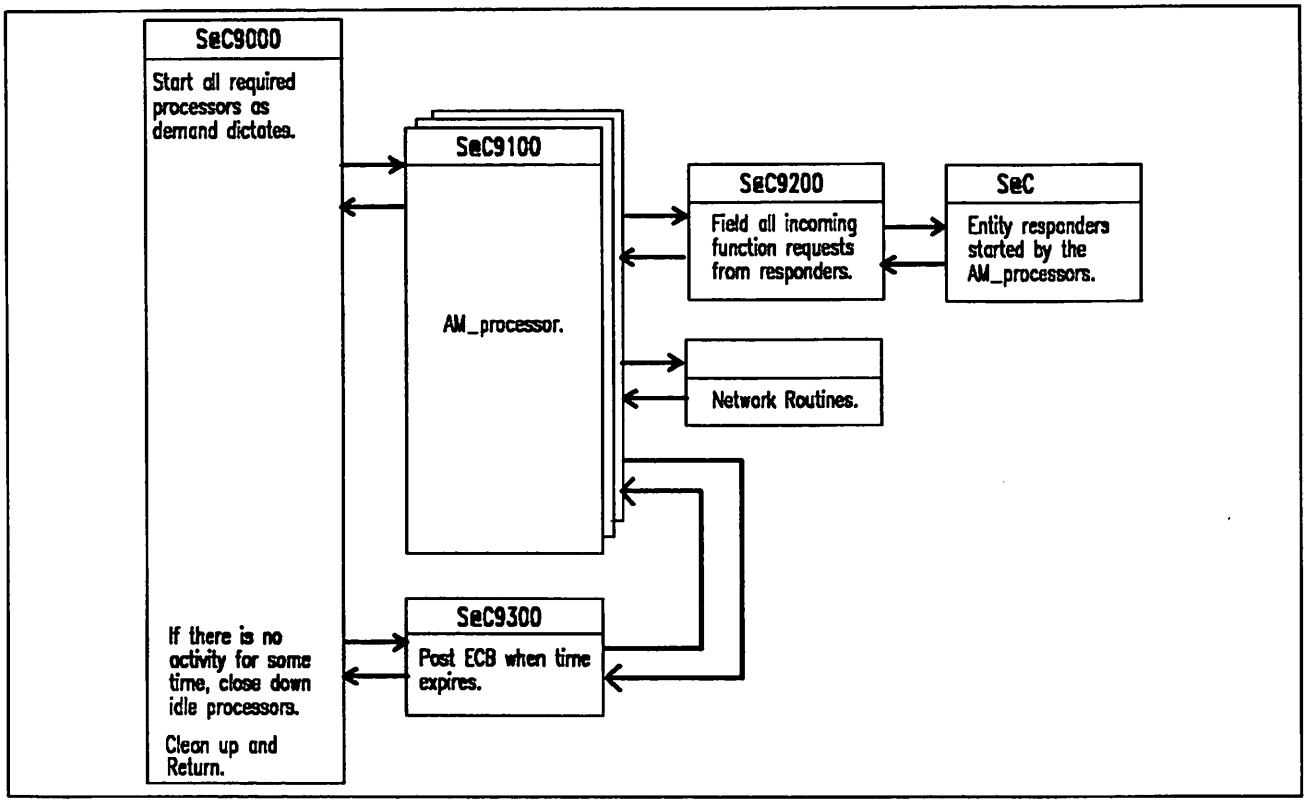


Figure 14. Association Manager Component Flow of Control

Association Manager Services

The Association Manager component provides the following services:

- Issues an A-OFFER service request primitive for each registered application title defined by the ATITLE statement within the SUPERLINK options. As soon as an inbound connection establishment request is delivered to the Association Manager, another A-OFFER service request primitive is issued.
- Passes connection end points to a responder (if a responder that can take the end point is active) upon receipt of an incoming A-ASSOCIATE indication
- Initiates a responder to take connection end points (if no responder is currently active) upon receipt of an incoming A-ASSOCIATE indication. Once the initiated responder becomes active, the connection end point is transferred to the responder.
- Initiates copies of responders that become congested and passes the work still to be processed to the new responders.
- Monitors for premature termination of responders. Any unprocessed connection end points are either rejected or passed to another responder for processing.

Association Manager Services Offered to Application Entity Initiators on COS

When the Association Manager receives an incoming A-ASSOCIATE indication from an initiator on COS, it attempts to match it to any responder that is already active. The Association Manager uses the following matching criteria for &AMTYPE=MULTIPLE:

- MVS user ID
- MVS group ID
- MVS accounting information
- MVS user ID password

The preceding criteria are supplied from the IDENT, AUTH, or TEXT fields within the incoming A-ASSOCIATE indication. If these fields are not specified, the criteria is supplied by default from the SUPERLINK/MVS Options.

If the default, &AMTYPE=SINGLE, is specified, the following information is included as part of the matching criteria:

- PSAP ID (COS job name and JSQ)
- Mainframe ID of the A-ASSOCIATE indication

Once a determination of a match has been made, one of the following occurs:

- If the responder is active and has registered itself with the Association Manager (by performing a LISTEN call using the Association Manager interface via the S@@MREQ macro), the connection end point is given to the responder and the Association Identifier (AID) is added to the responder's queue.
- If the responder has been submitted but has not yet performed a LISTEN call, the end point is held until the responder is ready. The AID is added to the responder's queue.

When a match is not made, the Association Manager initiates a new responder. SUPERLINK/MVS options and a series of variables are used to complete a JCL template. This JCL is used to submit the responder companion job. Any variables referred to within the JCL either use default values from the SUPERLINK options, or override these defaults using values from within the A-ASSOCIATE indication TEXT, IDENT, and AUTH fields. The variables include REGION, CLASS, PROC, and so on. For further details, refer to "S@C9100 - Association Manager Processor Subcomponent" on page 8-18. The SUPERLINK/MVS Installation, Tuning, and Customization Guide, publication SI-0180, provides more detail about specifying options.

Association Manager component processing is controlled by parameters supplied by the COS application entity initiator.

The following are the parameters in the incoming A-ASSOCIATE indication:

Parameter	*Description*
AUTH1	Level one authorization information (MVS password); overrides default and TEXT values.
AUTH2	Level two authorization information; not used by Association Manager component but is passed to AM User exit 4 (Security Validation)
AUTH3	Level three authorization information; not used by Association Manager component but is passed to AM User exit 4 (Security Validation)
IDEN1	Level one identification information (MVS user ID); overrides default and TEXT values.

Parameter	*Description*
IDEN2	Level two identification information (MVS accounting information); overrides default and TEXT values.
IDEN3	Level three identification information (MVS group ID); overrides default and TEXT values.
TEXT	Values for keyword variables; overrides all default values.
MF ID	Mainframe identifier (used only for &AMTYPE = SINGLE).
PSAP ID	Presentation service access point (PSAP) identifier (used only for &AMTYPE = SINGLE).

Association Manager Services Offered to Application Entity Responders on MVS

All responders initiated by the Association Manager can use the Association Manager Interface via the S@@MREQ macro. Using the interface allows them to do the following:

- LISTEN - The responder is active and ready to receive work.
- NOTIFY - The responder wants to be notified of an event via the POST mechanism.
- PROCESS - The responder wants details of an event, if there is one.
- DELETE-EP - The responder is not receptive to subsequent events, with the exception of termination requests.
- DELETE-ANY - The responder is not receptive to any events.

The following inbound or outbound events are valid:

- Connection end point given
- Termination order, defined as follows:
  - Graceful - All outstanding functions are completed, but new requests for SUPERLINK/MVS services are denied prior to termination processing.
  - Quick - All pending activity is flushed, then orderly termination of all active functions is performed.
  - Abort - All active functions are abruptly terminated.

Association Manager Interfaces

The following subsections describe interfacing to the Network Access Method, interfacing to user exits, interfacing to application entity responders, and the queue management facility for connection end points.

Interfacing to the Network Access Method

The Association Manager component uses ACSE as an interface to the Network Access Method.

Interfacing to User Exits

The Association Manager user exit handler, (S@C9UXAM), is invoked whenever a user exit is to be invoked. This module establishes the environment required for the user exit and also sets up the parameters that the user exit requires. The Association Manager user exit handler may be called from anywhere within the Association Manager component by using the S@C9EXIT macro.

Interfacing to Application Entity Responders

Application entity responders interface with the Association Manager component by issuing the S@@MREQ macro. The expansion of this macro results in a call to the AM_interface subcomponent (S@C9200), a reentrant module located in common storage.

The Queue Management Facility for Connection End Points

A queue management facility is used for transferring connection end points from the Association Manager to initiated responders via the AM_interface subcomponent. The items to be queued are AIDs. The Association Manager issues an A-ENDPOINT-GIVE service request primitive and queues the AID for a responder. When a responder becomes active, it calls the S@@MREQ macro and is given the AID, which is then used to take the connection by issuing an A-ENDPOINT-TAKE service request primitive.

The facility consists of the following two queues for each responder:

- Request queue (RQ); AM_processors add AIDs for connections to the queue as required.
- Work queue (WQ); when a responder is ready to accept work, the interface routine switches items from the request queue to the work queue and then extracts items from the queue as the responder requests them.

Each queue contains the following items:

- A queue anchor
- Queue elements (chained together)

Queue elements are processed in FIFO order. Therefore, the work queue is double-headed, double-threaded, and noncircular. To add or remove a queue element, the following operations must be performed in one action:

- Chain or dechain a queue element
- Update the double-headed queue anchor

To perform this action, the two-queue method has been adopted. The queues are organized as follows:

- A request queue is organized in LIFO order
- A work queue is organized in FIFO order

When adding an item, a queue element is added to the end of the request queue. When removing an item, the item is taken from the head of the work queue. If the work queue is empty, it is switched with the request queue. This is done with a "compare double and swap" instruction within a loop to ensure serialization. Queue elements are reordered from LIFO to FIFO by double-chaining the work queue prior to its processing.

This queue management facility provides the following features:

- No lock/unlock, eliminating possible contention
- No "bit-spin" loop
- Any number of tasks can simultaneously add elements to a single request queue.

All queue elements are aligned on a double-word boundary. Each queue has an associated anchor which is also aligned on a double-word boundary. The backward chaining is established as elements are added to the request queue and the forward chaining is established after a request queue has been switched to a work queue.

The following macros are used for handling the queues:

Macro	*Description*
S@C@QADD	Add an item to the request queue. This macro expansion obtains a queue element from a cell pool, inserts the data into the element, and chains it in the specified request queue.
S@C@QREM	Remove a queue element from the head of the work queue. This macro expansion dechains a work queue element, copies its contents to the specified target area, and returns the queue element to the cell pool.
S@C@QSWI	Switch a request queue to an empty work queue. This macro expansion switches an empty work queue with a request queue.

"Appendix B. SLCN Macros" on page B-1 provides a description of these macros.

Association Manager User Exits

A number of user exits are provided within the Association Manager to allow installations to perform validation and "local" processing. The Association Manager component has the following four installation exits:

- Variable validation installation exit
- Validation of JCL for submission exit
- Job scheduling installation exit
- Security validation exit

The SUPERLINK/MVS Installation, Tuning, and Customization Guide, publication SI-0180, provides more information about these user exits.

The installation exit modules are dynamically loaded at startup and remain until closedown. The exit module names are specified in the &UXAM*n* installation options.

An ESTAE is used to detect abends within user exits. If an abend occurs, an appropriate error message is output and the exit is disabled.

Normal conventions are followed for the use of registers. Upon entry the registers are as follows:

Register	*Contents*
R1	Pointer to parameter list
R13	Pointer to 72 byte register save area
R14	Return address

Register	*Contents*
R15	Base address of User exit
R0, R2-R12	Undefined

The installation exits receive control in problem program state, storage protect key 8 and in the addressing mode assigned to the load module. The exits must restore the caller's addressing mode upon return (using the BSM assembler instruction).

S@C9000 - Association Manager Controller Subcomponent

The Association Manager controller subcomponent (AM_controller) is responsible for the overall control of the Association Manager component under the direction of the SLCN root module (S@CC0000).

The AM_controller starts as many processors as required (up to the maximum specified on the &AMLIMIT parameter in the initialization options) to meet the demands placed upon it. It initially starts one processor for each application title specified, using the ATITLE statement in the SUPERLINK options. Each processor issues an A-OFFER service request primitive to process any incoming A-ASSOCIATE indications.

S@C9000 - Module Structure

The AM_controller consists of the following modules:

Module	*Function*
S@C9000	Initialization module
S@C9010	Error recovery (ESTAE) exit routine
S@C9020	State/event machine and termination
S@C9030	Termination request handler
S@C9099	AM Termination module

Figure 15 on page 8-8 shows the hierarchical structure of modules within the AM_controller subcomponent.

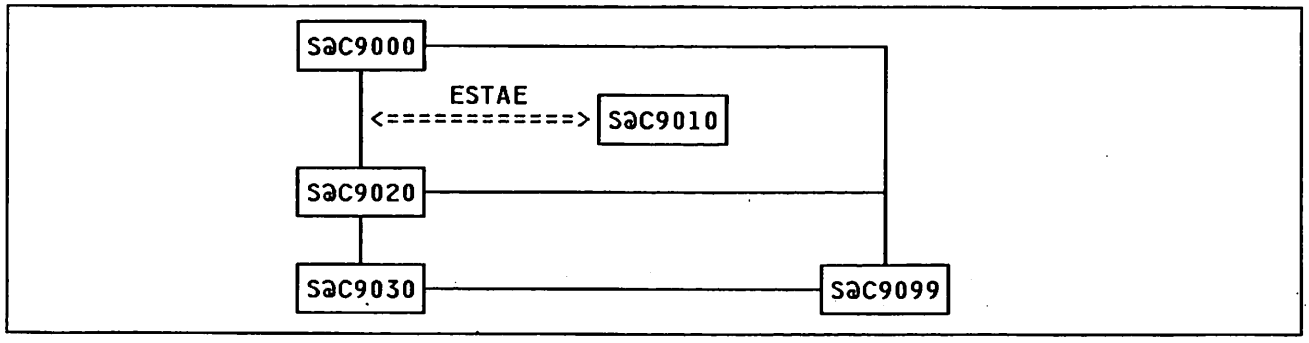


Figure 15. Module Structure of the Association Manager Controller Subcomponent

S@C9000 - Services

The AM_controller performs the initialization and termination of the Association Manager component. During normal running it attaches and controls AM_processor subtasks.

S@C9000 - Initialization phase

Root module S@CC0000 attaches the AM_controller during the initialization phase of SLCN. The AM_controller initializes the data areas that the Association Manager component requires in common storage. It then starts up one AM_processor subtask. Once initialized, the AM_controller enters the active state.

S@C9000 - Active state

The AM_controller listens for any of the following events to occur:

- Shutdown request from SLCN

  The shutdown signal is issued by the root module, S@CC0000. Once the request is accepted, the AM_controller enters its termination phase.

- "Processor idle" notification from a processor

  Once an AM_processor becomes idle, it is ready for work. If there are any application titles that do not have an outstanding receptor for an association request (OFFER), the AM_controller orders the idle AM_processor to register an application title by using the A-OFFER service request primitive. If additional application titles are required, the AM_controller attaches more AM_processors. This cycle continues until there is an A-OFFER request pending for all known application titles.

- "Offer accepted" notification from a processor

  When an A-ASSOCIATE indication is received in response to an A-OFFER service request primitive, an AM_processor subtask notifies the AM_controller, which in turn activates a new AM_processor subtask to issue another A-OFFER request. However, if an AM_processor subtask is already in the idle state, the AM_controller will activate the idle processor, by the POST macro, to indicate that another A-OFFER service request primitive must be issued. Otherwise, the AM_controller attaches a new AM_processor subtask to perform this operation.

- "Queued connection requests not being accepted" notification from a processor

This event occurs if an entity responder uses the S@@MREQ macro to perform a delete end point (DELETE-EP) (the responder will not accept any more A-ASSOCIATE indications) while end points wait for processing on the responder's queue. The AM_controller starts another copy of the entity responder and gives the queue with the end points to the new job for processing.

- "Time-out" inactivity for at least 10 minutes

  The quantity of AM_processors ATTACHed at any one time is determined by the demand placed upon the Association Manager component. If there is a period of inactivity lasting approximately 10 minutes, one idle AM_processor, if there is one, is terminated.

- Processor termination

  If an AM_processor terminates, the AM_controller releases the data areas as required.

S@C9000 - Termination Phase

The following termination sequences are defined:

- Graceful
- Quick
- Abort

The AM_controller reflects the appropriate termination order to its AM_processors, which in turn issue termination orders at the same level to all responders defined in the AM_responder directory. When all of its AM_processor subtasks have been detached, the AM_controller performs clean-up processing and terminates.

"Association Manager Services Offered to Application Entity Responders on MVS" on page 8-4 provides a description of these termination sequences.

S@C9000 - Interfaces

The interfaces are described under "Association Manager Interfaces" on page 8-4.

S@C9000 - Data Areas

When the Association Manager component is initiated, it creates its own series of tables. The following reside in common storage until the Association Manager and all of its processors have terminated:

Table	*Description*
AM_APD	Association Manager Application Program Directory AM_APD is a register of all application titles known to the Association Manager. An A-OFFER service request is issued for each defined application title.
AM_CDT	Association Manager Controller Data Table AM_CDT contains data used by the AM_controller.
AM_GWA	Index to the Association Manager Global Work Area

Table	*Description*
AM_PCT	Association Manager Processor Control Table AM_PCT contains a header for each ATTACHed AM_processor. An entry index is used to find the entries.
AM_RED	Association Manager Responder Directory AM_RED has an entry for each application entity responder. An entry index is used to find the entries.
AM_VT	Association Manager Vector Table AM_VT contains the entry point addresses of Association Manager component modules, pointers to data areas, and other global information. This is the only table accessible to modules that are external to the Association Manager component. It contains an ECB that is posted by an external module when it wants to inform the Association Manager component of termination processing.

"Appendix A. Data Area Descriptions" on page A-1 provides format descriptions of these data areas.

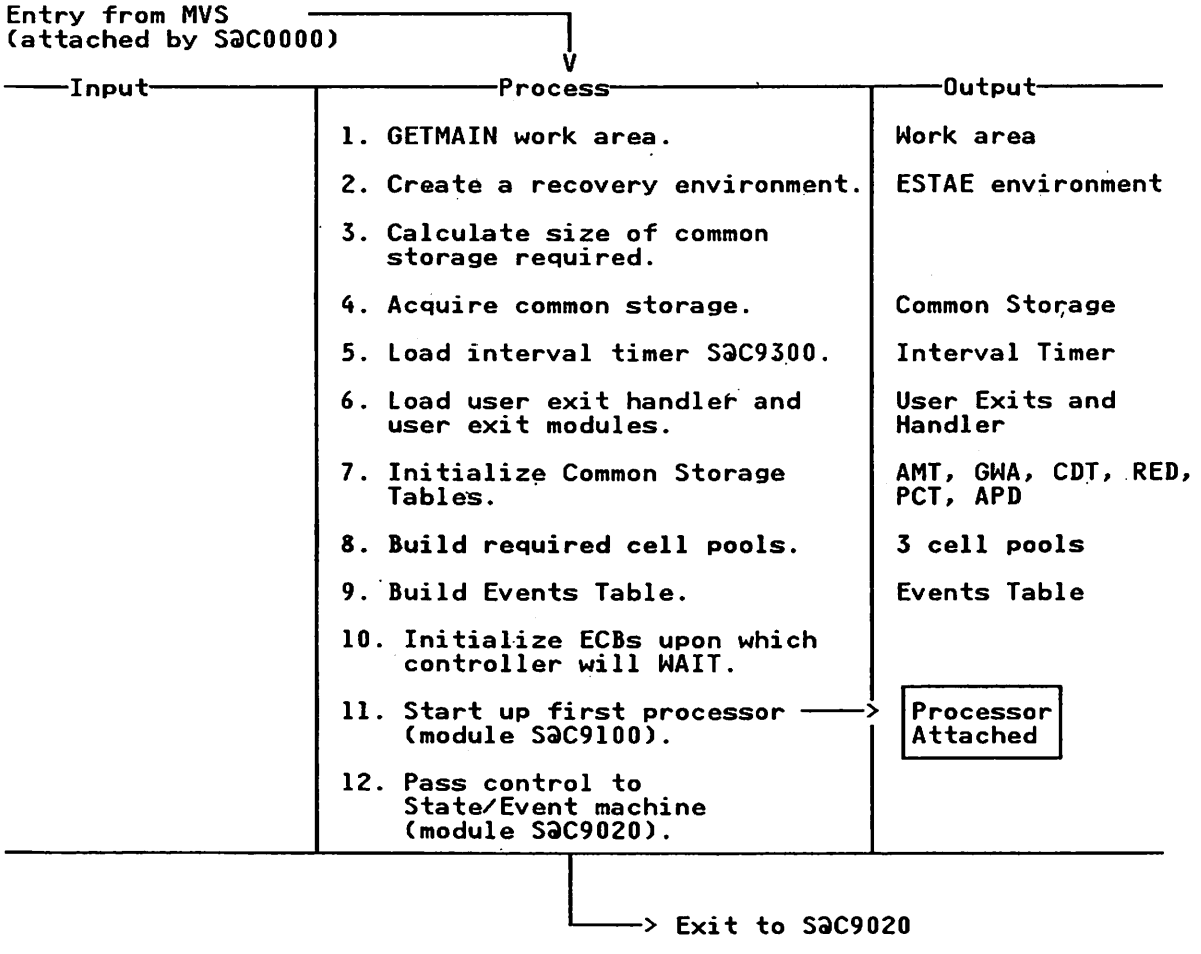
S@C9000 - Recovery

An ESTAE is established at initialization. A message is issued to the SUPERLINK LOG and retry is attempted from the start of the state/event machine processing (S@C9020). The retry routine proceeds as follows:

- Abend during initialization - The initialization sequence performed prior to the abend is reversed, and the Association Manager component then terminates.
- Abend during normal operation - The main cycle is reentered at the top. The process waits for the next event before continuing. If the same abend occurs again, the Association Manager component terminates in as orderly a way as possible.
- Abend during termination - The termination sequence that caused the abend is ignored, and the ESTAE reenters the termination sequence just beyond the section that caused the abend.

This page has been intentionally left blank.

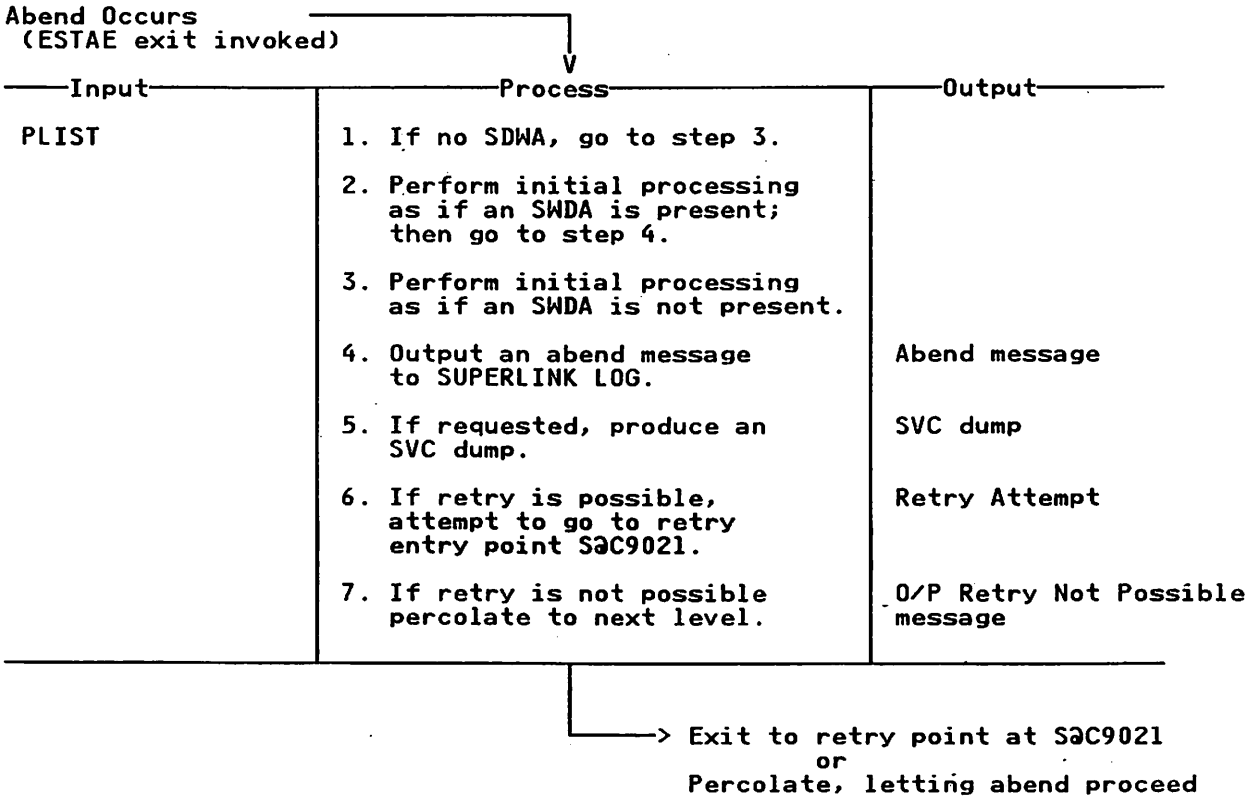
Diagram 8-1
S@C9000 - Root Module



Extended Description

Explanation	*Module*	*Label*
1. The acquired work area starts with the register save area and is followed by a more general work area. This area is passed from module to module in a controller context to eliminate the need for further GETMAINs.	S@C9000	
2. Module S@C9000 issues an ESTAE macro instruction to establish an ESTAE recovery environment. The ESTAE routine (S@C9010) traps all abends scheduled by this module.	S@C9000	
3. The following items in the list are totaled to give the required size of common storage: • Association Manager Vector Table (AMT) size • Global Work Area (GWA) index size • Controller Data Table (CDT) size • Processor Control Table (PCT) header size • (PCT entry size) x (Number of responders + 2) • Responder Directory (REDE) header size • Application Program Directory (APD) header size • (APD entry size) x (Number of Application Titles in options) • 24 bytes for rounding up to full-word boundaries	S@C9000	
5. The Interval Timer is module S@C9300. The entry point for the interval timer is saved in the initial AMT.	S@C9000	
6. The User exit handler is module S@C9UXAM. The entry points for the user exits are saved in the GWA index.	S@C9000	
8. The cell pools required are as follows: • Cell pool with REDE entry elements. The quantity set up is determined by the number of responders specified in the CIOT. • Cell pool containing 40-byte elements used as both queue elements and queue management cells in the REDE • Cell pool with PRB/PRC buffers which are used with each A-OFFER issued	S@C9000	
9. The following proportions are used to calculate the size of the Events Table: • 2 per responder (the ECB on the ATTACH and the ECB for processor-to-controller communication) • 1 for the termination ECB • 1 for the interval timer	S@C9000	
11. Before the processor is started, an entry in the PCT is acquired. The processor is then started using the ATTACH macro.	S@C9000	

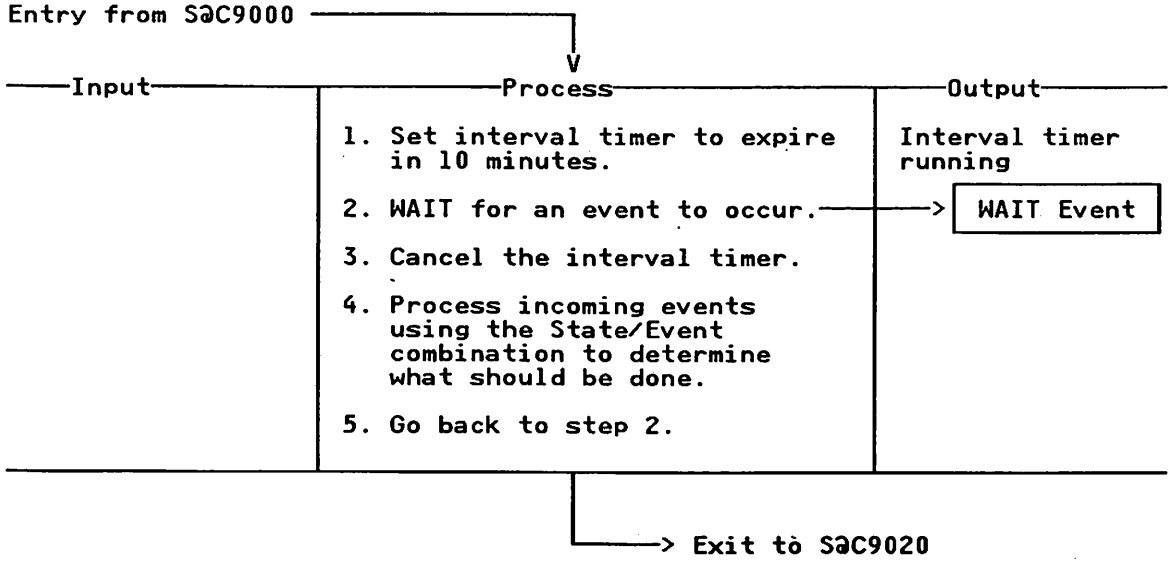
Diagram 8-2
S@C9010 - ESTAE Exit Routine



Extended Description

Explanation	*Module*	*Label*
1. Register 0 contains F'12' if there is no SDWA.	S@C9010	
2. If there is an SDWA, the abend code contained in the SWDA must be saved.	S@C9010	
3. If there is no SDWA, the abend code contained in register 1 must be saved.	S@C9010	
4. The abend message output contains the system abend code, the user abend code, and the reason code. It also indicates whether the abend occurred in the the Association Manager controller subcomponent (S@C9000) or one of the processors.	S@C9010	
6. Retry is not possible in the following conditions: • The retry-not-allowed flag is set in the SDWA. • This is a CANCEL. • This is a step abend. • This is an STAE error/higher task.	S@C9010	
7. If percolation does not occur, another message is written to the SUPERLINK LOG indicating that a retry was not possible.	S@C9010	

Diagram 8-3
S@C9020 - State-event Machine and Termination



Extended Description

Explanation	*Module*	*Label*
1. As demand increases, the quantity of ATTACHed processors increases. The timer is set to expire after 10 minutes of inactivity to terminate Idle processors.	S@C9020	
2. The program WAITs for one of the 'event-has-occurred' ECBs to be posted.	S@C9020	
3. Since an event has occurred, the inactivity timer is cancelled.	S@C9020	
4. As soon as an event occurs, the state/event machine is used to invoke one of the following actions:	S@C9020	

- Graceful termination - Allow no new activity and drain the present activity. As processors become idle, terminate them. When there are no more active processors, terminate the SUPERLINK Association Manager component.
- Quick termination - Flush all activity from the system. Send A-ASSOCIATE (negative) for all unprocessed A-ASSOCIATEs.
- Abort termination - Close down without tidying up.
- Processor gone IDLE - If an Idle processor is required to perform some work then the Idle processor will be POSTed to perform it.

  An idle processor could be used to do the following:

  - Perform an OFFER that needs to be put out
  - Handle association identifiers queued for a processor controlling a task that is unable to handle them.
- A-OFFER taken up - The A-OFFER needs to be issued again. A search for an IDLE processor to do this is performed. If there are no idle processors available to do this, another processor is ATTACHed.
- Timer Expired - The timer has indicated 10 minutes of inactivity. Search for one idle processor and POST it to terminate.
- ATTACHED Processor Terminated - The table entries for the processor are cleaned up. If the processor terminated abnormally, leaving unprocessed A-ASSOCIATEs, they are handled.

S@C9100 - Association Manager Processor Subcomponent

Each Association Manager Processor subcomponent (AM_processor) is ATTACHed and controlled by the AM_controller. During normal operation, separate processors are active simultaneously as different tasks.

S@C9100 - Module Structure

The AM_processor consists of the following modules:

Module	*Description*
S@C9100	Root module; main-line routine.
S@C9110	ESTAE exit routine
S@C9121	Action 1: Issue an A-OFFER service request primitive for each defined application title.
S@C9123	Action 3, Part 1: Handle incoming association protocol information by trying to match it to an active responder. If not possible, call Part 2.
S@C9123B	Action 3, Part 2: Handle incoming association protocol information by initiating a responder to process it.
S@C9124	Action 4: Handle interval timer expiration if "listen" pending
S@C9125	Action 5: Listen has been performed.
S@C9128	Action 8: Duplicate responder entity.
S@C91212	Action 12: Handle interval timer expiration if "delete-any" pending
S@C914A	TEXT and STEXT field parser
S@C914C	Job status/cancel processor
S@C914J	Card images generator
S@C914K	Keyword Table creation
S@C914R	SEND an A-ASSOCIATE (negative response)
S@C914S	Job submission processor
S@C914T	Task attach processor
S@C914X	Create/Delete system access facility (SAF) environment processor

Figure 16 on page 8-19 shows the hierarchical structure of modules within the AM_processor subcomponent.

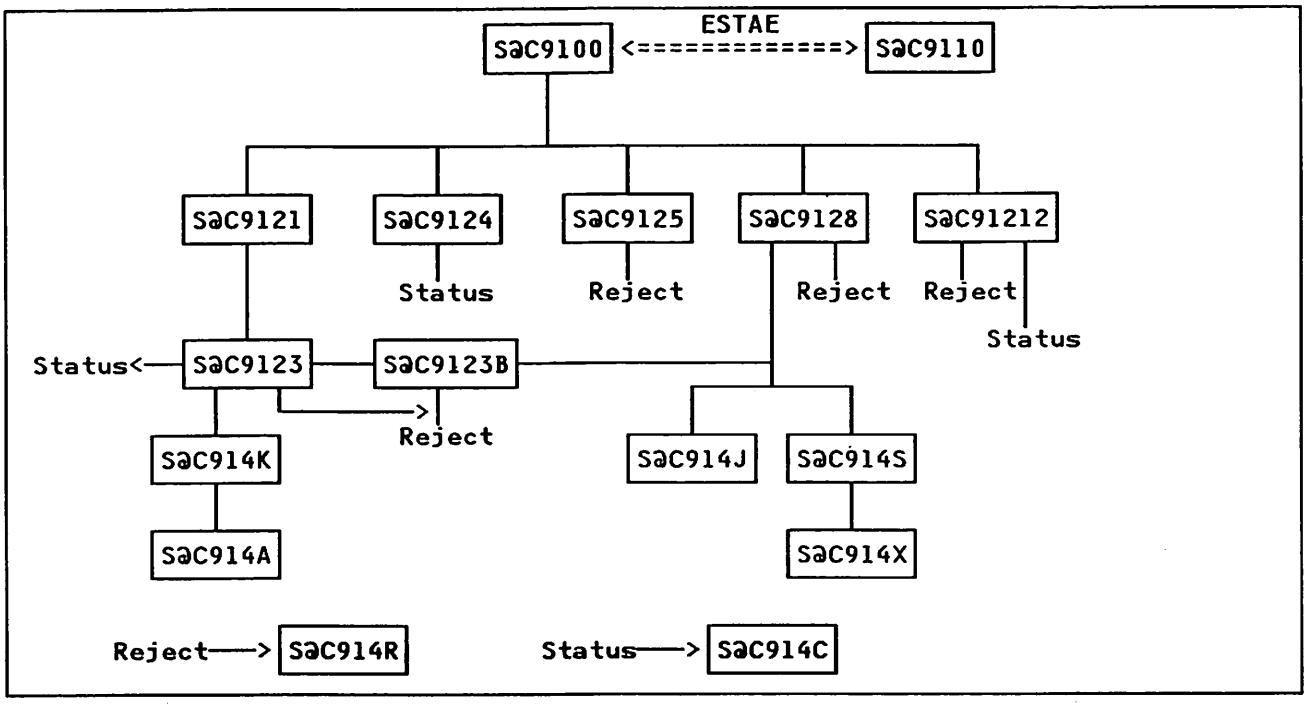


Figure 16. Module Structure of the Association Manager Processor Subcomponent

S@C9100 - Services

The AM_processor provides the following services:

- Initially issues an A-OFFER service request primitive for an Application Title (as requested by the AM_controller)
- Waits for an incoming A-ASSOCIATE indication
- Processes the connection end point if the Association Manager can locate an appropriate responder
- Takes the following actions if the Association Manager cannot locate an active responder:
  - Initiates a new responder to process the connection end point
  - Holds the connection end point until the responder becomes active
  - Passes over all queued end points once the responder is active

The AM_processor is event-driven and uses a finite-state machine (FSM) strategy to perform all its functions. The following states are used for the AM_processor FSM:

State	*Description*
FREE	Initial and final state of the FSM
INIT	AM_processor initializing; initialization not complete.
IDLE	Waiting for an order to be given by the AM_controller

State	*Description*
ACTIVE	An IDLE processor has been posted but the request has not yet been delivered.
OFFER-PD	OFFER pending; waiting for an A-ASSOCIATE indication primitive. The AM_processor can cancel the pending A-OFFER service request by issuing an A-RELEASE service request primitive. The A-OFFER service request is also cancelled if the AM_controller issues a termination order.
LISTEN-PD	LISTEN pending; waiting for a LISTEN indication from the submitted companion job.
DELETE-PD	DELETE pending; waiting for a DELETE TYPE = ANY indication from the MVS responder.
TERM	AM_processor terminating; termination not yet completed.

When an A-ASSOCIATE indication has been received by an AM_processor, the AM_processor notifies the AM_controller. The AM_controller responds by indicating to another AM_processor task, through the POST macro, that the AM_processor must issue an A-OFFER service request primitive for the application title. If necessary, a new AM_processor task is ATTACHed.

For each connection end point, an element is added to the request queue associated with the application entity responder. When a responder entity is listening, connection end points are handed over to it.

If a responder is to be initiated in response to an incoming association identifier, the following sequence of events is followed:

1. A Keyword Table is generated. This table contains all the variables permitted in the companion job's skeleton JCL, specified by SUPERLINK options. The TEXT, IDEN, and AUTH fields on the incoming A-ASSOCIATE indication may contain values that override the default values from the SUPERLINK Options.
2. The Keyword Table is passed to user exit 1 (S@C9UX1 - variable validation user exit).
3. The identification and authentication information is passed to user exit 4 (S@C9UX4 - security validation).
4. The JCL is resolved by taking the skeleton JCL associated with the application title being processed and substituting the real values for the variable keywords from the Keyword Table.
5. The resolved JCL is passed to user exit 2 (S@C9UX2 - pre-JCL submit user exit).
6. The JCL is submitted according to the following sequence:
    a. Dynamically allocate the JES internal reader
    b. Create environment; call to the SAF using RACINIT via RACROUTE.
    c. Open internal reader
    d. Output JCL using VSAM PUT
    e. Close the internal reader
    f. Delete environment; call to SAF using RACINIT via RACROUTE.
    g. Dynamically deallocate the JES internal reader

A check is made at regular intervals to determine if the submitted JOB has started execution. This is done as follows:

- Create a Subsystem OPTIONS Block (SSOB) to request the status of the JOB submitted and chain it off the Subsystem Interface Block (SSIB).
- Make the call to JES, in supervisor mode, using the MVS subsystem interface request macro (IEFSSREQ).
- Determine JOB status from the return code in the SSOB.

If the companion job is not initiated within an expected time period, one of the following events occurs:

- User exit 3 is called (S@C9UX3 - job scheduling user exit), if it exists. It returns with one of the following:
  - Continue; corrective action taken.
  - Please wait.
  - Try later.
  - Take action specified in Options.
- The cancel-the-job option is taken if it was specified via Options; if no option was specified, the cancel is performed in the same manner as a status request.
- The ask-the-operator option is taken if it was specified via Options. The operator is asked to select either the continue-to-wait or cancel-job option.

S@C9100 - Interfaces

See "Association Manager Interfaces" on page 8-4 for a complete description of AM_processor interfaces.

S@C9100 - Data Areas

See "S@C9000 - Data Areas" on page 8-9 for a complete description of AM_processor data areas.

S@C9100 - Recovery

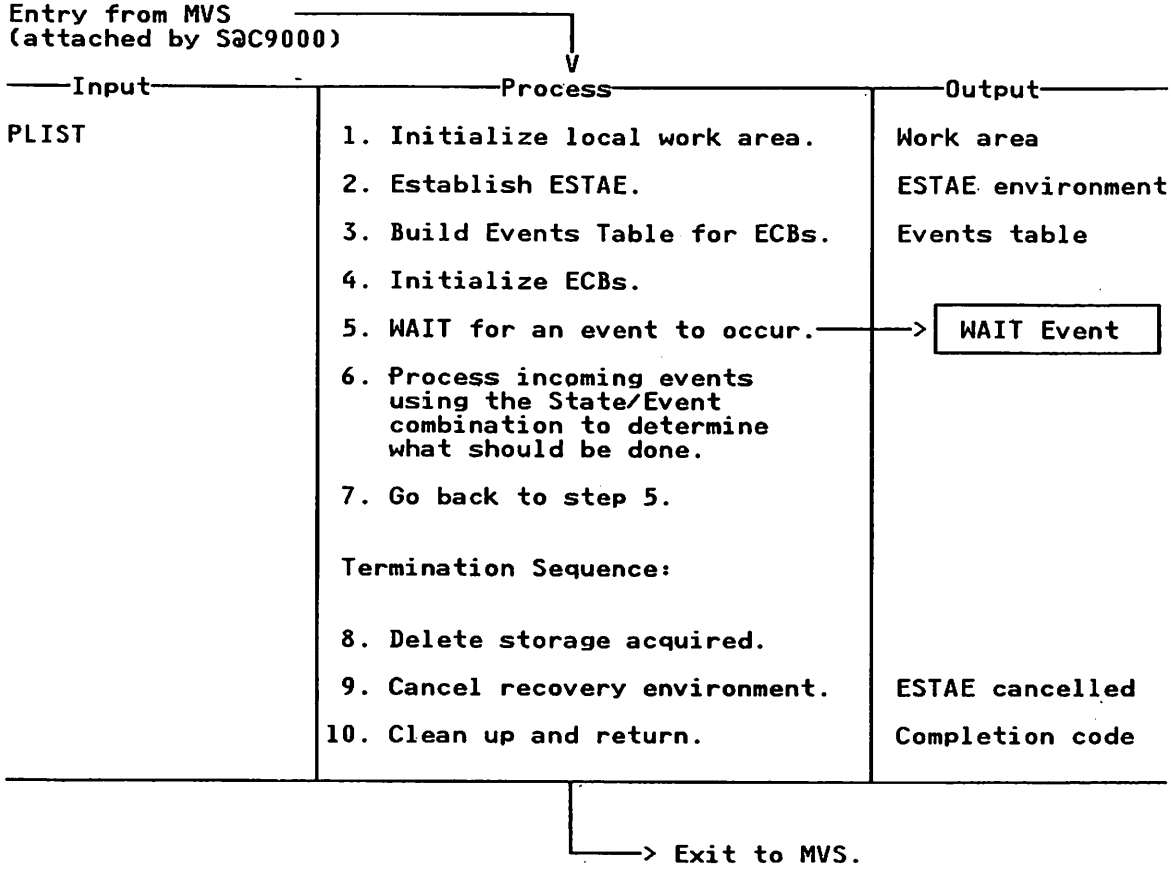
In the event that an abend occurs within an AM_processor, AM_processor ESTAE is invoked. This recovery routine performs the following sequence:

1. Issues a message containing the abend code and reason code.
2. Resets internal flags, indicating the new state of the AM_processor
3. Produces an SVC dump is produced if the option has been selected.
4. Attempts to restart the AM_processor, if this is not possible, the abend is percolated.

If percolation occurs, any storage being used for presentation request buffers (PRB) and presentation request contexts (PRCs) are released. Any jobs pending are also cancelled and all end points held by the processor are rejected.

If a retry is possible, the AM_processor is reentered at the start of the main cycle. Upon retry, a position flag is used to determine the appropriate action that is to be taken.

Diagram 8-4
S@C9100 - Root Module and Main-line Code

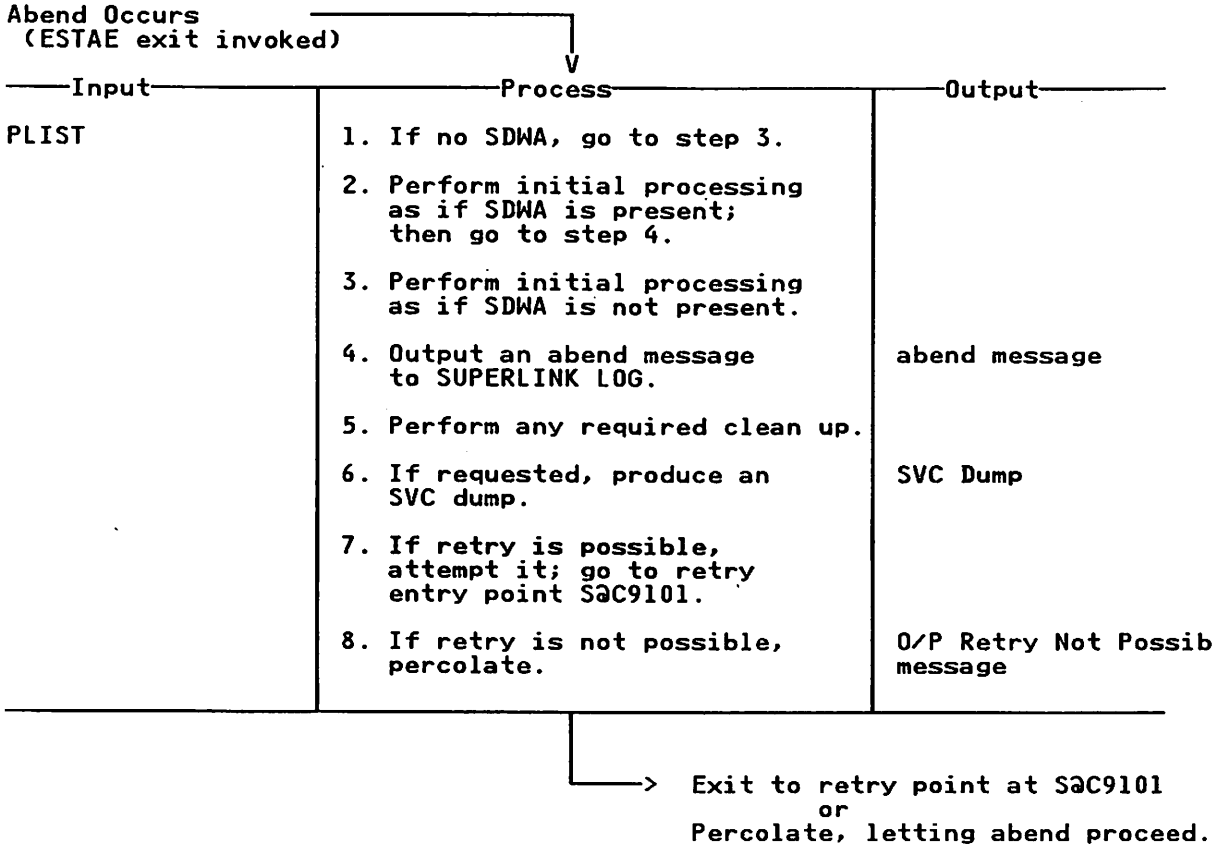


Extended Description

Explanation	*Module*	*Label*
2. S@C9100 issues an ESTAE macro instruction to establish an ESTAE recovery environment. The ESTAE exit routine (S@C9110) traps all abends scheduled by this module.	S@C9100	
3. The EVENTs macro is used to build a table to hold four ECBs.	S@C9100	
4. The ECBs in the Events Table are initialized. They are all contained in the PCT entry associated with the processor.	S@C9100	
5. The program WAITs for one of the ECBs to be POSTed.	S@C9100	
6. As soon as an event occurs, the State/Event combination is used to determine which of the specified actions should be executed. One of the following actions is performed when an event occurs:	S@C9100	

- Action 1: Put out an A-OFFER for a specific title, (Module S@C9121 is invoked to perform this action).
- Action 2: Cancel the A-OFFER that was put out.
- Action 3: Process an incoming A-ASSOCIATE indication, (Modules S@C9123 and S@C9123B are invoked to perform this action).
- Action 4: Timer expired when LISTEN pending so determine if responder is active, (module S@C9124 is invoked to perform this action).
- Action 5: LISTEN performed by JOB; perform END-POINT GIVES for all Queued Association Identifiers, (Module S@C9125 is invoked to perform this action).
- Action 6: DELETE-ANY performed by JOB; put processor back into an IDLE state.
- Action 7: Operator replied to a WTOR; analyze his response.
- Action 8: Cloning requested; spin off a copy of the requested job and swap queues, (Module S@C9128 is invoked to perform this action).
- Action 9: DELETE-EP performed by JOB; determine whether cloning is required. If so, set that process in motion.
- Action 10: ATTACHed task terminated; clean up after the task.
- Action 11: Handle a closedown request received while in the LISTEN-PENDING state.
- Action 12: Timer expired when DELETE pending so determine if responder still active, (module S@C91212 is invoked to perform this action).

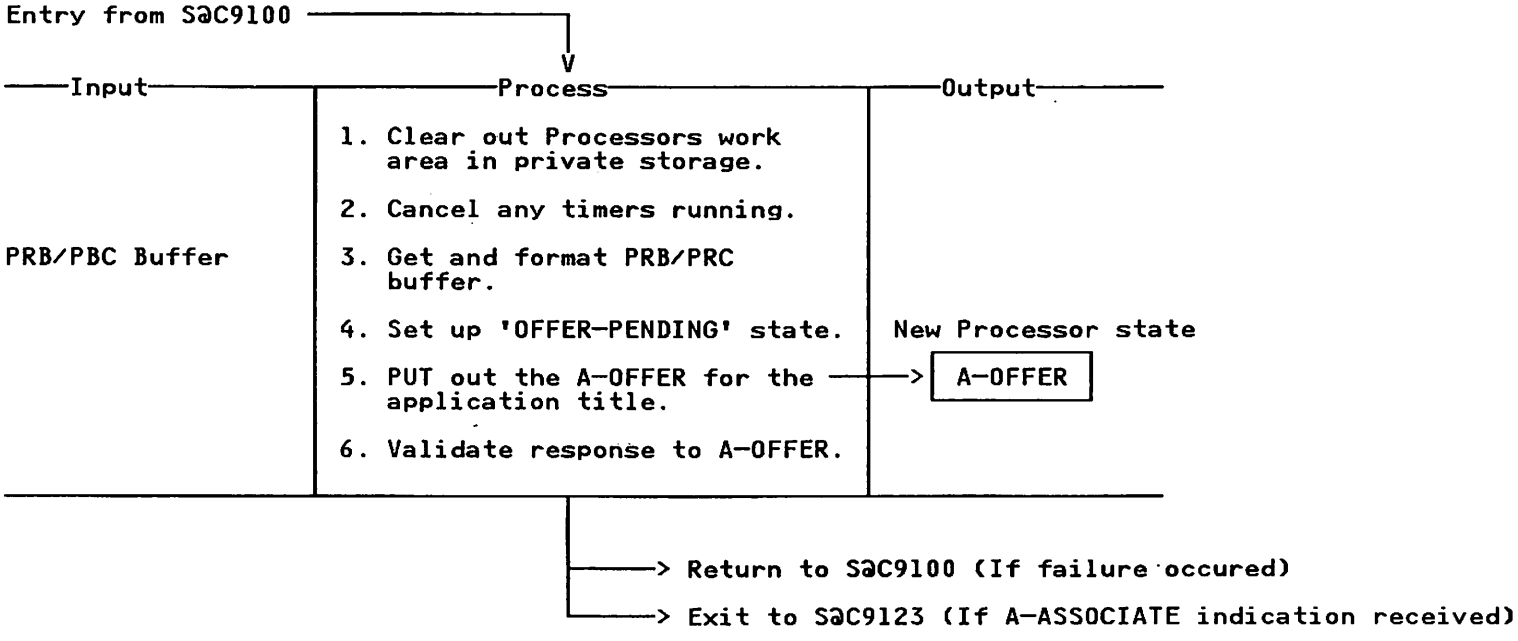
Diagram 8-5
S@C9110 - ESTAE Exit Routine



Extended Description

Explanation	*Module*	*Label*
1. Register 0 contains 12 if there is no SDWA.	S@C9110	
2. If there is an SDWA, the abend code contained in it must be saved.	S@C9110	
3. If there is no SDWA, the abend code contained in register 1 must be saved.	S@C9110	
4. The abend message output contains the system abend code, the user abend code, and the reason code. It also indicates if the abend occurred in the Association Manager controller subcomponent (S@C900) or in one of the processors.	S@C9110	
5. The required clean up actions are as follows: • Clear out Job-not-known-yet lock. • Any ENQs on the Responder Directory Header or the responder Directory entry are DEQed.	S@C9110	
7. Retry is not possible under the following conditions: • The Retry-not-allowed flag is set in the SDWA. • This is a CANCEL. • This is a step abend. • This is an STAE error/higher task.	S@C9110	
8. If percolation does not occur the following will occur: • An appropiate "PERCOLATION HAS OCCURED" message will be output • The PRB/PRC cell will be released if obtained for an A-OFFER • A responder is cancelled using module S@C914C if it is pending • Any outstanding connection end points will be rejected using module S@C914R	S@C9110	

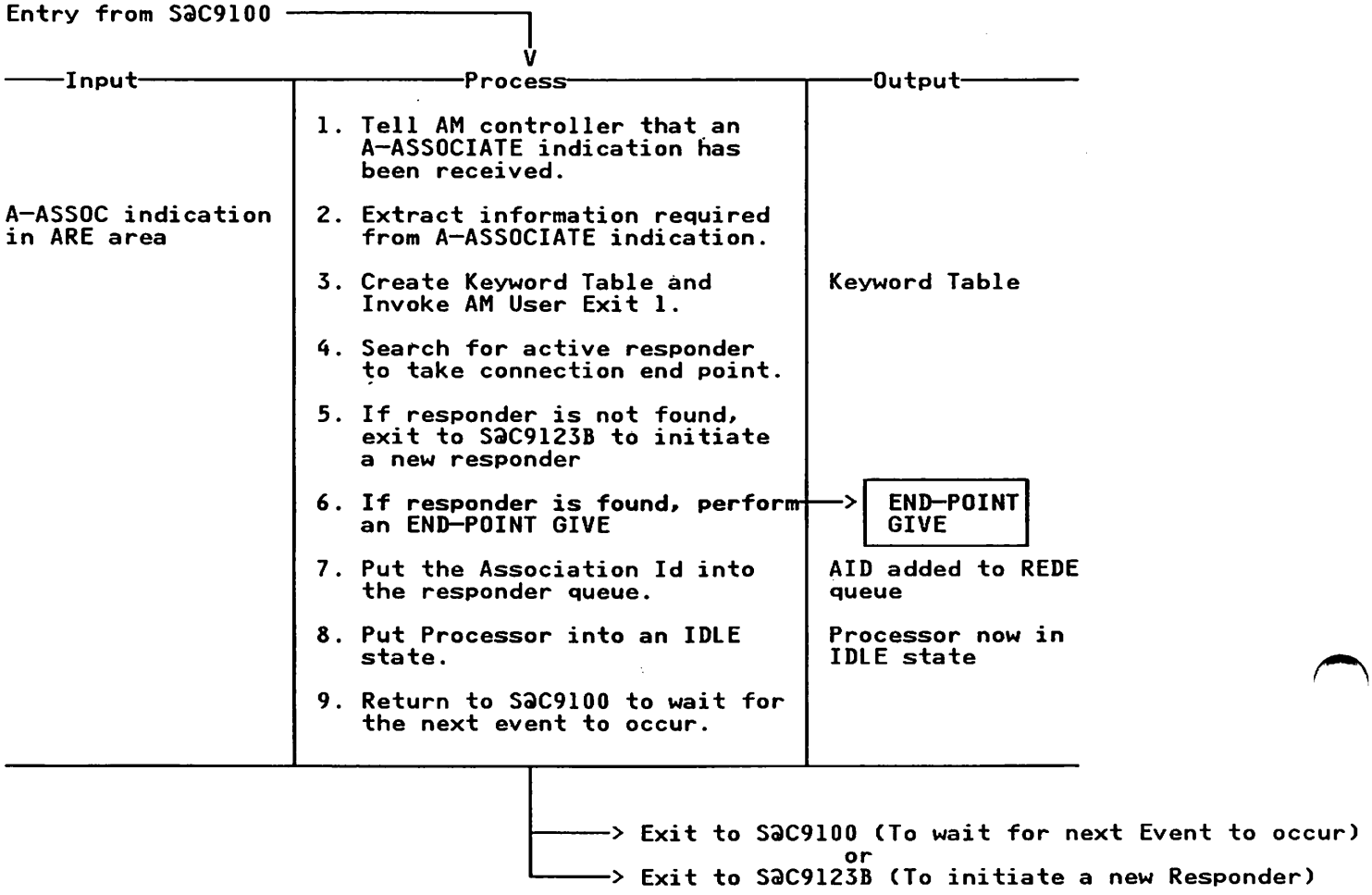
Diagram 8-6
S@C9121 - Action 1, Put Out an A-OFFER



Extended Description

Explanation	*Module*	*Label*
1. Only the top end of the processor's working storage is cleared out. This includes the private REDE section and the CASE ARE section.	S@C9121	
3. The PRB/PRC buffer is obtained from a cell pool and consists of a section of ECSA storage. It will be split up as follows: • One PRB • Three PRBEs • Two 4K Inbound buffers • One PRC • The required number of PRCEs (minimum two, maximum three)	S@C9121	
5. Once the CASE call is made to perform the OFFER, it will return only under the following conditions: • An A-ASSOCIATE is received. • A P-ABORT is received. • An error occurs. • The Association Manager controller subcomponent posts the Cancel ECB.	S@C9121	
6. Once a response is received to the A-OFFER the following validation is carried out: • If the cancel ECB was posted, return to S@C9100 to wait for another event, (Possibly a cancel). • If an A-ASSOCIATE indication was received, exit to module S@C9123 to process it. • If this module is in a termination phase, return to S@C9100 to wait for the closedown request. • If the A-OFFER failed because the maximum number of CASE responders limit has been reached, then set a timer to expire after an interval. Then exit to S@C9100 to wait for the timer to expire. When the timer expires, an attempt is made to re-issue the A-OFFER. • If an error occurred, output an error message. If this is the 5th occurrence of the error, mark the Application Title being handled as 'FAILED' and go idle. Otherwise, if its not the 5th occurrence, issue the A-OFFER again.	S@C9121	

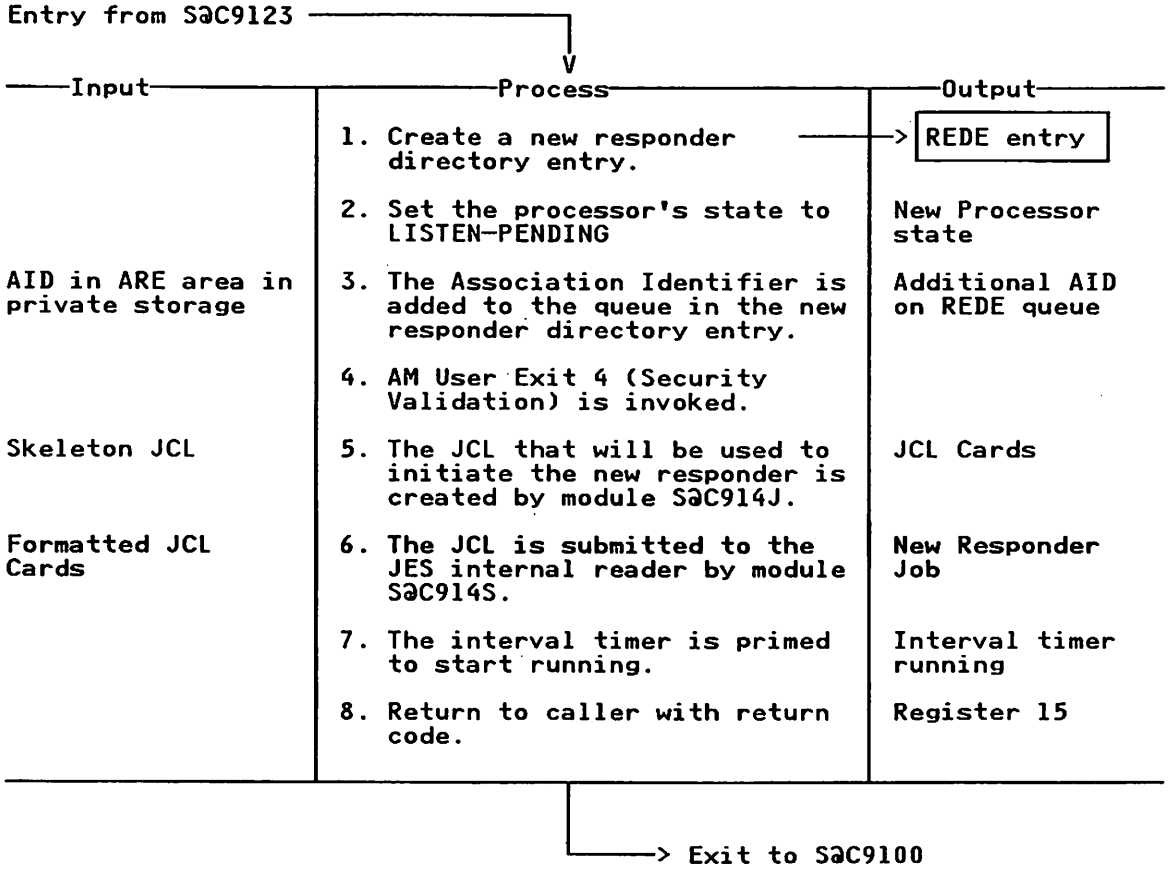
Diagram 8-7
S@C9123 - Action 3 (Part I), Give End-point To Active Responder



Extended Description

Explanation	*Module*	*Label*
1. The AM controller's ECB is POSTed. This causes the AM controller to get another processor to put out an A-OFFER for the particular application title being handled.	S@C9123	
2. The following is extracted from the incoming A-ASSOCIATE indication: • Identification Fields 1 to 3 • Authentication Fields 1 to 3, (2 and 3 are not used by the Association Manager itself, only passed to the Security Exit AM user exit 4) • The contents of the Text Field • The PSAP id • The originators Mainframe Id	S@C9123	
3. The Keyword table of variable values is created using module S@C914K. Module S@C914K invokes User Exit 1 (Variable validation) via the S@C9EXIT macro.	S@C9123	
4. The Responder Directory (REDE) is searched for a similar application title. If one is found, the caller's user ID, password, group ID, and account are compared with those of the responder initiator; if they match, the responder is permitted to process the Association Identifier (AID). For the duration of the search, seralization is ensured by ENQing upon the Responder Directory Headers address.	S@C9123	
5. Steps 6 and 7 are serialized using an ENQ. The address of the responders entry in the directory will be the minor name for the ENQ.	S@C9123	
6. If the responder is in a LISTEN state, then the end point is given to the responders address space. Otherwise, the end point is given to the SUPERLINK address space. (SUPERLINK/MVS keeps the end point until it is told that the responder issued a LISTEN or the responder terminates)	S@C9123	
7. The Association Identifier is inserted into the responders queue using the S@C@QADD macro.	S@C9123	

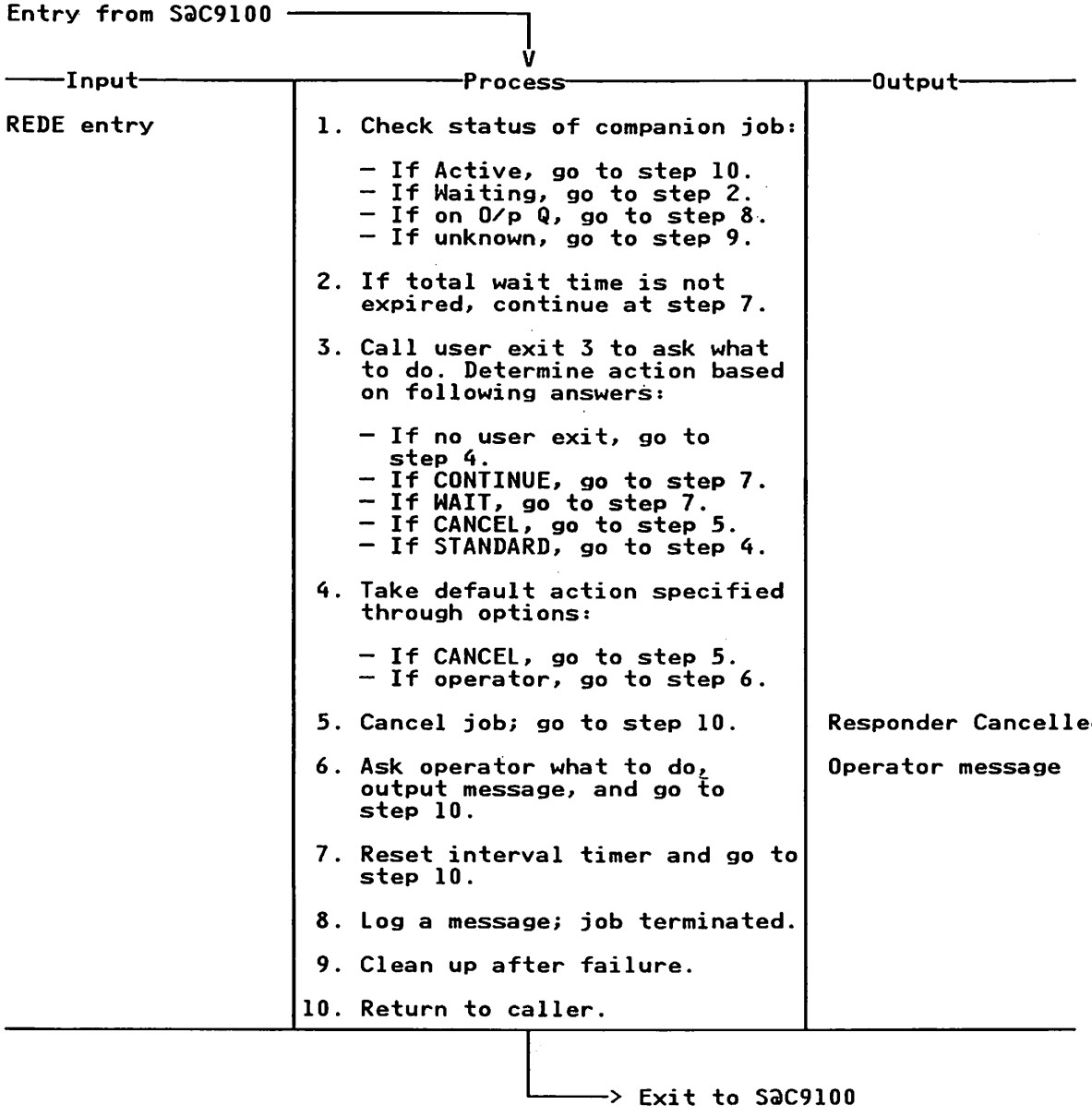
Diagram 8-8
S@C9123B - Action 3 (Part II), Start a New Responder



Extended Description

Explanation	*Module*	*Label*
1. The new REDE is obtained from a Cell pool. The pointer to it is inserted into the Responder Directory header. A Queue anchor is also obtained from another cell pool and the pointer to it is inserted into the new REDE entry. Both the new REDE and its queue anchor are located in ECSA. Serialization is ensured by ENQing upon the Responder Directory Headers address. If an error occurs in any of the following steps right up to step 8 the A-ASSOCIATE indication being handled will be rejected, the new REDE and its queue anchor will be released, the processors state will be set to IDLE, and control will be returned to S@C9100 to wait for the next event. Appropriate error messages will also be issued during this process.	S@C9123B	
3. The Association Identifier is inserted into the responder's queue using the S@C@QADD macro.	S@C9123B	
4. The user exit is invoked via the S@C9EXIT macro.	S@C9123B	
7. The interval for the timer is obtained from the SUPERLINK Options deck.	S@C9123B	

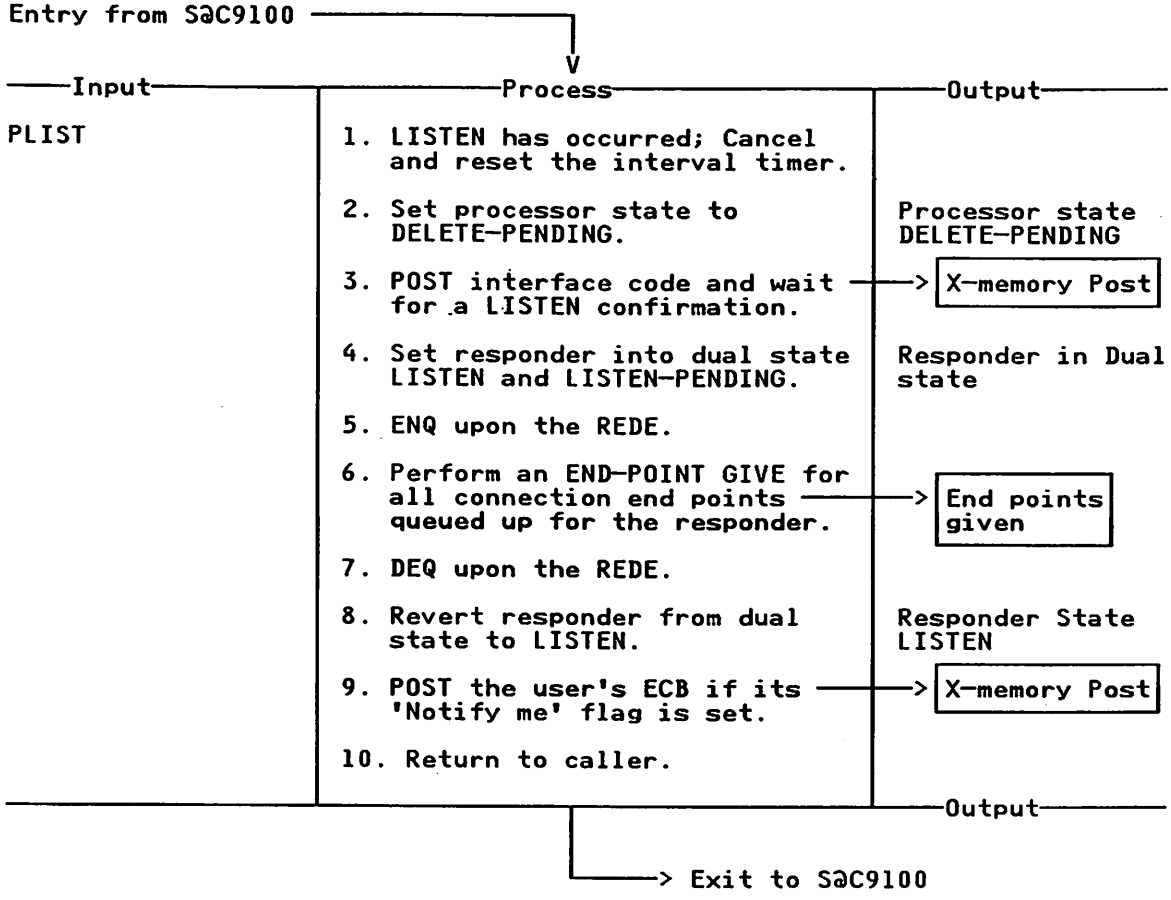
Diagram 8-9
S@C9124 - Action 4, Timer Expired When LISTEN Was Pending



Extended Description

Explanation	*Module*	*Label*
1. The status of the responder entity is determined by calling module S@C914C with register 1 set up to request job status.	S@C914C	
3. User exit 3 may ask the processor to perform one of the following actions when a job has failed: • Continue; corrective action has been taken. • WAIT for the job to start. • CANCEL the job and send a message (TRY LATER). • Perform the standard default action specified in the options.	S@C9124	
4. The default actions for job failures which may be specified through SUPERLINK/MVS options are as follows: • CANCEL the job. • Ask the operator what to do.	S@C9124	
5. The responder entity is cancelled by calling module S@C914C with register 1 set up to request a CANCEL.	S@C9124	
6. The messages macro is used to send a message to the operator, specifying TYPE = WTOR + LOG.	S@C9124	
9. Clean up consists of sending a negative response for the A-ASSOCIATE received and setting the processor's status to IDLE.	S@C9124	

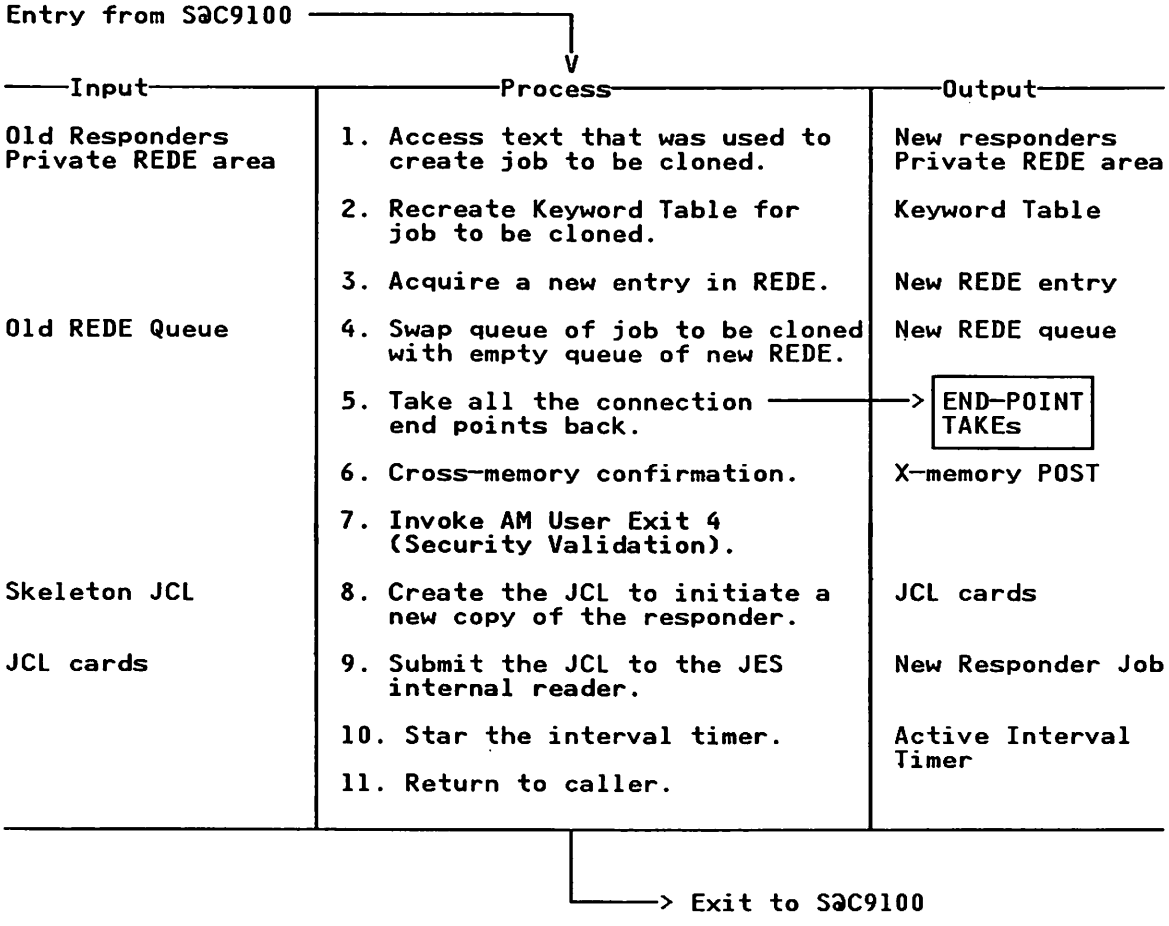
Diagram 8-10
S@C9125 - Action 5, Perform Required END-POINT GIVES Routine



Extended Description

Explanation	*Module*	*Label*
1. The Responder is now active and S@C9125 is no longer checking to see if it is active at intervals. S@C9125 still requires the timer to expire after intervals. To determine if it is still active, it might terminate without doing a DELETE-ANY	S@C9125	
2. If the STATE is already DELETE-PENDING, this is the second time through. Therefore, return to the main cycle without doing anything.	S@C9125	
3. The Listen notification from the interface code is serialized hence S@C9125 needs to cross-memory post it to confirm that the listen notification has been received.	S@C9125	
4. The responder is set into a dual state; LISTEN means that any further connections added to the responders queue will have an END-POINT GIVE performed and the LISTEN-PENDING means that the responder itself will be inhibited from taking queued items for processing until S@C9125 changes the state in step 8	S@C9125	
5. To ensure serialization occurs, an ENQ is performed. The responders REDE entry address will be the minor name. (If a DELETE-ANY occurs the interface code will hold the responder until a DEQ is performed).	S@C9125	
6. Both the request queue and work queue are processed. Prior to each END-POINT GIVE, a flag is checked to see if a DELETE-ANY was initiated and if so, any outstanding end points are rejected via module S@C914R and control is returned to S@C9100. If any of the END-POINT GIVEs fail, then the individual connection end point will be rejected.	S@C9125	
9. The "Notify Me" flag is set if the responder performs a successful NOTIFY call. The NOTIFY supplies the address of an ECB that is to be posted when there is something to pass to it. Since the END-POINT GIVEs have been performed, the queued items are now ready to be passed.	S@C9125	

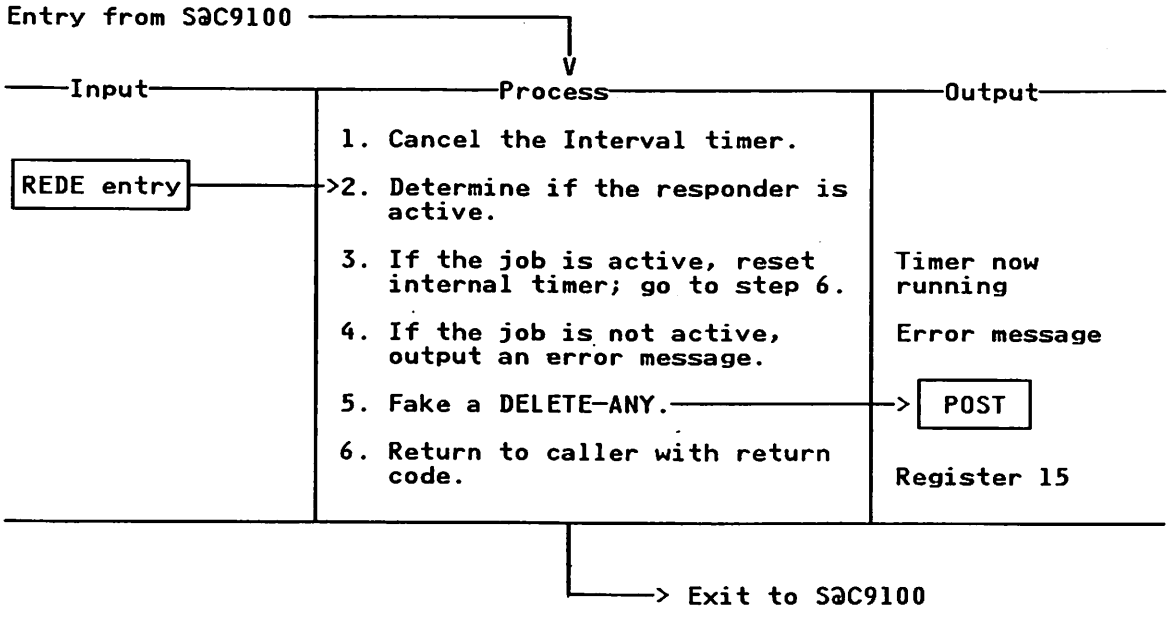
Diagram 8-11
S@C9128 - Action 8, Clone Responder Entity Routine



Extended Description

Explanation	*Module*	*Label*
1. The text is held in private storage; the responder's entry in the REDE points to this storage area. It is extracted and copied.	S@C9128	
2. Module S@C914K is used to recreate the Keyword Table.	S@C9128	
5. All the connection end points on the queue just exchanged the need to be taken with an END-POINT TAKE from the responder that had them. They will be given to the new responder when it perform a LISTEN	S@C9128	
6. The DELETE-EP that invoked the cloning process is a confirmed service. S@C9128 needs to cross-memory post the interface code to inform it that the queue items have now been successfully extracted and will be processed.	S@C9128	
7. User exit 4 is invoked using macro S@C9EXIT.	S@C9128	
8. Module S@C914J is used to create the JCL for the initiation of the new copy of the responder	S@C9128	
9. Module S@C914S is used to submit the JCL cards to the JES internal reader of the responder	S@C9128	
10. Now that the job has been started, S@C9128 will need to check at intervals to determine if the job is active yet or if it has failed. The timer is set to run for the first interval period.	S@C9128	

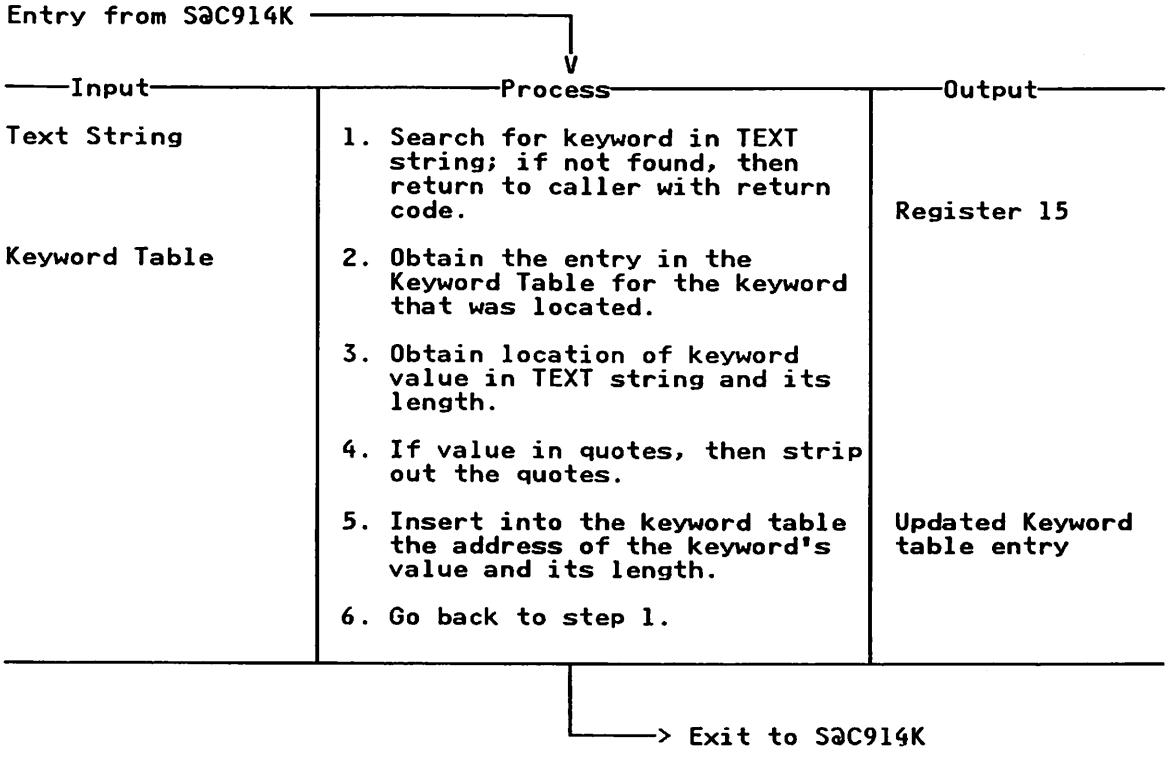
Diagram 8-12
S@C91212 - Action 12, Timer Expired When DELETE Was Pending



Extended Description

Explanation	*Module*	*Label*
1. To ensure that the interval timer is no longer running, it is cancelled because it is possible for a timer expiration to be faked to force the invoking of this module.	S@C91212	
2. Module S@C914C is used to determine if the responder is still active	S@C91212	
3. If the responder is still active, than everything is OK. All that needs to be done is to reset the timer so that a check can be made again later.	S@C91212	
4. The responder is no longer there and an appropriate error message is generated.	S@C91212	
5. The entry for the responder in the Responder directory and any outstanding end points need to be tidied up because a responder has terminated without performing a DELETE-ANY. To do this, a DELETE-ANY is faked using the POST macro.	S@C91212	

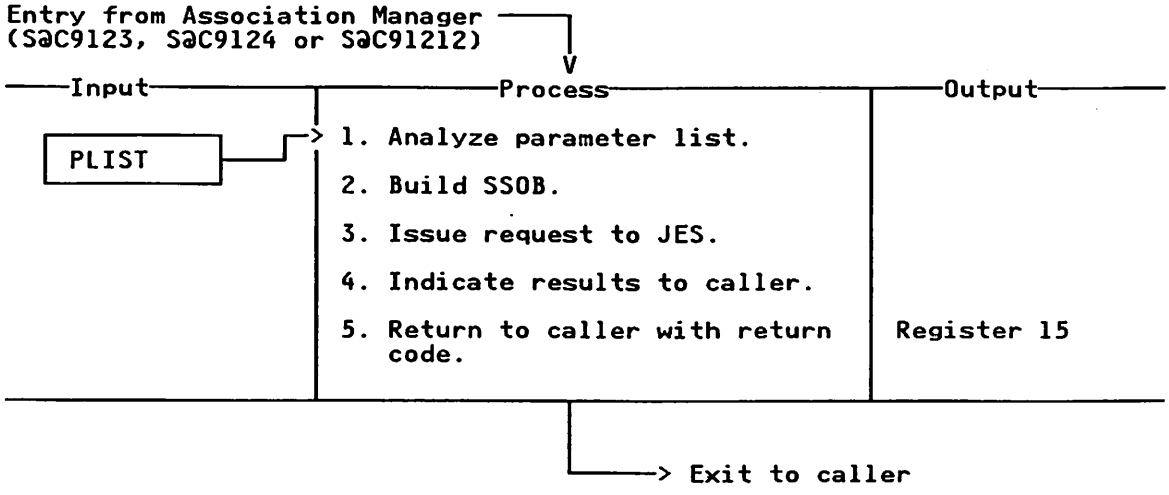
Diagram 8-13
S@C914A - JOBTEXT Field Parser



Extended Description

Explanation	*Module*	*Label*
1. S@C914A scans for the next keyword in the text string supplied by the caller. When the end of the string is reached, the caller is returned with an appropriate return code. If a syntax error is encountered, then the scan is aborted and a non-zero return code is given to the caller.	S@C914A	
2. The entry that is to be updated in the keyword table is located. If not found, (keyword not recognized), the module exits to the caller with an appropriate non-zero return code.	S@C914A	
3. If the value is enclosed within parenthesis, then everything within the parenthesis, (including the parenthesis), is considered to be the value.	S@C914A	
4. Everything between the quotes, (except the quotes themselves), is considered to be the value. Double quotes within the quotes are treated as a single quote.	S@C914A	
5. Everytime a keyword table entry is updated, a flag is set to indicate it has been processed. If a flag is found to be already set, then duplication has occurred. Another flag is set to indicate that this has happened. Upon return to the caller, a non-zero return code will be set to indicate duplication has occurred if the duplication flag is set.	S@C914A	

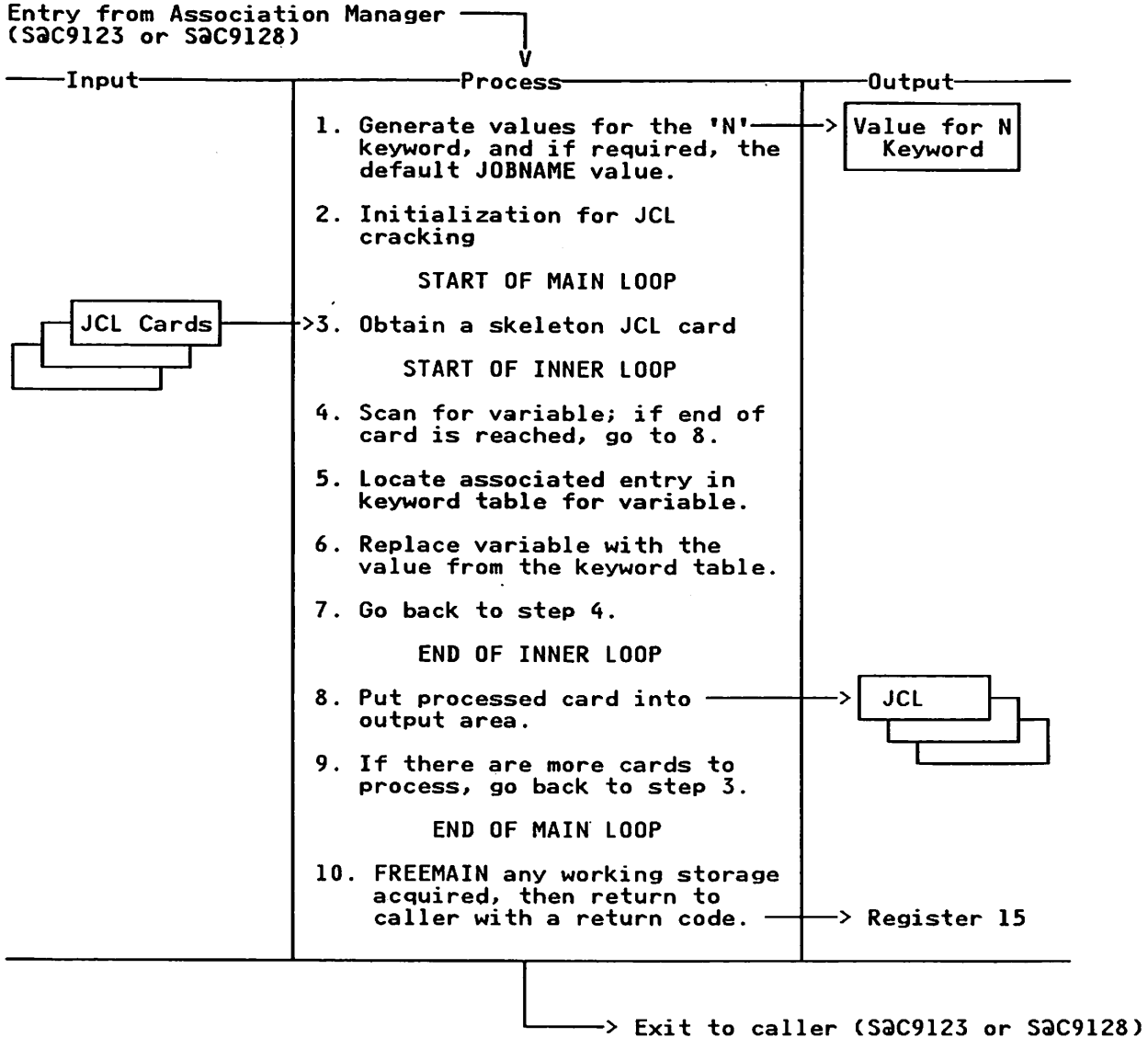
Diagram 8-14
S@C914C - Job Status/Cancel Processor



Extended Description

Explanation	*Module*	*Label*
1. The parameter list (PLIST) is for either a STATUS or a CANCEL request. It contains such associated information as JOBNAME and JOBID.	S@C914C	
2. The Subsystem Interface block (SSOB) is built, then attached to the SSIB.	S@C914C	
3. A subsystem request is issued using the IEFSSREQ macro.	S@C914C	
4. The results from the subsystem request are returned to the caller as completion code (register 15) and feedback code (register 0).	S@C914C	

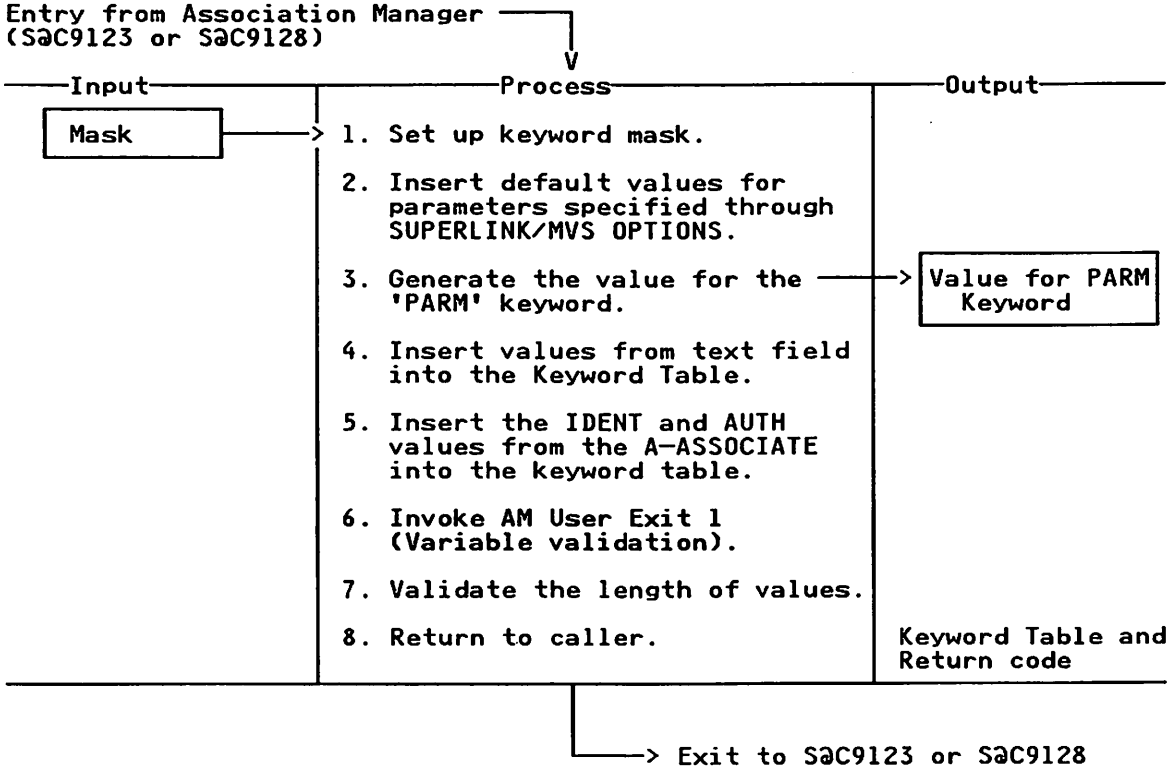
Diagram 8-15
S@C914J - Card Image Generator



Extended Description

Explanation	*Module*	*Label*
1. The 'N' keyword variable value is generated as a unique 4 digit number. If no JOBNAME value was specified, then the default value generated consists of the User Id plus a unique character. If a JOBNAME default was provided via the SUPERLINK options deck, then the default generated is this value padded out to eight bytes using the value of the 'N' keyword to pad it.	S@C914J	
2. Pointers are established to the keyword table and the Skeleton JCL. An output area for the processed cards in private storage below the line is also GETMAINed.	S@C914J	
3. The card to be processed is moved into a work area.	S@C914J	
4. The card is scanned for the variable delimiter which was specified in the SUPERLINK options deck. Once all of the card has been processed, then the inner loop is finished.	S@C914J	
5. If the variable in the skeleton is not a recognized keyword, then the card will be output with an appropriate error message.	S@C914J	
6. If a length was specified with the variable in the skeleton JCL, then only the length of the value requested will be inserted into the card. If the length requested is greater than the length of the value, then it will default to the actual length of the value.	S@C914J	
10. If any errors occurred, then the output area is FREEMAINed and processing returns to the caller with an appropriate error message. Otherwise, if no errors occurred, the processing returns to the caller with a zero return code.	S@C914J	

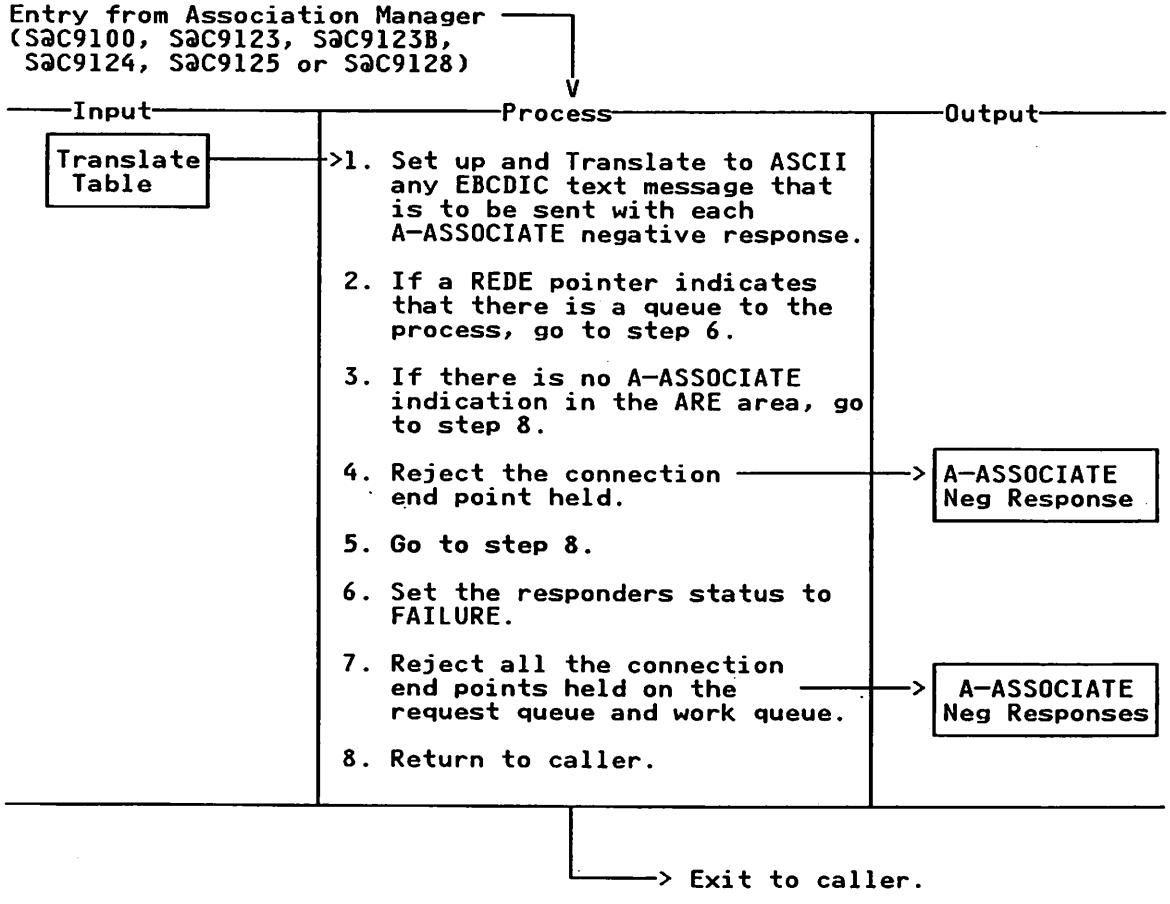
Diagram 8-16
S@C914K - Create Keyword Table



Extended Description

Explanation	*Module*	*Label*
1. The keyword mask contains the keywords and null values for the keyword parameters.	S@C914K	
3. The value generated for the "PARM" keyword consists of the SSVT address (8 digits giving a hex repesentation), the index number for the responder (8 digits giving a translation of the binary index number), and "R", which tells it to run in responder mode.	S@C914K	
4. The parse-a-text-string processor (S@C914A) inserts the text values into the table.	S@C914K	
5. The Identification and Authentication information is saved initially in private storage by module S@C9123 when an incoming A-ASSOCIATE indication is being processed. Pointers to those values are then inserted into the keyword table.	S@C914K	
6. The user exit is invoked via macro S@C9EXIT.	S@C914K	
7. The length of each value in the keyword table is checked to ensure that it does not exceed the maximum permitted for its associated keyword.	S@C914K	

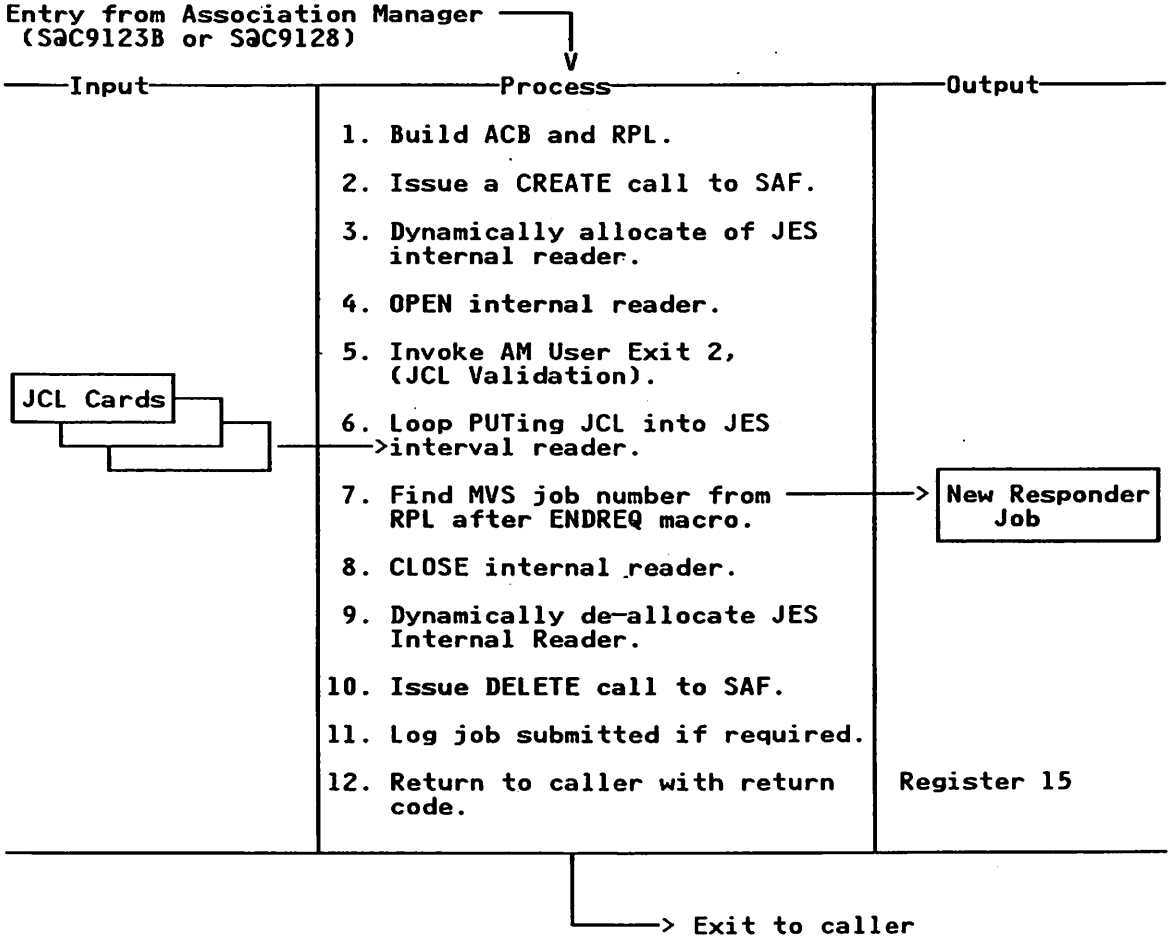
Diagram 8-17
S@C914R - Send A-ASSOCIATE (Negative Responses)



Extended Description

Explanation	*Module*	*Label*
2. Initially, S@C914R will search for a responder directory entry associated with the processor. If it finds one, S@C914R will reject all the connection end points on its queue. If it does not find one, it will search for a connection end point in the ARE area in private storage; if found, it will be rejected.	S@C914R	
4. S@C914R sends an A-ASSOCIATE response negative. An appropriate message is output to indicate which connection end point has been rejected. Finally, the PRB/PRC buffer is released back to its cell pool.	S@C914R	
7. Each individual end point on the queues are rejected and then cleaned up in the same manner as the individual end point in step 4. The end points are removed from the queues using the S@C@QREM macro. This has the effect of releasing the queue element as they are released back to the queue element cell pool.	S@C914R	

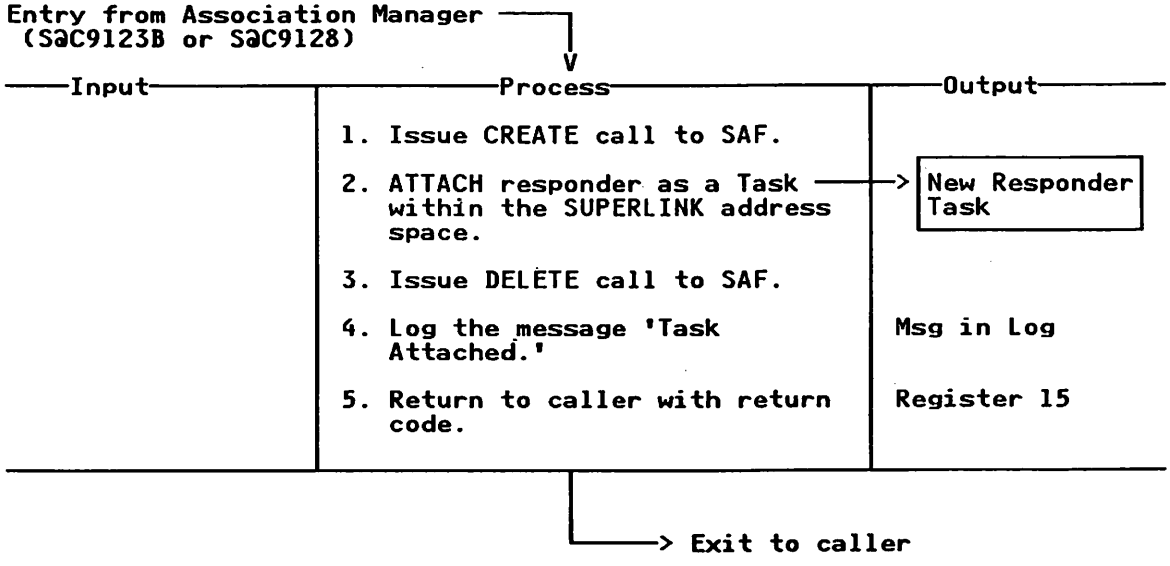
Diagram 8-18
S@C914S - Job Submission Processor



Extended Description

Explanation	*Module*	*Label*
2. The SAF Create is performed via module S@C914X The parameters passed to SAF are those for RACINIT; TYPE = CREATE, USER, GROUP, and PASSWORD. If the 'bypass password checking' flag is set, then PASSCHK = NO will be used.	S@C914S	
6. The cards inserted have been previously set up in an output area by a call to module S@C914J.	S@C914S	
7. An ENDREQ macro is issued to terminate the insertion of the JCL into the JES internal reader. This puts the job number of the job into the RPL. The job number will be extracted from the RPL and saved.	S@C914S	
10. The SAF Delete is performed via module S@C914X.	S@C914S	
11. If Association Manager tracing is on, then the JCL submitted will be logged in the SUPERLINK LOG. Any occurrences of PASSWORD= on any card will be suppressed.	S@C914S	

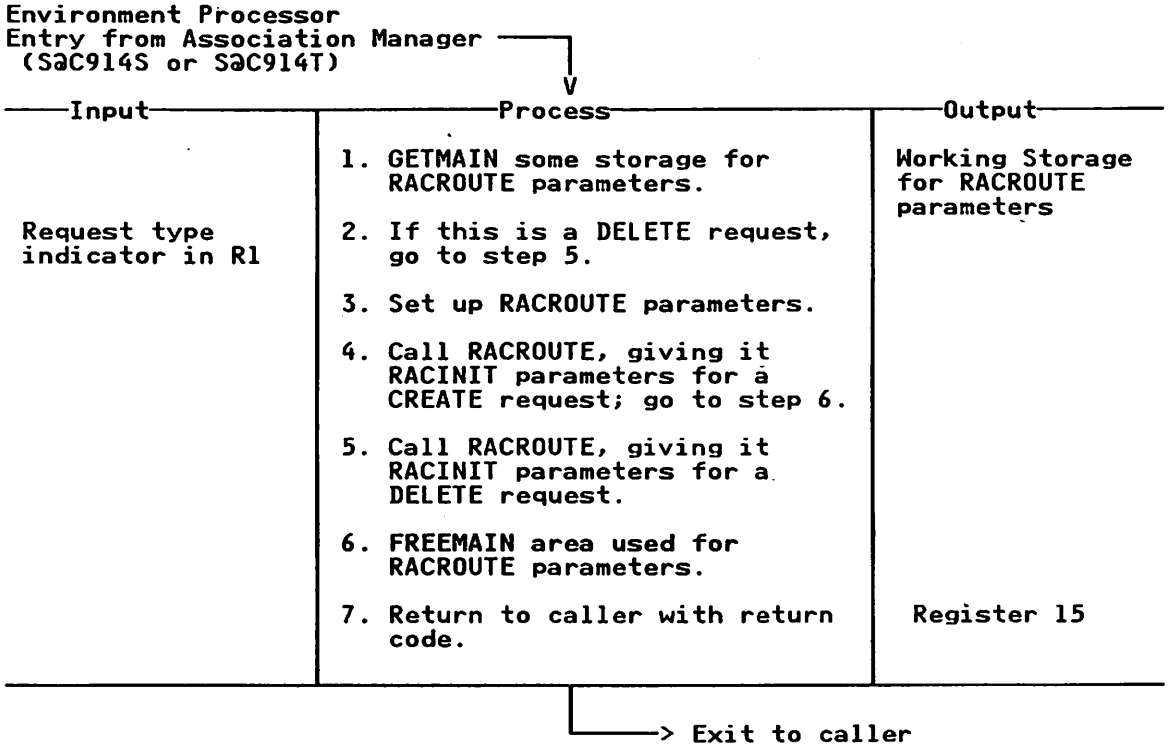
Diagram 8-19
S@C914T - Task ATTACH Processor



Extended Description

Explanation	*Module*	*Label*
1. The SAF Create call is made using module S@C914X.	S@C914T	
2. The ATTACH is done with an ECB so that the SUPERLINK Association Manager component can pick it up in the event of an abnormal termination.	S@C914T	
3. The SAF Delete call is made using module S@C914X.	S@C914T	

Diagram 8-20
S@C914X - Create/Delete System Authorization Facility (SAF)



Extended Description

Explanation	*Module*	*Label*
1. The RACROUTE parameters must be below the line. Since normal working storage is above the line, another GETMAIN is required.	S@C914X	
3. The parameters passed to RACROUTE for the SAF create request are handled as follows: • User ID - this is mandatory, it must be resolved. • Group ID - this is optional, if not resolved a null value will be used. In this instance the default Group Id for the user will take effect. • Password - this is mandatory if the "bypass password checking" flag is not set. If the flag is set, then it will be ignored.	S@C914X	

S@C9200 - Association Manager Interface

The Association Manager interface subcomponent (AM_interface) enables application entity responders that have been initiated by the Association Manager to interface with the Association Manager through the S@@MREQ macro. This subcomponent is loaded by SUPERLINK/MVS and may be used to perform interface requests.

The AM_interface communicates asynchronously with AM_processor tasks running in SLCN. The AM_interface routines are event-driven and use an FSM strategy to perform the preceding functions for the interface caller. The following states are used:

State	*Description*
IDLE	Waiting for a LISTEN request from a responder
LISTEN-PENDING	A LISTEN request was received from a companion job. The Association Manager issues A-ENDPOINT-GIVE service request primitives to pass the queued connections to the responder.
LISTEN	A LISTEN request was received from a companion job and all connection end points have been given to the responder. The companion job should now be processing queued events in FIFO order.
STOP-NEW-DATA	A DELETE TYPE = EP request was received from the companion job. The companion job may now receive only termination notifications from the Association Manager component.
TERM-ABORT	The Association Manager component has informed the companion job that an "abort" is in progress.
TERM-GRACE	The Association Manager component has informed the companion job that "graceful termination" is in progress; all system activity is being halted.
TERM-QUICK	The Association Manager component has informed the companion job that a "quick termination" is in progress.

S@C9200 - Module Structure

The AM_interface consists of only one module, S@C9200.

S@C9200 - Services

The AM_interface allows application entity responders to perform the following services via the S@@MREQ macro:

- LISTEN - The responder informs AM it is active and ready.
- NOTIFY -The responder requires notification of an event via the POST mechanism.
- PROCESS - The responder requests delivery of inbound data
- DELETE-EP - The responder is not receptive to subsequent events, with the exception of termination requests.

- DELETE-ANY - The responder is not able to accept subsequent events.

Valid inbound or outbound events are as follows:

- Connection end point given
- Termination order:
  - Graceful
  - Quick
  - Abort

  See "Association Manager Services Offered to Application Entity Responders on MVS" on page 8-4 for definition of these termination orders.

S@C9200 - Interfaces

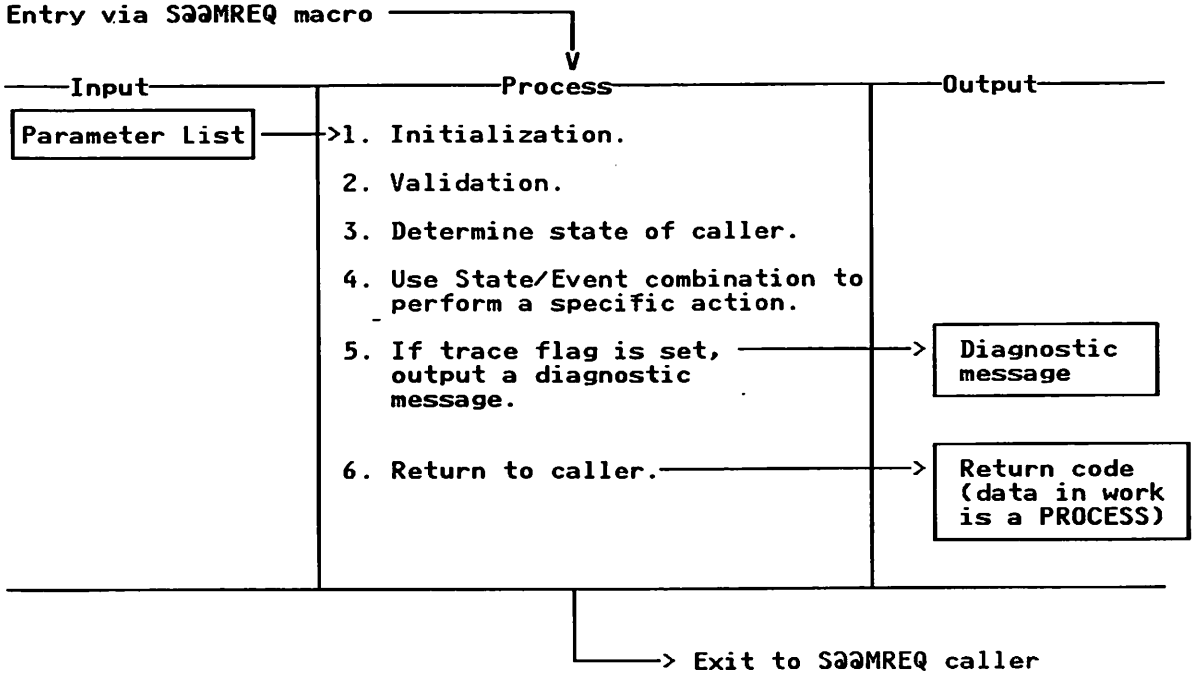
The main external interface to the AM_interface is the S@@MREQ macro. "Appendix B. SLCN Macros" on page B-1 describes the syntax of the S@@MREQ macro.

See the general discussion of interfaces under "Association Manager Interfaces" on page 8-4 for additional information.

S@C9200 - Data Areas

The Association Manager data areas are discussed under "S@C9000 - Data Areas" on page 8-9.

Diagram 8-21
S@C9200 - Association Manager Interface Code



Extended Description

Explanation	*Module*	*Label*
1. A work area is acquired, and the trace flag is saved for use upon termination.	S@C9200	
2. For the call to the interface code to proceed, the following must be true: • The SSVT address must be valid. • The SUPERLINK Association Manager component must be active. • The S@@MREQ macro's parameters must be valid, as follows: ▪ The ID parameter must be specified for all types (except the initial LISTEN request), and it must contain the identifier returned to the caller in register 1 after the initial LISTEN request. ▪ If the NOTIFY function was requested, the caller must use the ECB parameter to specify the address of an ECB. ▪ If the PROCESS function was requested, the caller must use the WKAREA parameter to specify the address of a work area.	S@C9200	
3. The STATE of the caller is determined by examining the responders status flags in its entry in the Responder Directory.	S@C9200	
4. Upon entry to the state/event machine one of the following action routines will be performed: **Action-R** Invalid state/event combination. Set up a response code indicating "request rejected" with a qualifier indicating the state. **Action-1** Search for the entry in the REDE; if it is not found, create a new entry. Then set the state to "LISTEN". **Action-2** Set the state from "STOP-NEW-DATA" back to "LISTEN". **Action-3** Post the user's ECB when there is an event for it to process. **Action-4** Insert details of an event (if there is one) into the user's work area. **Action-5** Set the state to STOP-NEW-DATA, inhibiting all events except termination requests. If there was anything on the queue, pass it back to the Association Manager controller subcomponent for handling. **Action-6** Set the state to IDLE and inform the Association Manager processor subcomponent that a "DELETE ANY" has been successfully performed.	S@C9200	
5. If the Trace flag was specified in options, a diagnostic message is written to the SUPERLINK LOG.	S@C9200	
6. The user is given a return code in register 15, with a qualifier in register 0.	S@C9200	

S@C9300 - Association Manager Interval Timer

The Association Manager interval timer (AM_timer) provides an interface between the Association Manager and the standard MVS timer facilities. Upon completion of a specified time interval, a specified ECB will be POSTed. Each Association Manager task may have one active timer. That timer can be SET and left to expire (POST an ECB), or can be cancelled before it expires. It uses the STIMER macro internally.

The timer service can be used only by the Association Manager component. It is loaded at initialization by the AM_controller and is invoked from the the S@C@TIMR macro.

S@C9300 - Module Structure

The AM_interval timer facility consists of the following modules:

Module	*Description*
S@C9300	Root module; mainline routine.
S@C9310	Timer expired routine

Figure 17 shows the hierarchical structure for modules within the AM_timer subcomponent.

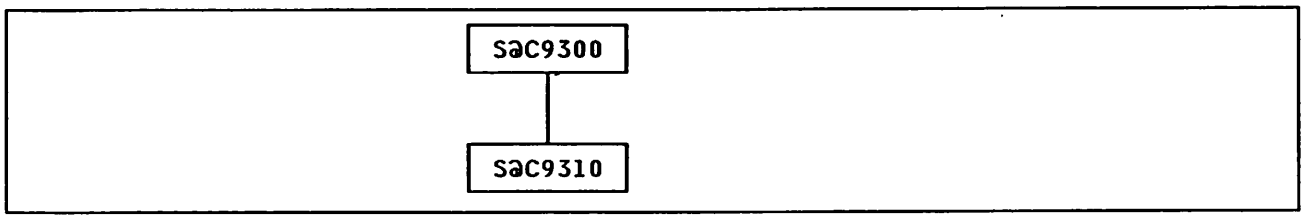


Figure 17. Module Structure of the Association Manager Interval Timer Subcomponent

S@C9300 - Services

The AM_timer performs the following functions:

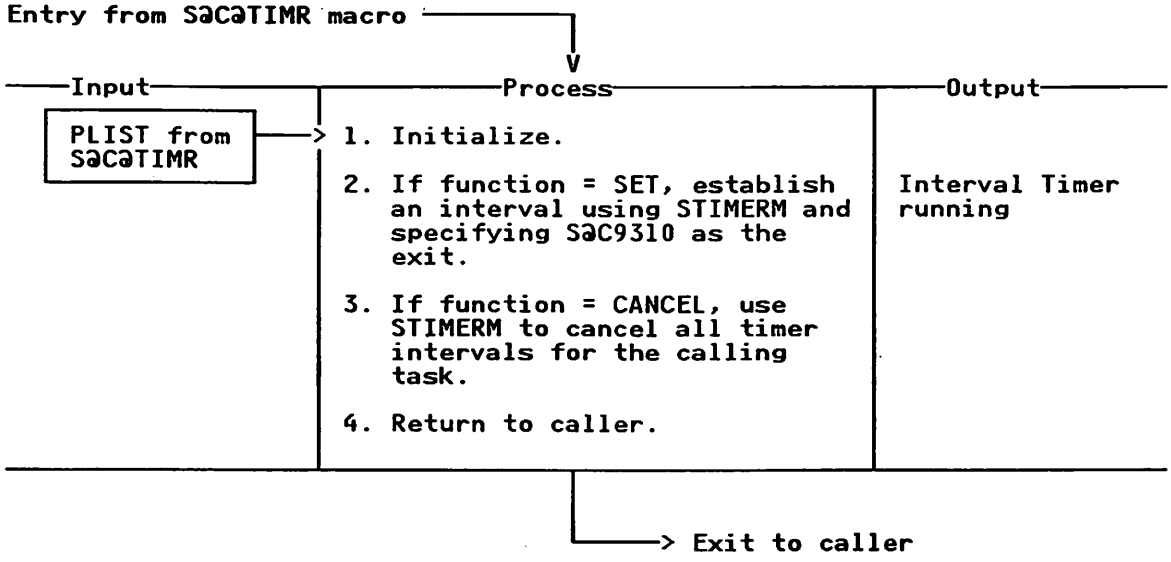
- Initiates a timing sequence, specifying a real-time interval in seconds (for the calling task only)
- Cancels any timing sequences that are running (for the calling task only)
- Notifies the requester when the real-time interval has expired

S@C9300 - Interfaces

AM_timer services are provided through specification of the S@C@TIMR execution-time macro instruction. "Appendix B. SLCN Macros" on page B-1 describes the syntax of the S@C@TIMR macro.

This page has been intentionally left blank.

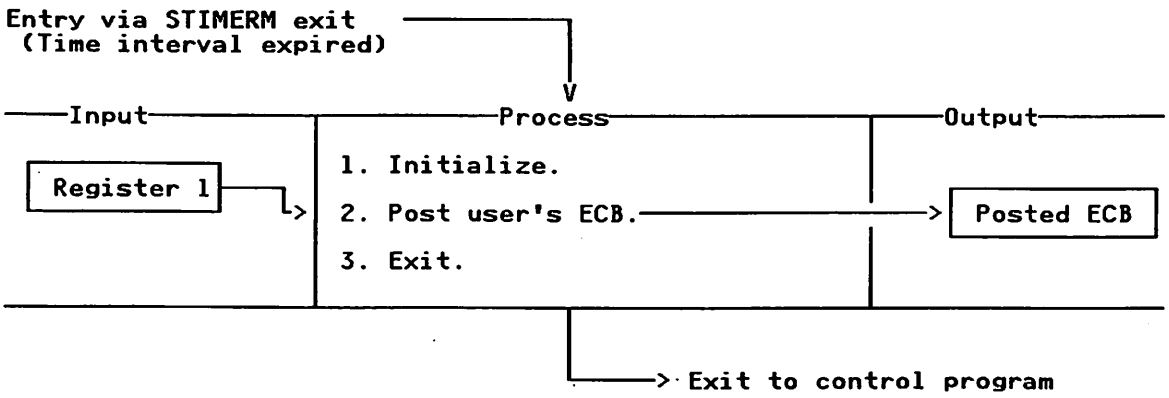
Diagram 8-22
S@C9300 - Root Module (Set and Cancel Timer)



Extended Description

Explanation	*Module*	*Label*
1. No storage is acquired. The caller must provide a work area through register 13. Register 13 should point to the caller's register save area (72 bytes), which is followed by a 20-word work area for use by the interval timer. (The register save area is not used).	S@C9300	
2. The STIMERM macro is used to establish the S@C9310 routine. It becomes active when the interval specified has expired and POSTs the user's ECB.	S@C9300	
3. The STIMERM macro is used to perform a universal CANCEL for all interval times established by the calling task.	S@C9300	
4. The caller is given a return code to indicate success or failure.	S@C9300	

Diagram 8-23
S@C9310 - Timer Expired Routine



Extended Description

Explanation	*Module*	*Label*
2. The user's ECB is supplied to the routine through the 8-byte area pointed to by register 1. The ECB is POSTed with a return code of "20".	S@C9310	
3. The address in register 14 is the return address to the control program.	S@C9310	

S@C9UXAM - Association Manager User Exit Handler

The Association Manager User Exit handler (AM_exit) provides an interface between the Association Manager and any user supplied exit routines. The objective is to isolate the user exit from Association Manager processing. The Exit Handler intercepts abends within the user exit and returns control to the Association Manager with an appropriate return code indicating the nature of the abend.

The User Exit Handler also isolates user exits from the Association Manager's storage by passing the exit parameters in private storage, (the private storage is obtained by the Exit Handler through the GETMAIN macro). This prevents a user exit corruption of Association Manager control blocks.

The Exit Handler may only be used by the Association Manager component. It is loaded at initialization by the AM_controller and is invoked by the S@C9EXIT macro.

S@C9UXAM - Module Structure

The User Exit Handler consists of one module, S@C9UXAM.

S@C9UXAM - Services

The Association Manager User Exit Handler performs the following functions:

- Determines if the user exit has been loaded. If not, control is returned with return code zero. If it does exist them it is invoked.
- Obtains private storage below the 16 megabyte line and establishes the required parameters for the exit.
- Ensures that the exit is invoked in problem program state
- Establishes an ESTAE environment for the duration of the exit call to recover from any abends within the user exit.
- Issues a message if the return code from the user exit is nonzero. Any text message returned by the exit is also output to the SUPERLINK LOG.

S@C9UXAM - Interfaces

The Association Manager User Exit Handler may be invoked from any location within the Association Manager by using the S@C9EXIT macro. The parameters passed are the user exit required, (1, 2, 3, or 4) and the type of call being made (1 for initialization, 2 for normal, or 3 for termination).

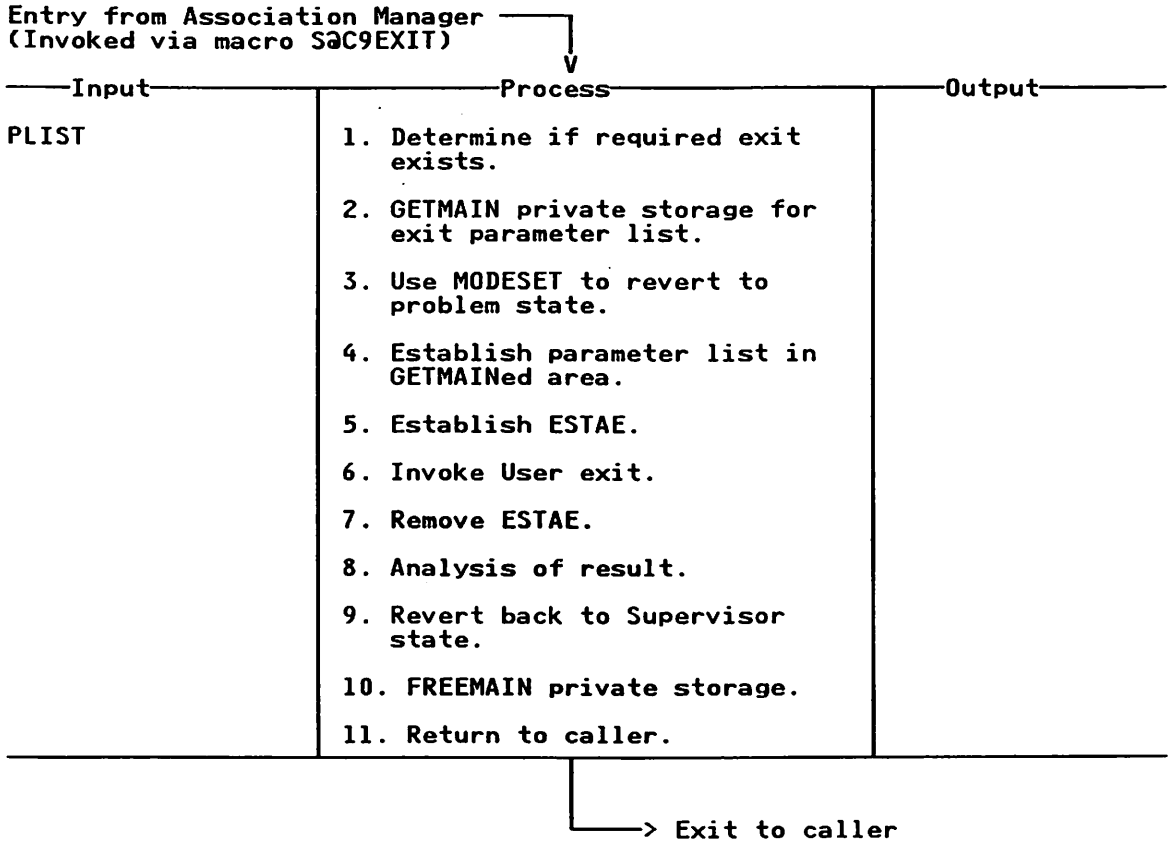
For each of the Association Manager User Exits an appropriate mapping macro is provided that defines the parameter list of each user exit. The relationship between user exits and macros is as follows:

User Exit	*Macro*
AM user exit 1	S@C9UX1
AM user exit 2	S@C9UX2
AM user exit 3	S@C9UX3
AM user exit 4	S@C9UX4

The SUPERLINK/MVS Installation, Tuning, and Customization Guide, publication SI-0180, provides more detailed information about the user exits and their respective macros.

This page has been intentionally left blank.

Diagram 8-24
S@C9UXAM - User Exit Handler



Extended Description

Explanation	*Module*	*Label*
1. The User exits are loaded by the Association Manager component if they are specified within the SUPERLINK options deck. If they were not loaded, control is returned to the caller with return code 0.	S@C9UXAM	
2. The storage GETMAINed is below the line. This will enable user exits to be in either AMODE 24 or AMODE 31.	S@C9UXAM	
4. The contents of the parameter list varies depending upon the exit being invoked. The type of call being made can also vary the parameter list. If invoked during Association Manager component initialization or termination, some of the parameters will not be available; This means that the pointers to them in the parameter list will be null. Only AM User Exit 4 (Security Validation) may be invoked during AM initialization or termination. This enables the exit to establish and remove any tables or storage it requires for normal validation calls.	S@C9UXAM	
5. The ESTAE provided resides at the end of the S@C9UXAM module. If invoked by an abend, it outputs an appropriate abend message and returns control to the main section of S@C9UXAM. S@C9UXAM FREEMAINs storage and returns control to the caller.	S@C9UXAM	
8. If a non-zero return code is received from the exit, an appropriate message is output to flag the non-zero return code. If the exit also returned a text message, (if a text area was provided for the exit) then this text message is also output.	S@C9UXAM	

9. SUPERLINK Message Processor Component

The Message Processor component of SLCN provides one source for both the documentation of messages and the generation of macros used to describe the messages within the SUPERLINK/MVS product. This source is maintained in generalized markup language (GML) format. The Message Processor component is designed to maintain consistency between the messages issued by the product and the message descriptions provided by SUPERLINK/MVS Messages, publication SI-0179.

The macros generated by GML processing produce nonexecutable CSECTs containing the message text available to the SUPERLINK/MVS product.

Message Processor Module Structure

The Message Processor component consists of the following modules:

Module	*Description*
S@@M000	Initial processing; locate the message.
S@@M010	Format the variables into the message.
S@@M015	Return a formatted message to the user, if requested.
S@@M020	Perform the TYPE = WTO or TYPE = WTOR request.
S@@M030	Perform the TYPE = LOG request.

Figure 18 shows the hierarchical structure of modules within the Message Processor component.

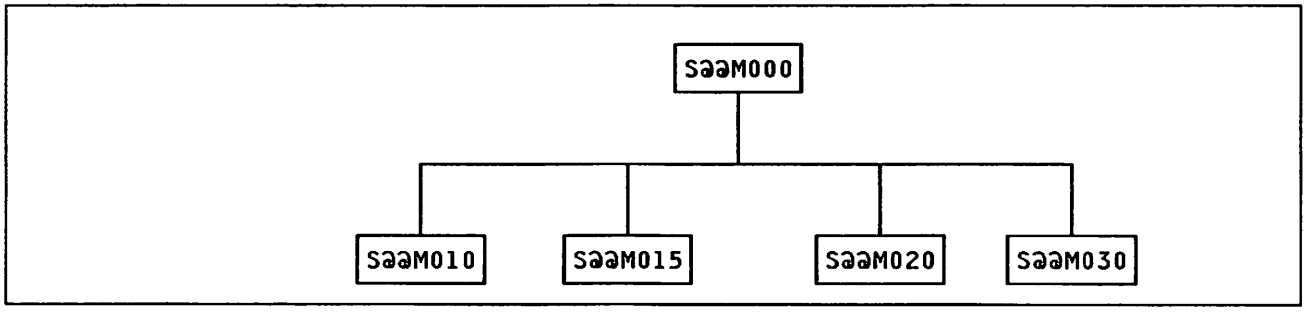


Figure 18. Module Structure of the Message Processor Component

Message Processor Services

The Message Processor provides the following services:

- Returns the address of a message within a message CSECT
- Returns a formatted message to an area provided by the caller
- Formats a message and outputs it to the MVS system log, the SUPERLINK LOG, or both.
- Formats a message and issues a WTOR for an operator reply, which is returned to an area provided by the caller.

Message Processor Interfaces

The following macros provide the interface to the Message Processor component.

Macro	*Description*
S@@MDEF	This macro is used to create the messages CSECT. The S@@MDEF macros are generated by DCF/SCRIPT from GML tags.
S@@MSGS	This macro is used to locate a particular message within a specified message table.
S@@MSG	This is the major macro of the Message Processor component. It calls the Message Processor component to perform various functions, including formatting a message, outputting it to a specific target, and so on. The S@@MSG macro has four forms: ordinary, list, format, and execute.

"Appendix B. SLCN Macros" on page B-1 describes the syntax of each of these macros.

Message Processor Data Areas

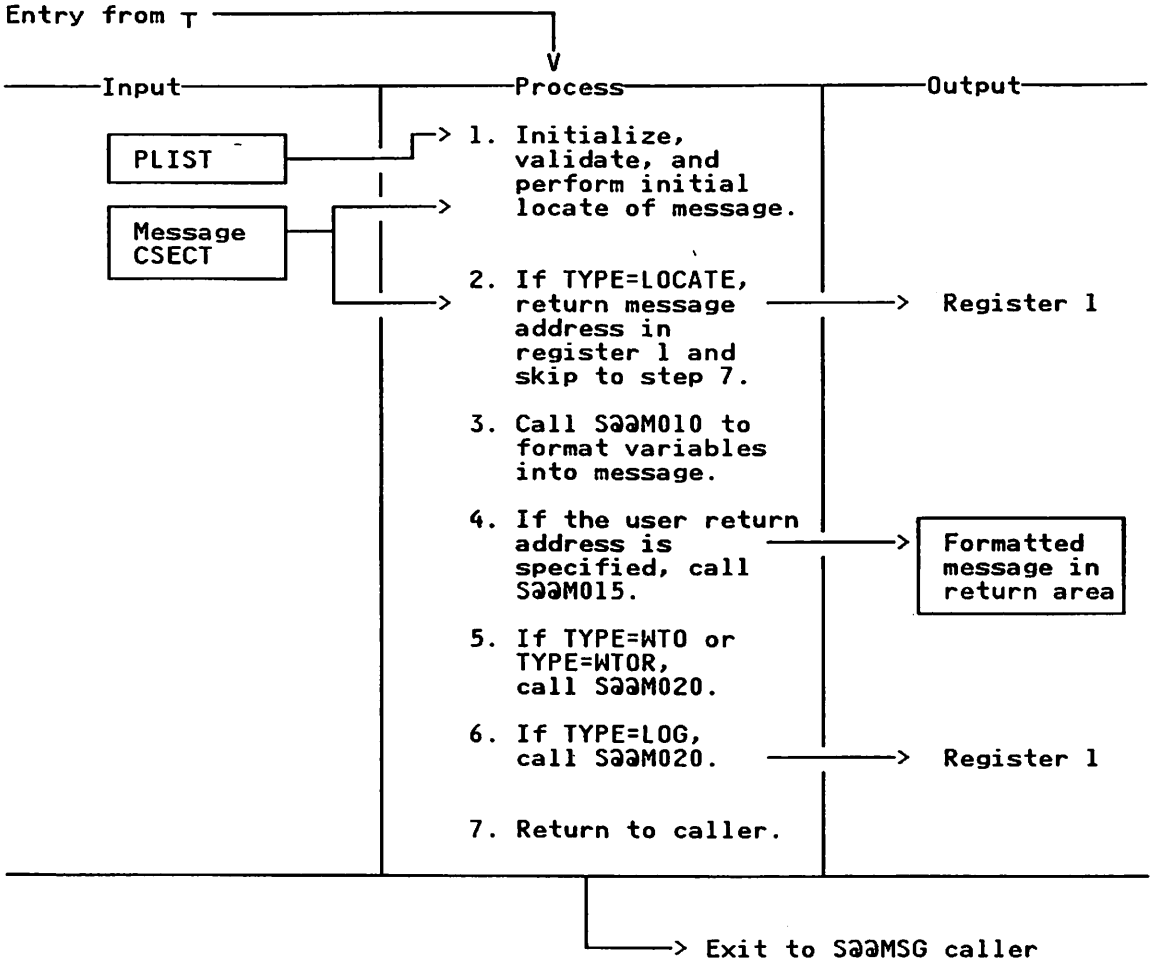
The following data areas, used by the Message Processor component, are contained within the message CSECTs:

Data Area	*Description*
MH_MI	Message index
MH_MIE	Message index entry
MH_ME	Message Element
MH_MPPL	Message processing parameter list

Message Processor Recovery

The Message Processor component does not perform its own recovery. All recovery actions are determined by the recovery environment enabled by the caller of the SUPERLINK/MVS Message Handler.

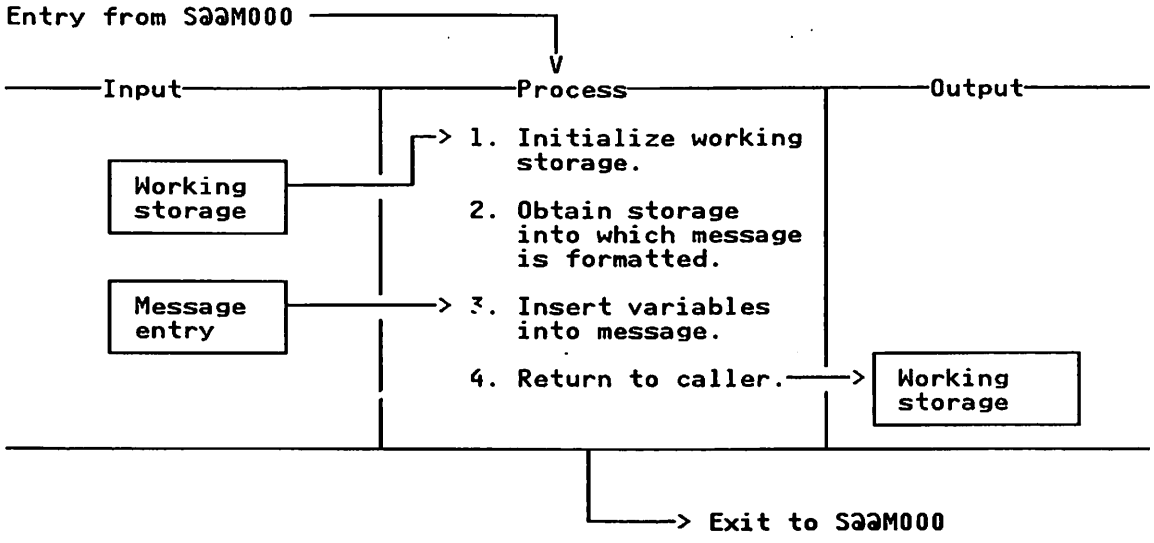
Diagram 9-1
S@@M000 - Initial Message Processing Module



Extended Description

Explanation	*Module*	*Label*
1. The initial validation of the parameters received is carried out at this stage. If TYPE = LOCATE, proceed to step 2. For all other message types, a GETMAIN is executed at this stage. The storage acquired is used as a save area and as working storage; the working storage used follows the save area. The parameters are set up in this T-storage area, and all work areas to be used are initialized. As control processes through the various modules, the address of this area is passed in register 13. The parameters are now validated. If they are all correct, the message is located within the Message Table using a binary search. A binary search jumps to the middle of the index and determines if the message lies above or below that point. This process is repeated with successively smaller fractions of the index until a match is made.	S@@M000	
2. If TYPE = LOCATE, the TABLE parameter is validated, and the requested message is located within the Message Table. The address of the message is set up in register 1, and a return is made to the caller.	S@@M000	
3. S@@M010 is now used to format the message. If TYPE = WTOR or TYPE = WTOR + LOG, the message is formatted into a WTOR parameter list; otherwise, the message is formatted into a WTO parameter list.	S@@M000	
4. If the user requested it, S@@M015 now inserts the formatted message into the user's return area.	S@@M000	
5. If TYPE = WTO or WTOR, S@@M020 is called to perform the required processing.	S@@M000	
6. For TYPE = LOG, S@@M030 is called to perform the required processing. Register 1 contains the address of the message that is to be output.	S@@M000	

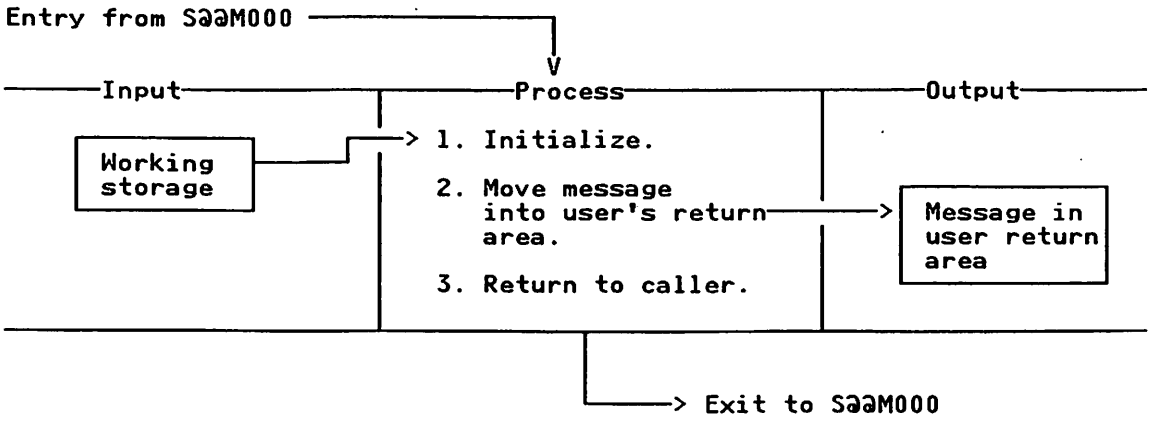
Diagram 9-2
S@@M010 - Format the Message Variables Module



Extended Description

Explanation	*Module*	*Label*
1. Initially, register 13 points to the save area, which is followed by the working storage. The working storage section uses register 11 as a base register, and register 13 points to the secondary save area.	S@@M010	
2. If TYPE = WTO or WTO + LOG, and the message has multiple lines, the required amount of storage is acquired by a GETMAIN. For all other types, the message is formatted into an area in working storage large enough to hold a WTOR parameter list of maximum size.	S@@M010	
3. The message is moved from the CSECT into the target area, and the variables are inserted into the message text. For TYPE = WTOR and TYPE = WTOR + LOG, the address of the formatted message in working storage points to a WTOR parameter list containing the message text. For all other types, the address points to a WTO parameter list containing the message text.	S@@M010	
4. Control is returned to the caller. The address of the formatted message is contained in the working storage section used.	S@@M010	

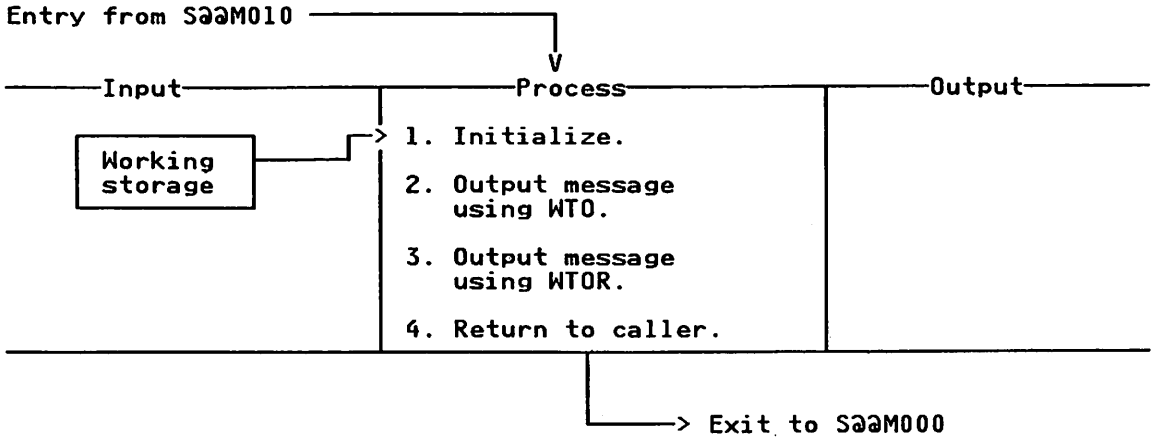
Diagram 9-3
S@@M015 - Formatted Message Returned to User Module



Extended Description

Explanation	*Module*	*Label*
1. Initially, register 13 points to the save area, which is followed by the working storage. Register 11 is used as a base register for the working storage section, and register 13 points to the secondary save area. If the user's return area address is null, go to step 3.	S@@M015	
2. The message is moved into the user's return area. A space is inserted between each character of a multiline message. If the length of data specified by the user at the start of the buffer is reached, the process ends, giving the user as much of the message as possible.	S@@M015	
3. Control is returned to the caller.	S@@M015	

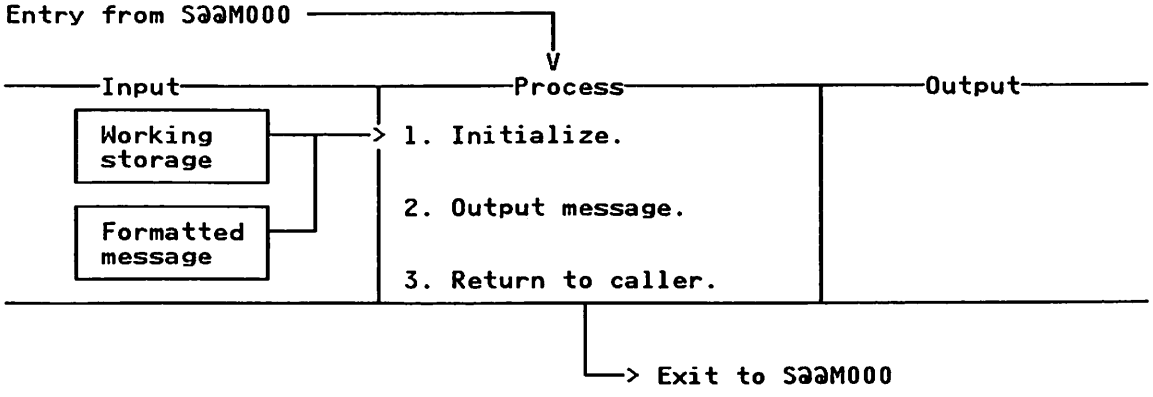
Diagram 9-4
S@@M020 - Write to WTO or WTOR Module



Extended Description

Explanation	*Module*	*Label*
1. Initially, register 13 points to the save area, which is followed by the working storage. Register 11 is used as a base register for the working storage section and register 13 points to the secondary save area.	S@@M020	
2. If TYPE=WTO or WTO+LOG, the execute form of the WTO macro outputs the message using the WTO parameter list that is pointed to.	S@@M020	
3. If TYPE=WTOR or WTOR+LOG the execute form of the WTOR macro outputs the message using the WTOR parameter list that is pointed to.	S@@M020	
4. A return is made to the caller, S@@M000.	S@@M020	

Diagram 9-5
S@@M030 - Write to LOG Module



Extended Description

Explanation	*Module*	*Label*
1. Initially, register 13 points to the save area, which is followed by the working storage. Register 11 is used as a base register for the working storage section, and register 13 points to the secondary save area.	S@@M030	
2. The required number of LOGEs is acquired. The formatted message is inserted into the LOGEs; the LOGEs are then queued for output. If TYPE=WTOR or WTOR+LOG, the text in the single-line WTOR parameter list is split over two LOGEs.	S@@M030	
3. A return is made to the caller, S@@M000.	S@@M030	

10. SUPERLINK SVC Component

In order to use the services offered by the SUPERLINK Network Access Method, the calling program is required to be in supervisor mode. For assembler programmers, this privilege can be assigned by the systems programmer. However, user programs typically run in problem program mode.

The SUPERLINK SVC component enables nonauthorized, problem program mode users to use Network Access Method services (SLNET) or Association Manager services (SLCN) of the SUPERLINK/MVS product. Specifically, this component allows callers from high-level languages (such as Fortran) to use the AAC interface and allows unauthorized assembler programs to use the ACSE interface. It does not allow the user's program to access any other services, and control is returned to the caller in the original processing mode, thus preserving MVS system integrity.

SVC Module Structure

The SVC component consists of the following modules:

Module	*Description*
S@CC0SVC	SVC routine
S@CC0SVE	ESTAE exit routine
S@CC0SVR	Retry routine
S@CC0SVM	Messages CSECT

Figure 19 shows the hierarchical structure of modules within the SVC component.

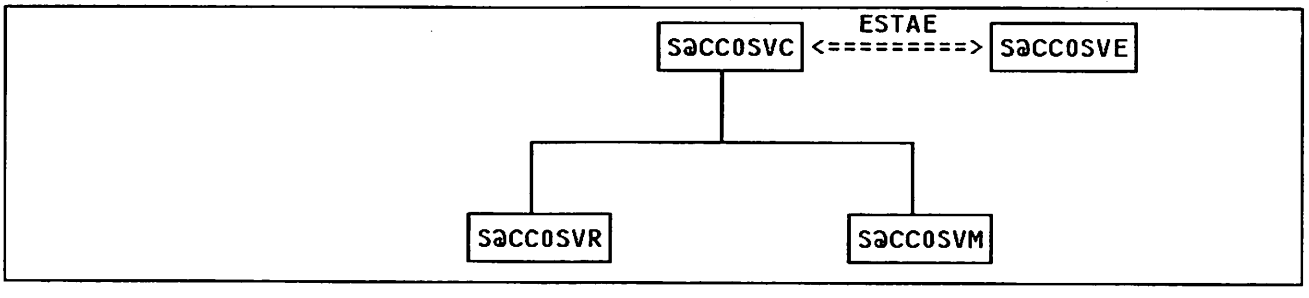


Figure 19. Module Structure of the SVC Component

SVC Services

The following global service routines can be accessed through the SVC component:

- Association Manager interface (SLCN)
- ACSE interface (SLNET)

The SVC component has been designed to satisfy the following requirements:

- Maintain MVS system integrity (MVS/XA System Programming Library: System Macros and Facilities Volume 1, GC28-1150, Protecting the System)
- Minimize system overhead; the SVC component uses branch entry services.
- Provide a universal interface so that new SUPERLINK services can easily be made accessible via the SVC component

SVC Interfaces

The S@@SVC macro is used to pass the parameters required by the SVC component. The first parameter is positional; all others are keyword parameters.

The SC_CIOT (S@C1CIOT mapping macro) must be addressable when the S@@SVC macro is issued. The SVC number is obtained from this control block. "Appendix B. SLCN Macros" on page B-1 describes the syntax of the S@@SVC macro.

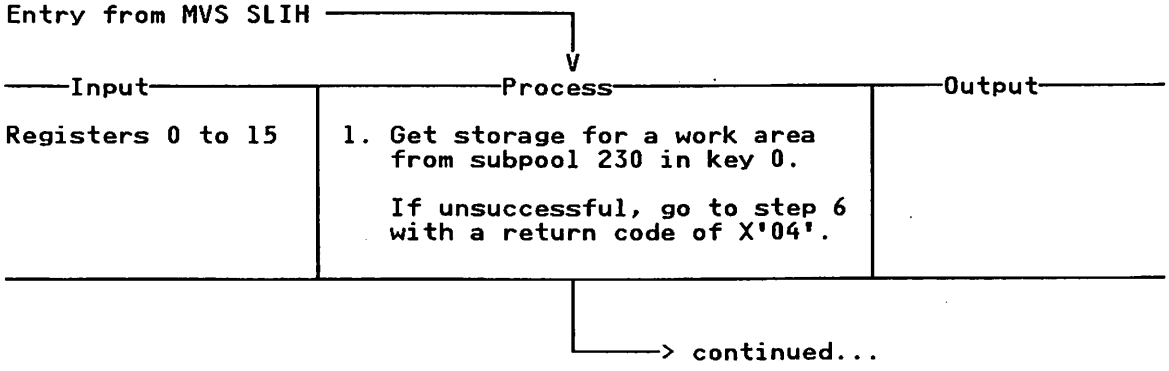
SVC Data Areas

The following data areas are used by the SVC component:

Data Area	*Description*
SV_ESTW	SVC ESTAE work area
S@CCSVEW	Mapping macro

This page has been intentionally left blank.

Diagram 10-1
S@CC0SVC - SVC Type 3 Routine (part 1 of 2)



Extended Description

Explanation	*Module*	*Label*
1. The SVC main routine receives control in PSW key 0, in supervisor state, enabled, and unlocked. Register contents at entry are as follows:	S@CC0SVC	

Register *Contents*

R0 This register contains the following bytes:

Byte	*Description*
0	Index in the SC_GST of the global service routine to be called
1	If byte 3, bit 0 is set, this register contains the offset of the SC_SSVT in the parameter list pointed to by R1.
2	Reserved
3	Option byte: Bit 0 - When this flag is set, the SC_SSVT pointer is in the entry parameter list. Otherwise the SC_SSVT pointer is contained in R15. Bit 1 - When this flag is set, an ESTAE exit is to be established.

R1 Address of the parameter list to be passed to the global service routine

R2 Unpredictable

R3 Address of the CVT

R4 TCB address

R5 SVRB address

R6 Address of the SVC routine entry point

R7 ASCB address

R8-R12 Unpredictable

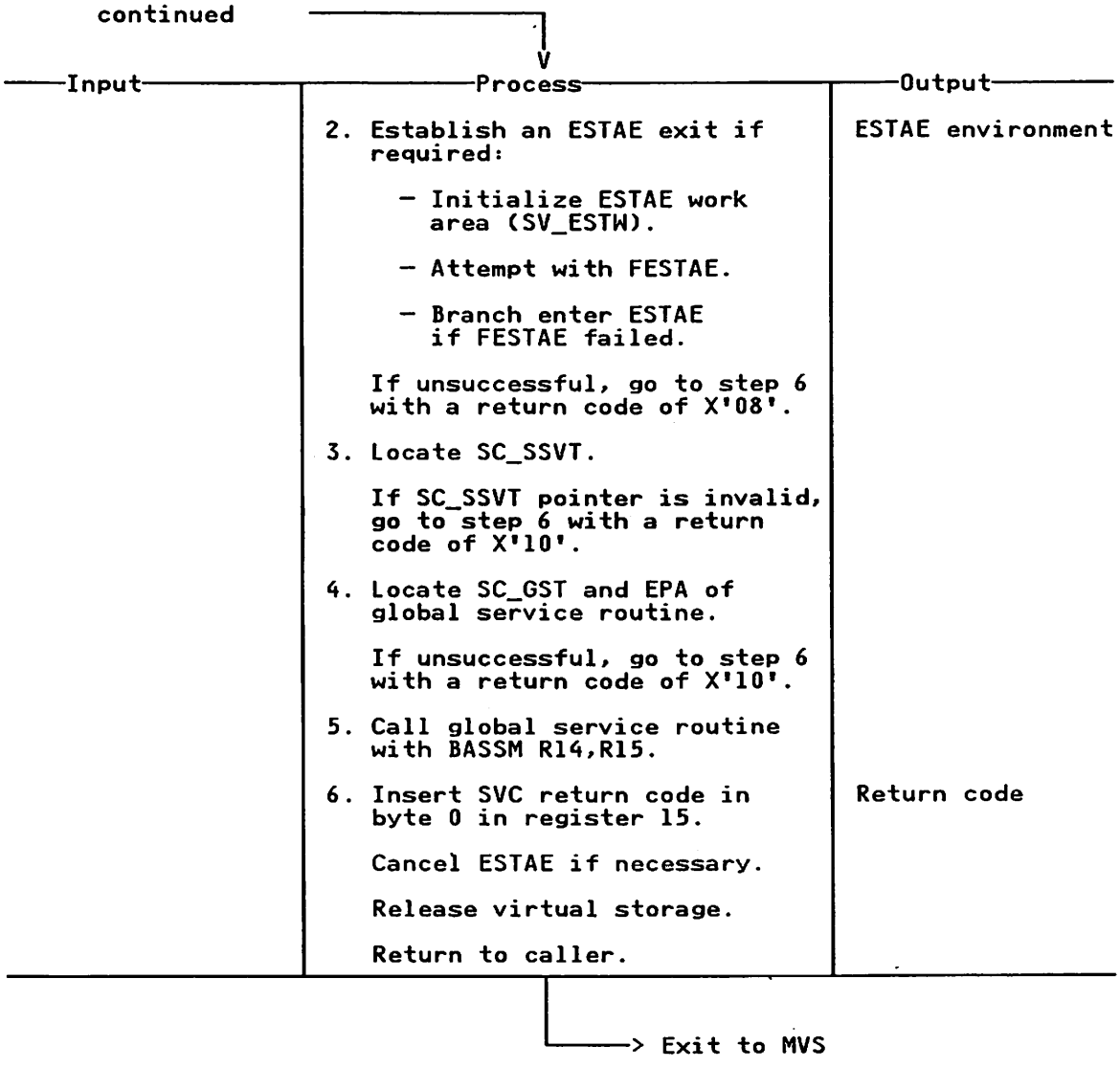
R13 Contents when the SVC instruction was executed

R14 Return address

- If byte 3, bit 0 in register 0 is on, content is irrelevant.
- If byte 3, bit 0 in register 0 is off, R15 contains the address of the SC_SSVT.

Storage for a work area is obtained from subpool 230 in key 0. If GETMAIN failed, processing continues at step 6 with a return code of X'04'.

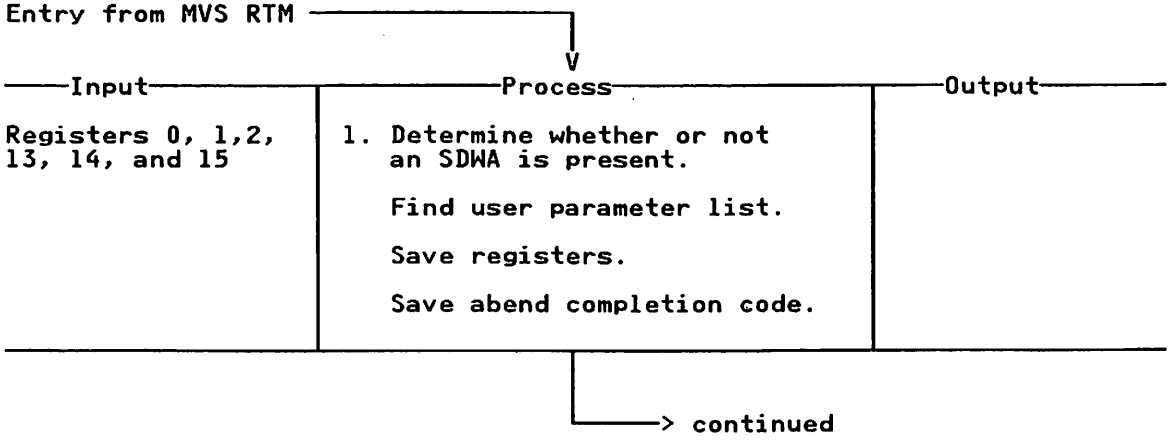
Diagram 10-2
S@CC0SVC - SVC Type 3 Routine (part 2 of 2)



Extended Description

Explanation	*Module*	*Label*
2. On entry, byte 3, bit 1 in register 0 indicates whether or not an ESTAE environment should be established by the SVC. If an ESTAE is required, the following actions are required: • The ESTAE work area, SV_ESTW, is initialized. • A fast ESTAE (FESTAE) macro instruction is issued first. If FESTAE fails, the branch entry interface of ESTAE is used. • If the ESTAE environment cannot be established, processing continues at step 6 with a return code of X'08'.	S@CC0SVC	
3. On entry, byte 3, bit 0 in register 0 indicates whether the SC_SSVT pointer is contained in register 15 or in the entry parameter list. The contents upon entry of registers 0, 1, and 15 have been saved in the SVRB. If the SC_SSVT pointer is not in register 15, the contents of byte 1 in register 0 upon entry contain its offset in the parameter list pointed to by register. To check the validity of the SC_SSVT pointer, loop through the SSVT chain until a match is found. If no match is found, or if the corresponding subsystem is not ready, processing proceeds to step 6 with a return code of X'0C'.	S@CC0SVC	
4. The SC_GST pointer is found in the SC_SSVT. If this pointer is null, or if the SC_GST does not contain a valid acronym, processing continues at step 6 with a return code of X'10'. On entry, the index of the requested service routine in the SC_GST is in the contents of byte 0 in register 0 (saved in the SVRB). The index is checked to determine whether it is within bounds and whether or not it identifies a service routine that can be invoked through the SVC. The corresponding entry point address in the SC_GST is not valid if it is null. If no suitable entry point address has been found, processing proceeds at step 6 with a return code of X'10'.	S@CC0SVC	
5. The global service routine, whose entry point was found in step 5, is called with BASSM registers 14 and 15. Standard linkage conventions are used. Register 13 points to an 18-word save area with a storage protection key of 0. The global service routine receives control in PSW key 0, in supervisor state, enabled and unlocked.	S@CC0SVC	
6. The SVC return code is inserted into byte 0 in register 15. The ESTAE environment is cancelled if one has been established. The storage allocated for the work area is released. Registers 0, 1, 14, and 15 are restored, and control is returned to MVS.	S@CC0SVC	

Diagram 10-3
S@CC0SVE - ESTAE Exit Routine (part 1 of 3)



Extended Description

Explanation	*Module*	*Label*
1. The ESTAE-type recovery routine receives control from RTM in PSW key 0, in supervisor state, enabled and unlocked.	S@CC0SVE	

Register contents at entry differ depending on whether or not RTM can obtain an SDWA.

If an SDWA was obtained, register contents at entry are as follows:

Register	*Contents*
R0	Code indicating the type of I/O processing performed
R1	Address of the SDWA
R13	Save area address (72 bytes)
R14	Return address
R15	Entry point address of the ESTAE recovery routine

The contents of all other registers are unpredictable.

If no SDWA was obtained, register contents at entry are as follows:

Register	*Contents*
R0	Decimal 12 to indicate that an SDWA was not obtained
R1	Abend completion code
R2	Address of user-supplied parameter list
R13	Unpredictable
R14	Return address
R15	Entry point address of the ESTAE recovery routine

The contents of all other registers are unpredictable.

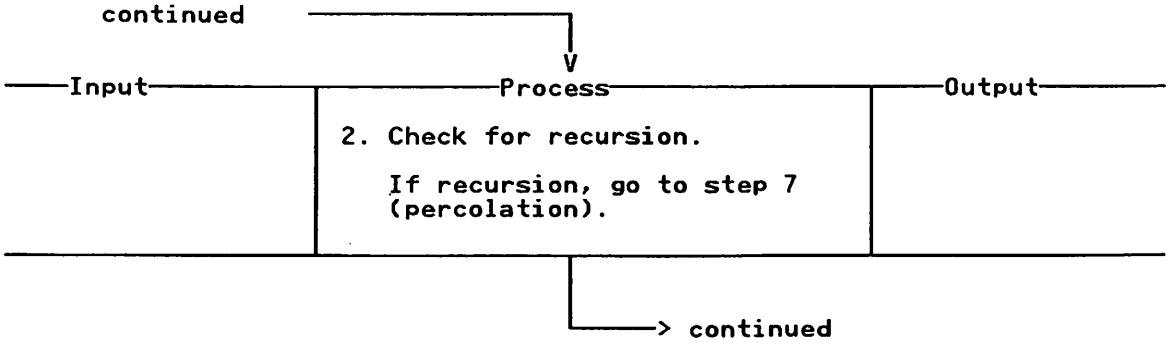
The code in register 0 is examined first to see if an SDWA was provided.

If there is an SDWA, registers 14 through 12 are saved in the save area pointed to by register 13. The pointer to the ESTAE work area, SV_ESTW, is found in the SDWAPARM field of the SDWA. The abend completion code is copied from the SDWA into the SV_ESTW for later analysis.

If there is no SDWA, register 2 points to the ESTAE work area, SV_ESTW. Registers 14 through 12 are saved in the SV_ESTW_SAVE standard save area. Register 7 is cleared to indicate that there is no SDWA. The abend completion code contained in register 1 is saved into the SV_ESTW for later analysis.

The ESTAE routine base register is established, and the save areas are chained.

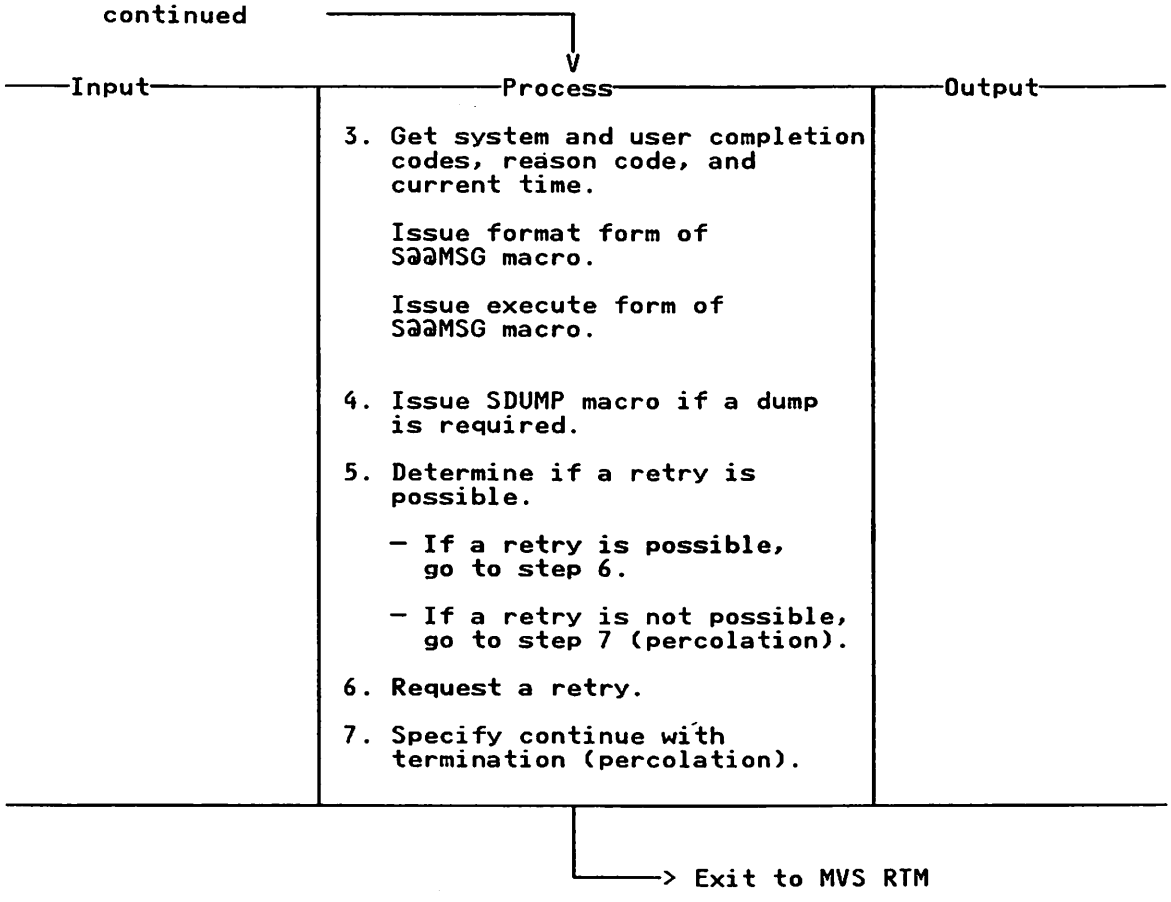
Diagram 10-4
S@CC0SVE - ESTAE Exit Routine (part 2 of 3)



Extended Description

Explanation	*Module*	*Label*
2. If bit SV_ESTW_ERECURS in SV_ESTW_FLAG is set, there is recursion. In this case, processing continues at step 7 (percolation).	S@CC0SVE	

Diagram 10-5
S@CC0SVE - ESTAE Exit Routine (part 3 of 3)



Extended Description

Explanation	*Module*	*Label*

3. The abend completion code was saved in SV_ESTW_ABENDCC in step 1 processing. — S@CC0SVE

   The reason code is found in the additional component service data extension of the SDWA when there is an SDWA.

   A TIME macro instruction is issued to get the current time.

   The preceding information is converted to printable form. If the SC_SSVT address is null, no message is issued; otherwise, the format form and the execute form of the S@@MSG macro are issued.

4. If there is no SDWA, a dump is always obtained. — S@CC0SVE

   If there is an SDWA, no dump is required if at least one of the following bits is set:

Bit	*Description*
SDWACTS	If the SDWACTS bit is on, another task within the same jobstep tree has requested a "STEP" abend.
SDWAMABD	If the SDWAMABD bit is on, an ancestor of this task has abended.
SDWAEAS	If the SDWAEAS bit is on, a dump has already been obtained.

   If the SDUMP routine executes successfully, the SDWAEAS bit is set to indicate that a dump has been obtained.

5. If the retry address in SV_ESTW_ARETRY is null, percolate. — S@CC0SVE

   If there is no SDWA, a retry is always attempted.

   If there is an SDWA, percolate if the SDWACLUP bit is set.

6. Reinstate the previous save area and test register 7 to determine whether or not an SDWA is present. — S@CC0SVE

   If there is an SDWA, use the SETRP macro to specify retry.

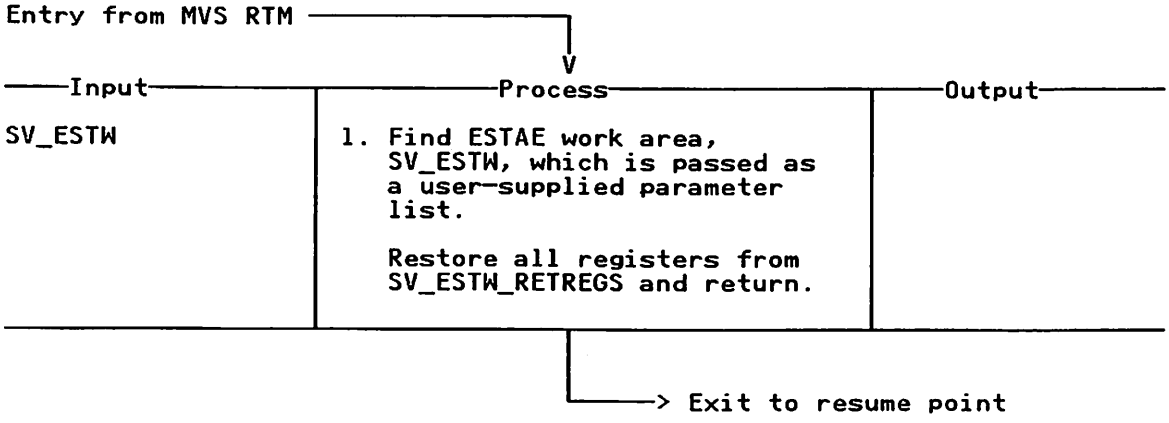
   If there is no SDWA, place retry code 4 in register 15; load register 0 with the retry routine entry point address; restore registers 1 through 12; and return to MVS.

7. Reinstate previous save area, and test register 7 to determine whether or not an SDWA is present. — S@CC0SVE

   If there is an SDWA, use the SETRP macro to specify percolation.

   If there is no SDWA, restore registers 14 through 12; place percolation code 0 in register 15; and return to RTM.

Diagram 10-6
S@CC0SVR - Retry Routine



Extended Description

Explanation	*Module*	*Label*
1. The retry routine receives control from RTM in PSW key 0, in supervisor state, enabled and unlocked.	S@CC0SVR	

Register contents at entry differ depending on whether or not RTM could obtain an SDWA and whether or not the SDWA was freed.

If the ESTAE-type recovery routine did not request register update or freeing of the SDWA, register contents upon entry are as follows:

Register	*Contents*
R0	Zero
R1	Address of the SDWA
R14	Address of supervisor-assisted linkage (SVC 3)
R15	Entry point address of the retry routine

The contents of all other registers are unpredictable.

If the ESTAE did not request register update but did request that the SDWA be freed, or if no SDWA was obtained, register contents at entry are as follows:

Register	*Contents*
R0	A decimal code as follows:

Code	*Meaning*
20	The ESTAE did not request register update but did request that the SDWA be freed.
12	No SDWA was obtained.

R1	Address of the user-supplied parameter list
R2	A pointer to the PIRL or 0
R14	Address of supervisor-assisted linkage (SVC 3)
R15	Entry point address of the retry routine

The contents of all other registers are unpredictable.

The ESTAE work area, SV_ESTW, is found directly or from the SDWA.

All registers are restored from the SV_ESTW_RETREGS field, and control is returned to the resume point through BR register 14.

11. SUPERLINK User Resource Manager Component

The User Resource Manager (URM) component of SLCN is a global service routine that provides basic functions for the manipulation of User Resource Elements (UREs) which are used to keep track of the use of SUPERLINK/MVS resources by task and ascending address space identifier (ASID).

Those services are invoked by the ACSE component of SLNET. A URE is defined for every connection end point and contains information that is essential for end-of-task processing: ASCB, task control block (TCB), and association identifier (AID). The ASCB and TCB identify the task which owns the end point. The AID uniquely identifies the connection end point.

page.User Resource Manager Module Structure

The User Resource Manager component consists of a single module:

Module	*Description*
S@CC0URM	User Resource Manager (URM) service routine

User Resource Manager Services

The following functions are provided for the manipulation of UREs:

- Locate
- Queue
- Dequeue
- Switch

User Resource Manager Interfaces

The S@CC0URM macro is used to pass the parameters required by the User Resource Manager component. The S@CC0URM macro is found in the macro library for the SLCN component.

User Resource Manager Data Areas

The following data areas are used by the User Resource Manager component:

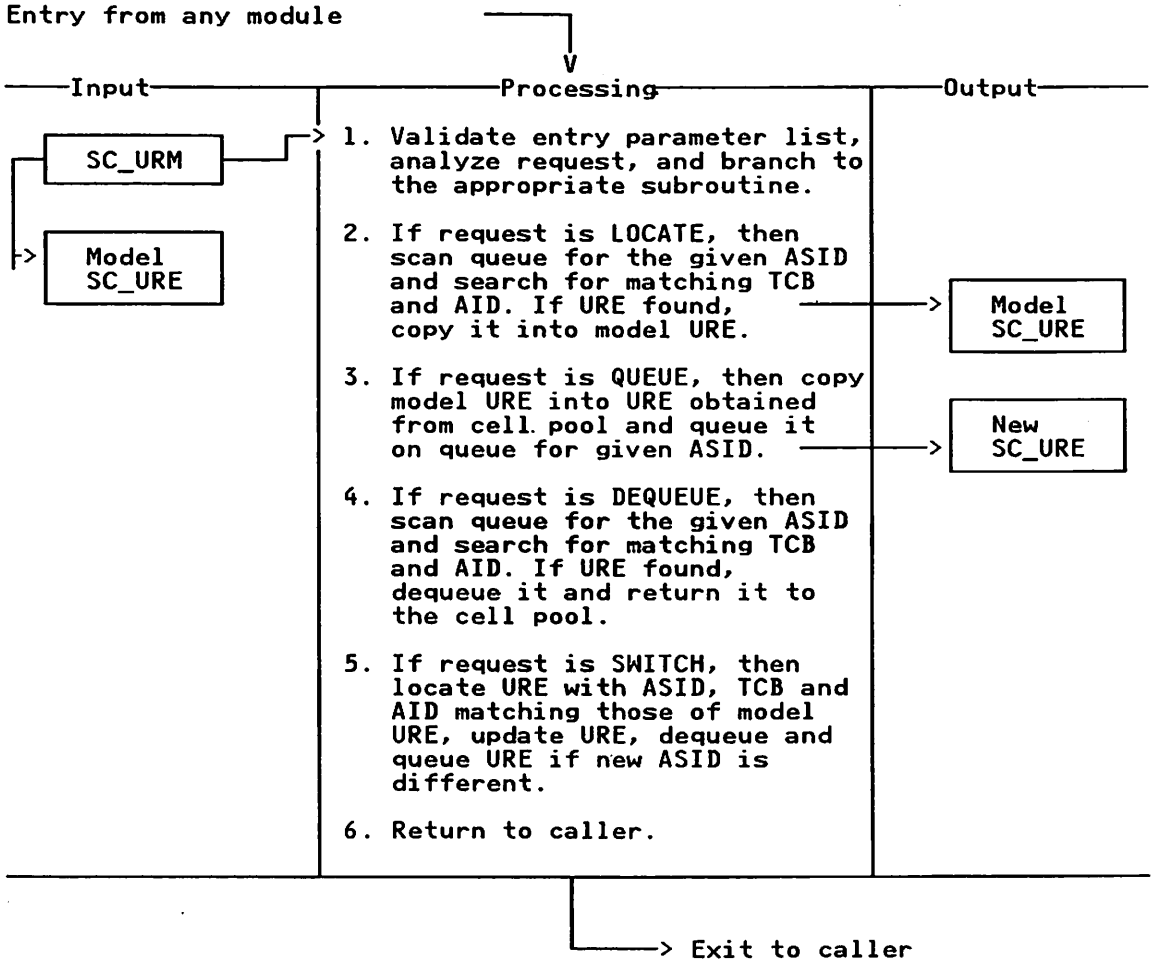
Data Area	*Description*
SC_SLASVT	Address Space Vector Table This table keeps track of the address spaces and the tasks within these address spaces that are making use of SUPERLINK/MVS resources. There is one entry per address space in ascending address space identifier (ASID) order, plus a special entry for ASID value 0. A null value in one of these pointers indicates that the associated address space is not making use of SUPERLINK/MVS resources. A non-null value is a pointer to a chain of control blocks called User Resource Elements (SC_UREs).
SC_URE	User Resource Element (URE) The User Resource Element is a control block that is used to keep track of resource utilization by task and ASID. There is one SC_URE per Task Control Block (TCB) per connection end point (AID).
SC_URM	User Resource Manager parameter list (SC_URM)

User Resource Manager Recovery

The User Resource Manager component does not perform its own recovery. All recovery actions are determined by the recovery environment enabled by the caller of the SUPERLINK/MVS User Resource Manager component.

This page has been intentionally left blank.

Diagram 11-1
S@CC0URM - User Resource Manager (part 1 of 2)



Extended Description

Explanation	*Module*	*Label*
1. The entry parameter list (SC_URM)) contains the requested function code and a pointer to a model SC_URE.	S@CC0URM	MAIN
2. For a LOCATE request, first the SC_URE is cleared out and then the queue of SC_URE's associated with the ASID specified in the SC_URM is searched for a matching TCB and AID. If a match is found, the located SC_URE is copied into the model SC_URE. Both the local and CMS locks are held while scanning the queue.	S@CC0URM	LOCATE
3. For a QUEUE request, a cell is obtained from the cell pool and the content of the model SC_URE is copied into it. The TCB, ASCB, ASID and AID fields are copied from the SC_URM parameter list. Then, the SC_URE obtained from the cell pool is queued in the queue associated with the ASID specified in the SC_URM with both the local and CMS locks held.	S@CC0URM	QUEUE
4. For a DEQUEUE request, the queue of SC_UREs associated with the ASID specified in the SC_URM is searched for a matching TCB and AID. If a match is found, the located SC_URE is dequeued and its storage is returned to the cell pool. Both the local and CMS locks are held while scanning the queue and dequeuing the SC_URE.	S@CC0URM	DEQUEUE
5. For a SWITCH request, the queue of SC_URE's associated with the ASID specified in the SC_URM is searched for a matching TCB and AID. If a match is found, the located SC_URE is updated with the AID, ASID, ASCB and TCB specified in the SC_URM. If the target ASID is different from the previous one, the SC_URE is dequeued and queued into the queue associated with the target ASID. Both the local and CMS locks are held while scanning the queue and/or updating, queuing, or dequeuing the SC_URE.	S@CC0URM	SWITCH

APPENDIX SECTION

Appendix A. Data Area Descriptions

AM_AMT

Common name: Association Manager Table

Macro ID: S@@AMT

DSECT name: AM_AMT

Created by: Association Manager (S@C9000)

Location: ECSA subpool 241 and key 8

Pointed to by: SC_SSVT_AMT field in the SSVT control block

Serialization: None

Function: Contains anchor points for Association Manager cell pools and global routines. It also contains the termination ECB.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	M_AMT_ID	TABLE ANACRONYM 'AMVT'
4	(4)	A-ADDRESS	4	M_AMT_S@C9300	--> AM INTERVAL TIMER CODE
8	(8)	A-ADDRESS	4	M_AMT_GWA	--> AM GLOBAL WORK AREA
12	(C)	FIXED	4	M_AMT_CPOOL	CELL POOL ID FOR REQUEST QUEUE'S
16	(10)	FIXED	4	M_AMT_CPOOL_4	CELL POOL ID FOR PRB'S AND PRC'S
20	(14)	FIXED	4	M_AMT_ECB	ECB TO BE POSTED IF TERMINATION
		....		RAIN_CODE	GRACEFUL TERMINATION
		1		LUSH_CODE	QUICK TERMINATION
		.... ..1.		BORT_CODE	ABORT
24	(18)	CHARACTER	4	M_AMT_ECB_ACRO	ECB ANACRONYM 'TERM'
		EQUATE	X'1C'	M_AMT_SIZE	LENGTH OF SC_AMT INC VAR PART
		EQUATE	241	M_AMT_SPOOL	SUBPOOL TO BE USED ON GETMAIN
		EQUATE	X'03'	M_AMT_CONTEXTS	MAX NUMBER OF CONTEXTS SUPPORTED
		EQUATE	X'258'	M_AMT_INTERVAL	TIMER INTERVAL WHILE DELETE-ANY

AM_APDE

Common name: AM Application Directory Entry

Macro ID: S@C@GWA

DSECT name: AM_APDE

Created by: Association Manager (S@C9000)

Location: ECSA subpool 241 and key 8

Pointed to by: Index value into Application Directory

Serialization: None

Function: Each APDE entry contains information for an application title being handled by the Association Manager

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	AM_APDE_ACRO	ACRONYM 'APDE'
4	(4)	FIXED	4	AM_APDE_QTPDU	QUALITY OF SERVICE TPDU SIZE
8	(8)	BITSTRING	1	AM_APDE_ATITLE_L	APPLICATION TITLE LENGTH
9	(9)	CHARACTER	32	AM_APDE_ATITLE	APPLICATION TITLE
41	(29)	CHARACTER	8	AM_APDE_TASK	MODULE NAME OF USERS TASK
49	(31)	CHARACTER	8	AM_APDE_CTXT	PRESENTATION CONTEXT NAME
57	(39)	CHARACTER	3		Reserved
60	(3C)	STRUCTURE	0		(ALIGN ON A WORD BOUNDRY)
60	(3C)	BITSTRING	1	AM_APDE_FLAGS	APPLICATION STATUS
		.1..		APDE_JOB	APPLICATION IS A JOB
		1...		APDE_APPL	APPLICATION IS A TASK
		1		APDE_OFFER	OFFER PENDING
		.... ..1.		APDE_FAILURE	UNABLE TO PUT OUT OFFER
61	(3D)	BITSTRING	1	AM_APDE_JCL	INDEX # CORRESPONDING JCL AREA
62	(3E)	BITSTRING	1	AM_APDE_QLVL	QUALITY OF SERVICE LEVEL
		1		APDE_QLOW	LOW
		.... ..1.		APDE_QMED	MEDIUM
		11		APDE_QHI	HIGH
63	(3F)	CHARACTER	1		Reserved
64	(40)	STRUCTURE	0		(ALIGN ON A WORD BOUNDRY)
		EQUATE	X'40'	AM_APDE_SIZE	LENGTH OF ENTRY

AM_APDH

Common name: AM Application Directory Header

Macro ID: S@C@GWA

DSECT name: AM_APDH

Created by: Association Manager (S@C9000)

Location: ECSA subpool 241 and key 8

Pointed to by: AM_GWA_APD field of the AM_GWA data area

Serialization: None

Function: Contains a count of the maximum and current number of Application Directory entries

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	AM_APDH_ACRO	ANACRONYM 'APDH'
4	(4)	FIXED	4	AM_APDH_MAX#	MAXIMUM NUMBER OF ENTRIES
8	(8)	FIXED	4	AM_APDH_CUR#	CURRENT NUMBER OF ENTRIES
		EQUATE	X'C'	AM_APDH_SIZE	LENGTH OF AM_APD ENTRY

AM_CDT

Common name: AM Controller Data Table

Macro ID: S@C@GWA

DSECT name: AM_CDT

Created by: Association Manager (S@C9000)

Location: ECSA Subpool 241 and key 8

Pointed to by: AM_GWA_CDT field in the AM_GWA data area

Serialization: None

Function: Contains general Association Manager Counts and status values used by the association Manager controller

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	M_CDT_ACRO	AN ACRONYM 'CDT '
4	(4)	CHARACTER	8	M_CDT_SL_JOBID	SLCN JOBID
12	(C)	BITSTRING	1	M_CDT_FLAG1	UNIVERSAL STATUS FLAGS
		1...		LAG1_DIAG_S	
		.1..		LAG1_DIAG_C	
		..1.		LAG1_DIAG_T	
		...1		LAG1_SVC_DUMP	
		.... 1...		LAG1_ST_ACTIVE	
		.... .1..		LAG1_ST_TERM_G	
		.... ..1.		LAG1_ST_TERM_Q	
		1		LAG1_ST_TERM_A	
13	(D)	BITSTRING	1	M_CDT_FLAG2	UNIVERSAL STATUS FLAGS
		...1		LAG2_TIMER_SET	
		1		LAG2_AIDS_STUCK	
14	(E)	BITSTRING	1	M_CDT_FLAG3	UNIVERSAL STATUS FLAGS
		...1		LAG3_ATTACH	
		1...		LAG4_PSAP_TEST	SET IF AMTYPE=MULTIPLE SPECIFIED
		.1..		LAG4_PASSCHK_NO	SET IF RACF CALL WITH PASSCHK=NO
15	(F)	BITSTRING	1	M_CDT_FLAG4	MORE ASSOC MGR STATUS FLAGS
16	(10)	FIXED	4	M_CDT_ATTACH	NUMBER OF PROCESSOR BEING
20	(14)	FIXED	4	M_CDT_FAILURE	THIS FIELD KEEPS A COUNT OF THE
24	(18)	FIXED	4	M_CDT_OFR_EXT	THIS KEEPS A COUNT OF PRB/PRC
28	(1C)	FIXED	4	M_CDT_ECB	ECB TO BE POSTED IF TIMEOUT
32	(20)	CHARACTER	4	M_CDT_ECB_ACRO	ECB ANACRONYM 'TIME'
		EQUATE	X'24'	M_CDT_SIZE	LENGTH OF SC_AMT INC VAR PART

AM_GWA

Common name: Association Manager Global Work Area

Macro ID: S@C@GWA

DSECT name: AM_GWA

Created by: Association Manager (S@C9000)

Location: ECSA Subpool 241 and key 8

Pointed to by: AM_AMT_GWA field in the AM_AMT data area

Serialization: None

Function: Contains pointers to all the other Association Manager tables

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	AM_GWA_ACRO	ANACRONYM 'GWA '
4	(4)	A-ADDRESS	4	AM_GWA_SSVT	--> SUPERLINK SSVT ADDRESS
8	(8)	A-ADDRESS	4	AM_GWA_CDT	--> CONTROLLER DATA TABLE
12	(C)	A-ADDRESS	4	AM_GWA_PCT	--> PROCESSOR CONTROL TABLE
16	(10)	A-ADDRESS	4	AM_GWA_RED	--> RESPONDER DIRECTORY
20	(14)	A-ADDRESS	4	AM_GWA_APD	--> APPLICATION PROGRAM DIR
24	(18)	A-ADDRESS	4	AM_GWA_S@C9UXAM	GENERAL USER EXIT INVOCKATION ROUTINE
28	(1C)	A-ADDRESS	4	AM_GWA_USERDATA	USER DATA FOR EXIT4 (SET BY INIT CALL
32	(20)	A-ADDRESS	4	AM_GWA_S@C9UX1	FOR S@C9100 - VARIABLE VALIDATION
36	(24)	A-ADDRESS	4	AM_GWA_S@C9UX2	FOR S@C9100 - JCL VALIDATION
40	(28)	A-ADDRESS	4	AM_GWA_S@C9UX3	FOR S@C9100 - JOB EXECUTION PROBLEMS
44	(2C)	A-ADDRESS	4	AM_GWA_S@C9UX4	SECURITY EXIT
48	(30)	A-ADDRESS	4	AM_GWA_S@C9UX5	(UNDEFINED)
52	(34)	A-ADDRESS	4	AM_GWA_S@C9UX6	(UNDEFINED)
		EQUATE	X'38'	AM_GWA_SIZE	LENGTH OF SC_GWA INDEX

AM_PCT

Common name: AM Processor Control Table header

Macro ID: S@C@GWA

DSECT name: AM_PCT

Created by: Association Manager (S@C9000)

Location: ECSA Subpool 241 and key 8

Pointed to by: AM_GWA_PCT field in the AM_GWA data area

Serialization: None

Function: Contains a count of the current and maximum number of entries in the processor control table.

Offsets		Type	Length	Name	Description
0	(0)	STRUCTURE	0	AM_PCTH	PCT - HEADER
0	(0)	CHARACTER	4	AM_PCTH_ACRO	ANACRONYM 'PCTH'
4	(4)	CHARACTER	4		Reserved
8	(8)	STRUCTURE	0		ALIGN ON DOUBLE WORD BOUNDRY
8	(8)	FIXED	4	AM_PCTH_MAX#	MAXIMUM NUMBER OF ENTRIES
12	(C)	FIXED	4	AM_PCTH_CUR#	CURRENT NUMBER OF ENTRIES IN USE
16	(10)	FIXED	4	AM_PCTH_JOBN	UNIQUE JOB NUMBER
20	(14)	FIXED	4	AM_PCTH_JOBI	INDICATOR OF CHAR FOR JOBNAME
		...1 1...		AM_PCTH_SIZE	LENGTH OF AM_PCT HEADER

AM_PCTE

Common name: Association Manager Processor Control Table Entry

Macro ID: S@C@GWA

DSECT name: AM_PCTE

Created by: Association Manager (S@C9000)

Location: ECSA Subpool 241 and key 8

Pointed to by: An Index offset into the PCT data area

AM_REDE_PCTEIX field in an associated REDE entry contains the index value

Serialization: None

Function: Contains a series of ECBs used for AM Processor control and information relating to the work being done by a processor.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	M_PCTE_ACRO	ACRONYM 'PCTE'
4	(4)	FIXED	4	M_PCTE_ECB1	ATTACH ECB FOR PROCESSOR
8	(8)	CHARACTER	4	M_PCTE_ECB1_ID	(ACRONYM 'ECB1')
12	(C)	FIXED	4	M_PCTE_ECB2	POSTED WHEN PROCESSOR GOES IDLE
16	(10)	CHARACTER	4	M_PCTE_ECB2_ID	(ACRONYM 'ECB2')
20	(14)	FIXED	4	M_PCTE_ECB3	POSTED WHEN A-ASSOCIATE RECEIVED
24	(18)	CHARACTER	4	M_PCTE_ECB3_ID	(ACRONYM 'ECB3')
28	(1C)	FIXED	4	M_PCTE_ECB4	POSTED WHEN CLONING IS REQUIRED
32	(20)	CHARACTER	4	M_PCTE_ECB4_ID	(ACRONYM 'ECB4')
36	(24)	FIXED	4	M_PCTE_ECB5	CONTROLLER WANTS AN A-OFFER DONE
40	(28)	CHARACTER	4	M_PCTE_ECB5_ID	(ACRONYM 'ECB5')
44	(2C)	FIXED	4	M_PCTE_ECB6	CONTROLLER WANTS CLOSE DOWN DONE
48	(30)	CHARACTER	4	M_PCTE_ECB6_ID	(ACRONYM 'ECB6')
52	(34)	FIXED	4	M_PCTE_ECB7	CONTROLLER WANTS CLONING DONE
56	(38)	CHARACTER	4	M_PCTE_ECB7_ID	(ACRONYM 'ECB7')
60	(3C)	FIXED	4	M_PCTE_ECB8	INTERFACE CODE : LISTEN DONE
64	(40)	CHARACTER	4	M_PCTE_ECB8_ID	(ACRONYM 'ECB8')
68	(44)	FIXED	4	M_PCTE_ECB9	INTERFACE CODE : DELETE-EP DONE
72	(48)	CHARACTER	4	M_PCTE_ECB9_ID	(ACRONYM 'ECB9')
76	(4C)	FIXED	4	M_PCTE_ECBA	INTERFACE CODE : DELETE-ANY DONE
80	(50)	CHARACTER	4	M_PCTE_ECBA_ID	(ACRONYM 'ECBA')

Offsets		Type	Length	Name	Description
84	(54)	FIXED	4	M_PCTE_ECBB	INTERVAL TIMER HAS POPPED
88	(58)	CHARACTER	4	M_PCTE_ECBB_ID	(ACRONYM 'ECBB')
92	(5C)	FIXED	4	M_PCTE_ECBC	OPERATOR RESPONDER TO A WTOR
96	(60)	CHARACTER	4	M_PCTE_ECBC_ID	(ACRONYM 'ECBC')
100	(64)	FIXED	4	M_PCTE_ECBD	ATTACHED TASK HAS TERMINATED
104	(68)	CHARACTER	4	M_PCTE_ECBD_ID	(ACRONYM 'ECBD')
108	(6C)	FIXED	4	M_PCTE_CASE	ECB FOR CASE
112	(70)	FIXED	4	M_PCTE_ENTRY	ENTRY NUMBER FOR THIS PCT ENTRY
116	(74)	A-ADDRESS	4	M_PCTE_APDE	POINTER TO ASSOC PCTE ENTRY
120	(78)	BITSTRING	1	M_PCTE_ATITLE_L	LTH OF APPL TITLE ON OFFER
121	(79)	CHARACTER	32	M_PCTE_ATITLE	APPLICATION TITLE ON OFFER
153	(99)	CHARACTER	3		Reserved
156	(9C)	A-ADDRESS	4	M_PCTE_TCB	TCB ADDRESS
160	(A0)	A-ADDRESS	4	M_PCTE_SWOP	ADDR OF RED ENTRY WITH STUCK Q
164	(A4)	STRUCTURE	0	M_PCTE_RQ	
164	(A4)	BITSTRING	1	M_PCTE_RQ_FLAGS	STATUS OF PROCESSOR
		....		CT_FREE	* PCT ENTRY FREE
		1		CT_INIT	* PCT ENTRY BEING INITIALISED
		.... ..1.		CT_WAIT	* WAIT FOR TASK TO TERMINATE
		.... .1..		CT_ACTIVE	* IDLE BUT HAS BEEN POSTED
		.... 1...		CT_TERM	* PCT ENTRY BEING TERMINATED
		...1		CT_IDLE	* PROCESSOR IDLE
		..1.		CT_OFFER_PD	* OFFER PENDING
		.1..		CT_LISTEN_PD	* LISTEN PENDING
		1...		CT_DELETE_PD	* DELETE PENDING
165	(A5)	BITSTRING	1	M_PCTE_ACTIONS	SECONDARY STATUS VALUES
		.... ..1.		CT_NO_CELL	* NO PRB/PRC CELL, UNABLE TO
		.... .1..		CT_POSTED	* IDLE PROCESSOR REQUESTED
166	(A6)	BITSTRING	2		Reserved
168	(A8)	FIXED	4	M_PCTE_REDEIX	ASSOCIATED AM_RED ENTRY INDEX
172	(AC)	FIXED	80	M_PCTE_MACWORK	WORK AREA FOR THE ATTACH MACRO
		EQUATE	X'FC'	M_PCTE_SIZE	LENGTH OF AM_PCT ENTRY

AM_REDE

Common name: Responder Directory Entry (Common Storage)

Macro ID: S@C@GWA

DSECT name: AM_REDE

Created by: Association Manager Processor (S@C9123B or S@C9128)

Location: ECSA subpool 241 and key 8

Pointed to by: An index value gives the offset to an address within the REDH data area that contains a pointer to it.

AM_PCTE_REDEIX field in an associated PCTE data area contains the index value.

Serialization: Serialization is performed by using ENQ and DEQ. The major name used is "ASSOCMGR" and the minor name used is the address of the REDE entry.

Function: Contains information that relates to a responder initiated by the Association Manager

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	AM_REDE_ACRO	ANACRONYM 'REDE'
4	(4)	FIXED	4	AM_REDE_INDEX	INDEX VALUE FOR THIS ENTRY
8	(8)	BITSTRING	1	AM_REDE_TYPE	RESPONDER TYPE
		.1..		REDE_JOB	:RESPONDER IS A STARTED JOB
		1...		REDE_TASK	:RESPONDER IS AN ATTACHED TASK
9	(9)	CHARACTER	4	AM_REDE_SUBSYS	SUBSYSTEM NAME
13	(D)	CHARACTER	8	AM_REDE_JOBNAME	MVS JOBNAME
21	(15)	CHARACTER	8	AM_REDE_JOBID	MVS JOB ID
13	(D)	CHARACTER	8	AM_REDE_TASK	MVS TASK NAME
21	(15)	CHARACTER	3		Reserved
24	(18)	A-ADDRESS	4	AM_REDE_TASK_TCB	MVS TASK TCB ADDRESS
28	(1C)	CHARACTER	1		Reserved
29	(1D)	CHARACTER	1	AM_REDE_ATITLE_L	APPLICATION TITLE LENGTH
30	(1E)	CHARACTER	32	AM_REDE_ATITLE	APPLICATION TITLE
62	(3E)	CHARACTER	2		Reserved
64	(40)	A-ADDRESS	4	AM_REDE_APDE	APD ENTRY
68	(44)	A-ADDRESS	4	AM_REDE_POINTER	POINTER TO DATA IN PRIVATE STORE
72	(48)	A-ADDRESS	4	AM_REDE_ASCB	ASCB ADDRESS FOR A-ENDPOINT-GIVE
76	(4C)	A-ADDRESS	4	AM_REDE_TCB	TCB ADDRESS FOR A-ENDPOINT-GIVE
80	(50)	A-ADDRESS	4	AM_REDE_UECB	ADDRESS OF USERS ECB TO BE
84	(54)	FIXED	4	AM_REDE_PCTEIX	ASSOCIATED AM_PCT ENTRY INDEX

Offsets	Type	Length	Name	Description
88 (58)	A-ADDRESS	4	AM_REDE_REDG	ADDRESS OF QUEUE MANAGEMENT CELL
	EQUATE	X'5C'	AM_REDE_SIZE	LENGTH OF AM_REDE ENTRY

AM_REDH

Common name: Responder Directory Header

Macro ID: S@C@GWA

DSECT name: AM_REDH

Created by: Association Manager (S@C9000)

Location: ECSA Subpool 241 and key 8

Pointed to by: AM_GWA_RED field of the AM_GWA data area

Serialization: Serialization is performed using ENQ and DEQ. The major name used is "ASSOCMGR" and the minor name used is the address of the responder directory header.

Function: Contains a count of the current and maximum number of REDE entries in the directory. It also contains an indexed list of pointers to the entries such that an index value of 1 refers to the first responders pointer, an index value of 2 refers to the second responders pointer and so on.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	AM_REDH_ACRO	ANACRONYM 'REDH'
4	(4)	CHARACTER	4		Reserved
8	(8)	STRUCTURE	0		ALIGN ON DOUBLE WORD BOUNDRY
8	(8)	FIXED	4	AM_REDH_MAX#	MAXIMUM NUMBER OF ENTRIES
12	(C)	FIXED	4	AM_REDH_CUR#	CURRENT NUMBER OF ENTRIES
16	(10)	FIXED	4	AM_REDH_LOCK	PCTE THAT LAST ENQ'ED UPON REDH
20	(14)	FIXED	4	AM_REDH_CPOOL	CELL POOL ID FOR RED ELEMENTS
24	(18)	A-ADDRESS	4	AM_REDH_EBASE	START OF LIST OF ENTRY ADDRESS'S

AM_REDQ

Common name: Responder Directory Entry (Queue Management Cell)

Macro ID: S@C@GWA

DSECT name: AM_REDQ

Created by: Association Manager Processor (S@C9123B or S@C9128)

Location: ECSA subpool 241 and key 8

Pointed to by: AM_REDE_REDG field in the REDE entry that owns the queue

Serialization: Compare Double and Swap (CDS) serialization techniques are used here

Function: Contains the status of the Associated Responder Directory Entry and the headers for the Request queue and Work queue which are used to contain requests for the associated responder

Offsets		Type	Length	Name	Description
0	(0)	BITSTRING	1	AM_REDQ_RQAFLAGS	RESPONDER STATUS FLAGS
		1...		NOTIFY_USER	: BIT 0 SET IF NOTIFY REQUIRED
		1		FLAG_LISTEN_PD	:
		.... ..1.		FLAG_LISTEN	:
		.... .1..		FLAG_STOP_DATA	: BITS 1 -- 7ARE SET TO
		.... 1...		FLAG_TERM_G	: INDICATE THE STATUS OF
		...1		FLAG_TERM_Q	: THE CONNECTION
		..1.		FLAG_ABORT	:
		.1..		FLAG_FAILURE	:
1	(1)	CHARACTER	1	AM_REDQ_FLAG2	MORE RESPONDER STATUS FLAGS
		1...		FLAG2_DELETE_ANY	:DELETE-ANY REQUEST PERFORMED
		.1..		FLAG2_CLONING	:RESPONDER BEING CLONED
2	(2)	FIXED	2	AM_REDQ_RQACNT	REQUEST QUEUE : ELEMENT COUNT
4	(4)	A-ADDRESS	4	AM_REDQ_RQATAIL	REQUEST QUEUE : BACKWARD LINK
8	(8)	A-ADDRESS	4	AM_REDQ_WQAHEAD	WORK QUEUE : FORWARD LINK
12	(C)	A-ADDRESS	4	AM_REDQ_WQATAIL	WORK QUEUE : BACKWARD LINK
16	(10)	BITSTRING	24		(UNDEFINED)
		EQUATE	X'28'	AM_REDQ_SIZE	LENGTH OF AM_REDQ ENTRY

AM_REDE_PRIVATE

Common name: Responder Directory Entry (Private Storage)

Macro ID: S@C@GWA

DSECT name: AM_REDE_PRIVATE

Created by: Association Manager Processor (S@C9123B and S@C9128)

Location: Private storage subpool 0 and key 8 (below 16M line)

Pointed to by: AM_REDE_POINTER

Serialization: None

Function: Contains identification and authentication information relating to the associated responder. This information is in private storage so that access to it is restricted to the Association Manager's address space.

Offsets	Type	Length	Name	Description
0 (0)	FIXED	2	AM_REDE_USERL	LENGTH OF USER IDIDENTIFIER
2 (2)	CHARACTER	8	AM_REDE_USER	USER IDENTIFIER (IDENT1)
10 (A)	FIXED	2	AM_REDE_USACL	LENGTH OF ACCOUNT INFORMATION
12 (C)	CHARACTER	70	AM_REDE_USAC	ACCOUNT INFORMATION (IDENT2)
82 (52)	FIXED	2	AM_REDE_GROUPL	LENGTH OF GROUP IDENTIFIER
84 (54)	CHARACTER	8	AM_REDE_GROUP	GROUP IDENTIFIER (IDENT3)
92 (5C)	FIXED	2	AM_REDE_PASSL	LENGTH OF USER PASSWORD
94 (5E)	CHARACTER	8	AM_REDE_PASS	USER PASSWORD (AUTH1)
102 (66)	BITSTRING	1	AM_REDE_AUTH2_L	LENGTH OF AUTH2
103 (67)	CHARACTER	16	AM_REDE_AUTH2	** UNDEFINED ** (AUTH2)
119 (77)	BITSTRING	1	AM_REDE_AUTH3_L	LENGTH OF AUTH3
120 (78)	CHARACTER	16	AM_REDE_AUTH3	** UNDEFINED ** (AUTH3)
136 (88)	BITSTRING	1	AM_REDE_PSAPL	LENGTH OF INITIATORS PSAP ID
137 (89)	CHARACTER	16	AM_REDE_PSAP	INITIATORS PSAP ID
153 (99)	CHARACTER	2	AM_REDE_MF	INITIATORS M/F ID
155 (9B)	BITSTRING	1	AM_REDE_TEXT_L	A-ASSOCIATE TEXT LENGTH
156 (9C)	CHARACTER	255	AM_REDE_TEXT	A-ASSOCIATE TEXT (SAVE AREA)
411 (19B)	BITSTRING	1	AM_REDE_R_TEXT_L	RESOLVED TEXT LENGTH
412 (19C)	CHARACTER	255	AM_REDE_R_TEXT	RESOLVED TEXT (SAVE AREA)

Offsets	Type	Length	Name	Description
	EQUATE	X'29B'	AM_REDE_PRIVATE_LTH	LENGTH OF PRIVATE AREA

LP_LOGE

Common name: Log Element

Macro ID: S@@LOGE

DSECT name: LP_LOGE

Created by: Log Processor initialization routine (S@C2210)

Location: Subpool 241, key 8

Pointed to by: SSVT

Serialization: None

Function: Holds information on a log message

Offsets		Type	Length	Name	Description
0	(0)	A-ADDRESS	4	LP_LOGEFWD	FORWARD CHAIN ('LOGE' - WORK Q)
		....		LP_LOGEID	ANACRONYM 'LOGE'
4	(4)	A-ADDRESS	4	LP_LOGEBWD	BACKWARD CHAIN
8	(8)	A-ADDRESS	4	LP_LOGEBLK	NEXT MESSAGE IN A BLOCK (OR 0)
12	(C)	STRUCTURE	0		ALIGN FOR WTO
		EQUATE	X'4A'	LP_LOGE_MSGLENM	MAXIMUM VALUE OF LP_LOGE_MSGLEN
12	(C)	BITSTRING	2	LP_LOGE_MSGLEN	LENGTH OF MESSAGE TEXT CONTAINED
14	(E)	CHARACTER	2		FOR WTO
16	(10)	CHARACTER	0	LP_LOGEMID	SUPERLINK MESSAGE IDENTIFIER
16	(10)	CHARACTER	2	LP_LOGEMID_GLBL	GLOBAL MESSAGE PREFIX S@/SL
18	(12)	CHARACTER	2	LP_LOGEMID_COMP	COMPONENT ID (EG UV=OPTIONS)
20	(14)	CHARACTER	4	LP_LOGEMID_NUM	MESSAGE NUMBER
24	(18)	CHARACTER	1	LP_LOGEMID_TYPE	CLASSIFICATION OF THE MESSAGE
		11.. 1..1		LP_LOGEMID_TYPEI	'I' - INFORMATION
		111. .11.		LP_LOGEMID_TYPEW	'W' - WARNING
		11.. .1.1		LP_LOGEMID_TYPEE	'E' - ERROR
		111. ..1.		LP_LOGEMID_TYPES	'S' - SEVERE
		11.. .1..		LP_LOGEMID_TYPED	'D' - DISASTER
25	(19)	CHARACTER	1	LP_LOGEMID_BLK	EYE CATCHER FOR BLOCKED MESSAGES
		.1.1 11..		LP_LOGEMID_BLKC	'*' - THIS LOGE IS A CONTINUATION OF PREVIOUS LOGE
		.1..		LP_LOGEMID_BLKS	' ' - NORMAL SETTING OF LP_LOGEMID_BLK
		EQUATE	60	LP_LOGE_MSGTEXTM	MAX MSG TEXT
26	(1A)	CHARACTER	60	LP_LOGE_MSGTEXT	MESSAGE TEXT
86	(56)	BITSTRING	4		SPACE FOR ROUTE AND DESCRIPTOR

Offsets	Type	Length	Name	Description
90 (5A)	BITSTRING	1	LP_LOGE_ROUTCDE	AN INDICATION OF TO WHERE THE
	1		LP_LOGE_SLLOG	OUTPUT TO SUPERLINK LOG
	.... ..1.		LP_LOGE_MVSLOG	OUTPUT TO MVS SYSTEM LOG
91 (5B)	CHARACTER	1		Reserved
92 (5C)	STRUCTURE	0		ENSURE CPOOL ALIGNS
	EQUATE	X'5C'	LP_LOGE_LEN	LENGTH OF LOG ELEMENT
	EQUATE	300	LP_LOGE_COUNT	COUNT OF LOG ELEMENTS IN THE POOL (TOTAL-SECONDARIES=0)
	EQUATE	241	LP_LOGE_SPOOL	SUBPOOL NUMBER FOR STORAGE

MH_ME

Common name: Message Entry

Macro ID: S@@M0MT

DSECT name: MH_ME

Loaded by: SLCN root module S@CC0000

Location: SLCN private storage

Pointed to by: MH_MIE_MSGADDR field in MH_MIE

Serialization: None

Function: Contains the text for a specific message. It also includes a count of variables and the offset and length of each

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	MH_ME_MSGNUM	MESSAGE NUMBER (IN EBCDIC)
4	(4)	CHARACTER	1	MH_ME_SEVERITY	MESSAGE SEVERITY INDICATOR
5	(5)	A-ADDRESS	1	MH_ME_NOMSGS	NUMBER OF LINES IN MESSAGE
6	(6)	A-ADDRESS	2	MH_ME_MSGLEN	LENGTH OF MESSAGE TEXT IN LINE
8	(8)	A-ADDRESS	1	MH_ME_NOVARS	NUMBER OF VARIABLES IN THIS LINE
9	(9)	A-ADDRESS	1	MH_ME_VAROFF	VARIABLE OFFSET WITHIN MSGTEXT
10	(A)	A-ADDRESS	1	MH_ME_VARLTH	LENGTH OF VARIABLE IN MSGTEXT
11	(B)	STRUCTURE	0	MH_ME_MSGTEXT	MESSAGE TEXT STARTS HERE

MH_MI

Common name: Message Index

Macro ID: S@@M0MT

DSECT name: MH_MI

Loaded by: SLCN root module S@CC0000

Location: SLCN private storage

Pointed to by: SC_SSVT_MSGTAB field of the SC_SSVT

Serialization: None

Function: Provides the number of entries in the message index and the size of each index entry.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	2	MH_MI_ID	CONTROL BLOCK ACRONYM - MI
2	(2)	CHARACTER	2		Reserved
4	(4)	A-ADDRESS	4	MH_MI_NOENTRY	NUMBER OF ENTRIES WITHIN INDEX
8	(8)	A-ADDRESS	4	MH_MI_ENTRYLEN	LENGTH OF EACH INDEX ENTRY
		EQUATE	X'C'	MH_MI_SIZE	LENGTH OF INDEX HEADER

MH_MIE

Common name: Message Index Entry

Macro ID: S@@M0MT

DSECT name: MH_MIE

Loaded by: SLCN root module S@CC0000

Location: SLCN private storage

Pointed to by: An offset within the message index

Serialization: None

Function: Provides a pointer to the message entry for a specific message

Offsets		Type	Length	Name	Description
0	(0)	A-ADDRESS	4	MH_MIE_MSGNUM	MESSAGE NUMBER (IN BINARY)
4	(4)	A-ADDRESS	4	MH_MIE_MSGADDR	ADDRESS OF THE MESSAGE ENTRY
		EQUATE	X'8'	MH_MIE_SIZE	LENGTH OF INDEX ENTRY

MI_MACB

Common name: Management Interface Association Control Block

Macro ID: S@CIMACB

DSECT name: MI_MACB

Created by: S@CI0000

Location: SLCN private storage

Pointed to by: MI_MICT

Serialization: None

Function: Contains all the information required by the connection manager subtask to manage a single management interface connection.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	MI_MACB_ID	'MACB' ACRONYM
4	(4)	CHARACTER	8	MI_MACB_NAME	MI CONNECTION NAME
12	(C)	A-ADDRESS	4	MI_MACB_SSVT	--> SC_SSVT
16	(10)	A-ADDRESS	4	MI_MACB_MICT	--> MI_MICT
20	(14)	A-ADDRESS	4	MI_MACB_IPARMS	--> INIT. PARMS OFF THE CIOT
24	(18)	BITSTRING	1	MI_MACB_TITLELN	LENGTH OF TITLE FIELD CONTENTS
25	(19)	CHARACTER	32	MI_MACB_TITLE	APPLICATION ENTITY TITLE FIELD
57	(39)	BITSTRING	1	MI_MACB_MFIDLN	CALLED MFID LENGTH FIELD
58	(3A)	CHARACTER	2	MI_MACB_MFID	CALLED MFID FIELD
60	(3C)	FIXED	2	MI_MACB_CONNID	CONNECTION IDENTIFIER
62	(3E)	FIXED	2	MI_MACB_PDUSIZE	MIPDUSIZE IN WORK AREA
64	(40)	A-ADDRESS	4	MI_MACB_ENDSCAN	-> END OF MIPDU SCAN ADDRESS
68	(44)	A-ADDRESS	4	MI_MACB_WORKAREA	-> WORK BUFFER FOR CASE DATA
		EQUATE	X'2000'	MI_MACB_WORKSIZE	SIZE OF THE WORK BUFFER
72	(48)	A-ADDRESS	4	MI_MACB_WORKMRQE	-> WORK BUFFER FOR MIPDU-DECODE
		....		MI_MACB_WORKPOOL	SUBPOOL FOR THE WORK BUFFER
76	(4C)	BITSTRING	1	MI_MACB_STATUS	STATUS OF THIS MI_MACB
		1...		MI_MACB_ACTIVE	MANAGER SUBTASK IS ACTIVE
		.1..		MI_MACB_TERM	MANAGER SUBTASK IS TERMINATING
		..1.		MI_MACB_LOGON	CONNECTION IS LOGGED ON
		...1		MI_MACB_LOGOFF	CONNECTION LOGOFF PENDING
		.... 1...		MI_MACB_REINS	SUBTASK IS TO BE RE-INSTATED

Offsets	Type	Length	Name	Description
	.... .1..		MI_MACB_IWAIT	SUBTASK IN INITIAL OPERATOR WAIT
	.... ..1.		MI_MACB_CASEDATA	CASE DATA PRESENT FOR PROCESSING
77 (4D)	BITSTRING	1	MI_MACB_STATUS2	STATUS OF THIS MI_MACB #2
	1...		MI_MACB_NOWAIT	NO CENTRAL WAIT REQUIRED
	.1..		MI_MACB_NORECV	NO CENTRAL RECEIVE REQUIRED
	..1.		MI_MACB_TRACE	TRACE ACTIVE FOR THIS MI_MACB
	...1		MI_MACB_XECBPOST	EXTERNAL ECB WAS POSTED
	.... 1...		MI_MACB_REQMRQE	RE-QUEUE CURRENT MI_MRQE
78 (4E)	BITSTRING	1	MI_MACB_REQUEST	REQUEST FOR THIS MI_MACB
	1...		MI_MACB_RTERM	SUBTASK IS TO TERMINATE
79 (4F)	BITSTRING	1	MI_MACB_RTYPE	CLARIFICATION OF TERM REQ. TYPE
	....		MI_MACB_TNORMAL	TERMINATE NORMAL REQUEST
	1		MI_MACB_TQUICK	TERMINATE QUICK REQUEST
	.... ..1.		MI_MACB_TABORT	TERMINATE ABORT REQUEST
80 (50)	BITSTRING	1	MI_MACB_INEVENT	CURRENT INPUT EVENT IN PROCESS
	....		MI_MACB_IMILGRQ	
	1		MI_MACB_IMIOFRQ	
	.... ..1.		MI_MACB_ILGRQ	
	11		MI_MACB_IMILGRSP	
	.... .1..		MI_MACB_IMILGRSN	
	.... .1.1		MI_MACB_ILGRSP	
	.... .11.		MI_MACB_ILGRSN	
	.... .111		MI_MACB_IMILFRQ	
	.... 1...		MI_MACB_ILFRQ	
	.... 1..1		MI_MACB_IMILFRSP	
	.... 1.1.		MI_MACB_IMILFRSN	
	.... 1.11		MI_MACB_ILFRSP	
	.... 11..		MI_MACB_ILFRSN	
	.... 11.1		MI_MACB_IMIABRQ	
	.... 111.		MI_MACB_IABRT	
	 1111		MI_MACB_IPBRT	
	...1		MI_MACB_IMICMRQ	
	...1 ...1		MI_MACB_ICMRQ	
	...1 ..1.		MI_MACB_IMIMGRQ	
	...1 ..11		MI_MACB_IMGRQ	
	...1 ..11		MI_MACB_MAXINEV	MAXIMUM INPUT EVENT NUMBER
81 (51)	BITSTRING	1	MI_MACB_OUTEVENT	CURRENT OUTPUT EVENT IN PROCESS
	....		MI_MACB_OLGRQ	
	1		MI_MACB_OAOFFRQ	
	.... ..1.		MI_MACB_OMILGCNP	

Offsets		Type	Length	Name	Description
		11		MI_MACB_OMILGCNN	
		.... .1..		MI_MACB_OLFRQ	
		.... .1.1		MI_MACB_OMILFIN	
		.... .11.		MI_MACB_OMILFCNP	
		.... .111		MI_MACB_OMILFCNN	
		.... 1...		MI_MACB_OABRT	
		.... 1..1		MI_MACB_OMIABIN	
		.... 1.1.		MI_MACB_OMIPAIN	
		.... 1.11		MI_MACB_OLFRSP	
		.... 11..		MI_MACB_OLFRSN	
		.... 11.1		MI_MACB_OMILGIN	
		.... 111.		MI_MACB_OLGRSP	
		 1111		MI_MACB_OLGRSN	
		...1		MI_MACB_OCMRQ	
		...1 ...1		MI_MACB_OMICMIN	
		...1 ..1.		MI_MACB_OMGRQ	
		...1 ..11		MI_MACB_OMIMGIN	
82	(52)	BITSTRING	1	MI_MACB_STATE	CURRENT STATE OF THIS MI_MACB
83	(53)	BITSTRING	1	MI_MACB_OLDSTATE	OLD STATE FOR TRACEBACK
		....		MI_MACB_STATE_0	STATE NUMBER STA0
		1		MI_MACB_STATE_1	STATE NUMBER STA1
		.... ..1.		MI_MACB_STATE_2	STATE NUMBER STA2
		11		MI_MACB_STATE_3	STATE NUMBER STA3
		.... .1..		MI_MACB_STATE_4	STATE NUMBER STA4
		.... .1.1		MI_MACB_STATE_5	STATE NUMBER STA5
		.... .11.		MI_MACB_STATE_6	STATE NUMBER STA6
		.... .11.		MI_MACB_MAXSTATE	MAXIMUM STATE NUMBER
84	(54)	BITSTRING	1	MI_MACB_ACT	ACTION TO BE PERFORMED
		1		MI_MACB_ACT_U1	ACTION NUMBER U1
		.... ..1.		MI_MACB_ACT_U2	ACTION NUMBER U2
		11		MI_MACB_ACT_U3	ACTION NUMBER U3
		.... .1..		MI_MACB_ACT_U4	ACTION NUMBER U4
		.... .1.1		MI_MACB_ACT_U5	ACTION NUMBER U5
		.... .11.		MI_MACB_ACT_U6	ACTION NUMBER U6
		.... .111		MI_MACB_ACT_U7	ACTION NUMBER U7
		.... 1...		MI_MACB_ACT_U8	ACTION NUMBER U8
		.... 1..1		MI_MACB_ACT_U9	ACTION NUMBER U9
		.... 1.1.		MI_MACB_ACT_U10	ACTION NUMBER U10
		.... 1.11		MI_MACB_ACT_U11	ACTION NUMBER U11
		.... 11..		MI_MACB_ACT_A1	ACTION NUMBER A1
		.... 11.1		MI_MACB_ACT_A2	ACTION NUMBER A2
		.... 111.		MI_MACB_ACT_A3	ACTION NUMBER A3
		 1111		MI_MACB_ACT_A4	ACTION NUMBER A4
		...1		MI_MACB_ACT_A5	ACTION NUMBER A5
		...1 ...1		MI_MACB_ACT_A6	ACTION NUMBER A6
		...1 ..1.		MI_MACB_ACT_A7	ACTION NUMBER A7
		...1 ..11		MI_MACB_ACT_A8	ACTION NUMBER A8
		...1 .1..		MI_MACB_ACT_A9	ACTION NUMBER A9
		...1 .1.1		MI_MACB_ACT_A10	ACTION NUMBER A10
		...1 .11.		MI_MACB_ACT_I1	**INTERNAL** ACTION NUMBER I1
		...1 .111		MI_MACB_ACT_I2	**INTERNAL** ACTION NUMBER I2
		...1 .111		MI_MACB_MAXACT	MAXIMUM ACTION NUMBER

Offsets	Type	Length	Name	Description
85 (55)	BITSTRING	1		RESERVED FOR LATER USE
86 (56)	FIXED	2	MI_MACB_#BADCASE	# CONSECUTIVE BAD A-RECEIVES
	.... .1.1		MI_MACB_MAXRECV#	MAXIMUM SUCH NUMBER ALLOWED
88 (58)	A-ADDRESS	4	MI_MACB_TCB	--> TCB OF MANAGER SUBTASK
92 (5C)	FIXED	4	MI_MACB_EXTECB	EXTERNAL ECB FOR THIS CONNECTION
96 (60)	FIXED	4	MI_MACB_PECB	ECB POSTED - SUBTASK ENDED
100 (64)	FIXED	4	MI_MACB_IECB	ECB POSTED - SUBTASK INITIALISED
104 (68)	STRUCTURE	8	MI_MACB_PARMLIST	IMBEDDED PARAMETER LIST
112 (70)	STRUCTURE	136	MI_MACB_ARE	IMBEDDED APPLICATION REQ ELT
248 (F8)	STRUCTURE	24	MI_MACB_PRB	IMBEDDED PRES. REQ BUFFERS
272 (110)	STRUCTURE	36	MI_MACB_PRBEMIF	IMBEDDED PRES. BUFF EL(M I/F)
308 (134)	STRUCTURE	36		IMBEDDED PRES. BUFF EL(CASE)
	.... ..1.		MI_MACB_#PRBE	# OF PRBE
	.11.		MI_MACB_PRB_LEN	LENGTH OF PRB/PRBE DEFN.
344 (158)	STRUCTURE	24	MI_MACB_PRBABRT	IMBEDDED ABORT PRB
368 (170)	STRUCTURE	36	MI_MACB_PRBEABRT	IMBEDDED ABORT PRBE EL(M I/F)
404 (194)	STRUCTURE	36		IMBEDDED ABORT PRBE EL(CASE)
	.... ..1.		MI_MACB_#PRBEAB	# OF PRBE ELTS
	.11.		MI_MACB_PRBA_LEN	LENGTH OF PRB/PRBE ABORT DEFN
440 (1B8)	STRUCTURE	20	MI_MACB_PRC	IMBEDDED PRES. CONTEXT DATA
460 (1CC)	STRUCTURE	12	MI_MACB_PRCEMIF	IMBEDDED PRES. CTXT EL(M I/F)
472 (1D8)	STRUCTURE	12		IMBEDDED PRES. CTXT EL(CASE)
	.... ..1.		MI_MACB_#PRCE	# OF PRCE
	..1. 11..		MI_MACB_PRC_LEN	LENGTH OF PRC/PRCE DEFN.
484 (1E4)	A-ADDRESS	4	MI_MACB_WORKQ_F	FIFO ORDERED REQUEST QUEUE
488 (1E8)	A-ADDRESS	4	MI_MACB_WORKQ_L	LIFO ORDERED REQUEST QUEUE
492 (1EC)	Y-ADDRESS	2	MI_MACB_SESSREQ	SESSION REQUEST PARAMETERS
494 (1EE)	BITSTRING	1	MI_MACB_PRESREQ	PRESENTATION REQUEST PARAMETERS
495 (1EF)	BITSTRING	1		RESERVED FOR LATER USE
496 (1F0)	CHARACTER	8	MI_MACB_CNTXNAME	CONTEXT NAME TO BE USED

Offsets	Type	Length	Name	Description
504 (1F8)	FIXED	4	MI_MACB_TIMER	TIMER INTERVAL FOR STIMERM
508 (1FC)	FIXED	4	MI_MACB_TIMERECB	ECB POSTED BY INTERVAL TIMER
512 (200)	FIXED	4	MI_MACB_QOS	QUALITY OF SERVICE PARAMETERS
516 (204)	FIXED	4		RESERVED FOR LATER USE
	EQUATE	X'208'	MI_MACB_SIZE	LENGTH OF MI_MACB
	EQUATE	0	MI_MACB_SPOOL	SUBPOOL TO USE

MI_MICT

Common name: Management Interface Control Table

Macro ID: S@CIMICT

DSECT name: MI_MICT

Created by: S@CI0000

Location: SLCN private storage

Pointed to by: SC_SSVT

Serialization: None

Function: This is the major control block of the management interface component. It describes the state of the component as a whole.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	MI_MICT_ID	'MICT' ACRONYM
4	(4)	A-ADDRESS	4	MI_MICT_SSVT	--> SC_SSVT
8	(8)	BITSTRING	1	MI_MICT_TITLELN	LENGTH OF TITLE FIELD CONTENTS
9	(9)	CHARACTER	32	MI_MICT_TITLE	APPLICATION ENTITY TITLE FIELD
41	(29)	BITSTRING	1	MI_MICT_STATUS	STATUS OF THIS MI_MICT
		1...		MI_MICT_TPEND	M I/F TERMINATION PENDING
		.1..		MI_MICT_STILLACT	M I/F SUBTASKS STILL ACTIVE
		..1.		MI_MICT_NODEFS	NO MICOND DEFINITINIONS EXIST
		...1		MI_MICT_NONMIC	NO MICON STATEMENT PRESENT
		.... 1...		MI_MICT_DETACHED	SUBTASK ABORTIVE DETACH DONE
42	(2A)	BITSTRING	2		RESERVED FOR LATER USE
44	(2C)	FIXED	4	MI_MICT_ELST	MANAGEMENT INTERFACE
		.... .1..		MI_MICT_ELEN	LENGTH OF A SINGLE ENTRY
48	(30)	FIXED	20		2ND AND SUBSEQUENT ENTRIES
		...1 1...		MI_MICT_LEN	LENGTH OF TOTAL LIST
		.... .11.		MI_MICT_ENUM	NUMBER OF ENTRIES
68	(44)	FIXED	12	MI_MICT_BXLESCAN	PARMS FOR BXLE SCAN OF MI_MACBS
80	(50)	FIXED	4	MI_MICT_TERMECB	M I/F TO TERMINATE ECB
		....		MI_MICT_TERM_N	-------- TERMINATE NORMAL
		1		MI_MICT_TERM_Q	-------- TERMINATE QUICK
		.... ..1.		MI_MICT_TERM_A	-------- TERMINATE ABORT
84	(54)	FIXED	4	MI_MICT_WORKECB	M I/F WORK-TO-DO-ECB
88	(58)	FIXED	4	MI_MICT_SUBTECB	CONN. MGR. SUBTASK TERMINATED

Offsets		Type	Length	Name	Description
92	(5C)	FIXED	4	MI_MICT_CONNECB	A CONNECTION HAS LOGGED ON ECB
96	(60)	A-ADDRESS	4	MI_MICT_MACBQ	-> MI_MACB TABLE
100	(64)	FIXED	4	MI_MICT_MACBN	# MI_MACBS IN THE TABLE
104	(68)	FIXED	4	MI_MICT_MACBL	LENGTH OF WHOLE MI_MACB TABLE
108	(6C)	A-ADDRESS	4	MI_MICT_MIFIQ_L	INPUT QUEUE (LIFO ORDER)
112	(70)	A-ADDRESS	4	MI_MICT_MIFIQ_F	INPUT QUEUE (FIFO ORDER)
116	(74)	A-ADDRESS	4	MI_MICT_MIFOQ_L	OUTPUT QUEUE (LIFO ORDER)
120	(78)	A-ADDRESS	4	MI_MICT_MIFOQ_F	OUTPUT QUEUE (FIFO ORDER)
124	(7C)	BITSTRING	4		RESERVED FOR LATER USE
128	(80)	STRUCTURE	0		ENSURE WE'RE MULTIPLE OF DWORD
		EQUATE	X'80'	MI_MICT_SIZE	LENGTH OF MI_MICT
		EQUATE	0	MI_MICT_SPOOL	SUBPOOL TO USE

MI_MRQE

Common name: Management Interface Request Element

Macro ID: S@CIMRQE

DSECT name: MI_MRQE

Created by: Components of SLCN wishing to communicate with the management interface component (such as the Product Operator component)

Location: SLCN private storage

Pointed to by: Request block chained from a queue anchored to MI_MICT or MI_MACB

Serialization: None

Function: Conveys requests from components of SLCN to the Management Interface component.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	MI_MRQE_ID	'MRQE' ACRONYM
4	(4)	A-ADDRESS	4	MI_MRQE_NEXT	--> NEXT MI_MRQE ON CHAIN
8	(8)	FIXED	4	MI_MRQE_SIZE	SIZE OF MI_MRQE
12	(C)	FIXED	4	MI_MRQE_SUBPOOL	SUBPOOL CONTAINING THE MI_MRQE
16	(10)	CHARACTER	8	MI_MRQE_CONAME	TARGET/SOURCE CONNECTION NAME
24	(18)	BITSTRING	1	MI_MRQE_REQUEST	REQUEST TO BE PERFORMED
		....		MI_MRQE_RLOGON	LOGON REQUEST
		1		MI_MRQE_RLOGOFF	LOGOFF REQUEST
		.... ..1.		MI_MRQE_RLOFFER	LOGON OFFER REQUEST
		11		MI_MRQE_RCOMMAND	COMMAND REQUEST
		.... .1..		MI_MRQE_RMESSAGE	MESSAGE REQUEST
		.... .1.1		MI_MRQE_ICOMMAND	COMMAND INDICATION
		.... .11.		MI_MRQE_IMESSAGE	MESSAGE INDICATION
		.... .111		MI_MRQE_RSTART	START REQUEST
		.... 1...		MI_MRQE_ILOGOFF	LOGOFF INDICATION
		.... 1..1		MI_MRQE_CLOGOFF	LOGOFF CONFIRMATION
		.... 1.1.		MI_MRQE_ILOGON	LOGON INDICATION
		.... 1.11		MI_MRQE_CLOGON	LOGON CONFIRMATION
		.... 11..		MI_MRQE_IABORT	ABORT INDICATION
		.... 11.1		MI_MRQE_IPBORT	P-ABORT INDICATION
		.... 111.		MI_MRQE_SLOGON	LOGON RESPONSE
		 1111		MI_MRQE_SLOGOFF	LOGOFF RESPONSE
		...1		MI_MRQE_RABORT	ABORT REQUEST
		...1		MI_MRQE_MAXREQ	MAXIMUM PRESENTLY ALLOCATED
25	(19)	BITSTRING	1	MI_MRQE_RESULT	RESULT OF REQUEST
		....		MI_MRQE_ACCEPTED	ACCEPTED
		1		MI_MRQE_REJECTED	REJECTED
26	(1A)	BITSTRING	1	MI_MRQE_REASON	REASON FOR REJECTION
		....		MI_MRQE_LGRSNORM	LOGON REQ NORMAL LOGON

Offsets		Type	Length	Name	Description
		1		MI_MRQE_LGRSUNSP	----"---- UNSPECIFIED REASON
		.... ..1.		MI_MRQE_LGRSPNSP	----"---- PROTOCOL NOT SUPPORTED
		....		MI_MRQE_LFRQNORM	LOGOFF REQ NORMAL LOGOFF
		1		MI_MRQE_LFRQURGT	----"----- URGENT LOGOFF
		.... ..1.		MI_MRQE_LFRQUNSP	----"----- UNSPECIFIED REASON
		....		MI_MRQE_LFRSNORM	LOGOFF RESP. NORMAL LOGOFF
		1		MI_MRQE_LFRSNCOM	----"------- NOT COMPLETED
		.... ..1.		MI_MRQE_LFRSUNSP	----"------- UNSPECIFIED REASON
27	(1B)	BITSTRING	1		RESERVED FOR LATER USE
28	(1C)	FIXED	4	MI_MRQE_DATALEN	LENGTH OF DATA FIELD
		EQUATE	X'20'	MI_MRQE_HLEN	LENGTH OF MI_MRQE FIXED HEADER
		EQUATE	0	MI_MRQE_SPOOL	SUBPOOL TO USE
32	(20)	STRUCTURE	0	MI_MRQE_DATA	START OF DATA FIELD

Extension to request element for logon request

32	(20)	BITSTRING	1	MI_MRQELOGON_PV	LOGON REQUEST PROTOCOL VERSION
		....		MI_MRQELOGON_PV0	LOGON REQUEST PROTOCOL VERSION-0
33	(21)	CHARACTER	1		Reserved
34	(22)	FIXED	2	MI_MRQELOGON_UDL	LOGON REQUEST USER DATA LENGTH
36	(24)	CHARACTER	0	MI_MRQELOGON_UD	LOGON REQUEST USER DATA
		EQUATE	X'20'	MI_MRQELOGON_XLN	LENGTH OF FIXED PART
		EQUATE	X'24'	MI_MRQELOGON_LEN	LOGON REQUEST FIXED HEAD LENGTH
		EQUATE	X'04'	MI_MRQELOGON_DLEN	LOGON RQEUEST DATA FIXED LEN

Extension to request element for logon offer

32	(20)	STRUCTURE	36		SAME AS FOR LOGON REQ
		EQUATE	X'20'	MI_MRQELOFFR_XLN	LENGTH OF FIXED PART
		EQUATE	X'44'	MI_MRQELOFFR_LEN	LOGON OFFER FIXED HEAD LENGTH
		EQUATE	X'04'	MI_MRQELOFFR_DLEN	LOGON OFFER DATA FIXED LEN

Offsets		Type	Length	Name	Description
Extension to request element for logoff request					
32	(20)	BITSTRING	1	MI_MRQELOGOF_TYP	LOGOFF REQUEST TYPE
		....		MI_MRQELOGOF_TNO	-- NORMAL LOGOFF REQUEST
		1		MI_MRQELOGOF_TUR	-- URGENT LOGOFF REQUEST
		.... ..1.		MI_MRQELOGOF_TUS	-- UNSPECIFIED LOGOFF REQUEST
33	(21)	CHARACTER	1		Reserved
34	(22)	FIXED	2	MI_MRQELOGOF_UDL	LOGOFF REQUEST USER DATA LENGTH
36	(24)	CHARACTER	0	MI_MRQELOGOF_UD	LOGOFF REQUEST USER DATA
		EQUATE	X'20'	MI_MRQELOGOF_XLN	LENGTH OF FIXED PART
		EQUATE	X'44'	MI_MRQELOGOF_LEN	LOGOFF REQUEST FIXED HEAD LENGTH
		EQUATE	X'04'	MI_MRQELOGOF_DLEN	LOGOFF REQUEST DATA FIXED LEN
Extension to request element for command request					
32	(20)	BITSTRING	1	MI_MRQECMD_TYP	COMMAND REQUEST TYPE
		....		MI_MRQECMD_TFO	-- FORMATTED CMD REQUEST
		1		MI_MRQECMD_TUF	-- UNFORMATTED CMD REQUEST
33	(21)	BITSTRING	1	MI_MRQECMD_DST	COMMAND DESTINATION
		....		MI_MRQECMD_DOP	-- LOCAL OPERATING SYSTEM
		1		MI_MRQECMD_DSL	-- SUPERLINK
34	(22)	BITSTRING	1	MI_MRQECMD_SRF	SOURCE TYPE FLAG FIELD
		....		MI_MRQECMD_SRFNP	SOURCE FIELD NOT PRESENT
		1		MI_MRQECMD_SRFTU	SOURCE USER FIELD PRESENT
		.... ..1.		MI_MRQECMD_SRFCN	MCS CONSOLE ID FIELD PRESENT
		11		MI_MRQECMD_SRFRF	REMOTE SYSTEM FORMAT PRESENT
35	(23)	CHARACTER	1		Reserved
36	(24)	FIXED	0	MI_MRQECMDSOURCE	START OF SOURCE FIELD
36	(24)	BITSTRING	1	MI_MRQECMD_SRCC	CONSOLE ID IF OPERATOR ISSUED
36	(24)	CHARACTER	8	MI_MRQECMD_SRC	COMMAND SOURCE USER ID
36	(24)	BITSTRING	1	MI_MRQECMD_SRRFL	REMOTE FORMAT LENGTH FIELD
37	(25)	BITSTRING	20	MI_MRQECMD_SRRFD	REMOTE FORMAT SOURCE FIELD

Offsets		Type	Length	Name	Description
		EQUATE	X'17'	MI_MRQECMDSRCLEN	LENGTH OF WHOLE SRC FIELD
57	(39)	CHARACTER	1		Reserved
68	(44)	FIXED	2	MI_MRQECMD_UDL	CMD REQUEST USER DATA LENGTH
70	(46)	CHARACTER	0	MI_MRQECMD_UD	CMD REQUEST USER DATA
		EQUATE	X'20'	MI_MRQECMD_XLN	LENGTH OF FIXED PART
		EQUATE	X'46'	MI_MRQECMD_LEN	CMD REQUEST FIXED HEAD LENGTH
		EQUATE	X'1C'	MI_MRQECMD_DLEN	CMD REQUEST DATA FIXED LEN

Extension to request element for command request

32	(20)	BITSTRING	1	MI_MRQEMSG_TYP	MESSAGE REQUEST TYPE
		....		MI_MRQEMSG_TFO	-- FORMATTED MSG REQUEST
		1		MI_MRQEMSG_TUF	-- UNFORMATTED MSG REQUEST
33	(21)	BITSTRING	1	MI_MRQEMSG_DST	MESSAGE DESTINATION
		....		MI_MRQEMSG_DSL	-- SUPERLINK LOG
		1		MI_MRQEMSG_DMC	-- MASTER CONSOLE
		.... ..1.		MI_MRQEMSG_DML	-- MASTER CONSOLE AND LOG
		11		MI_MRQEMSG_DOR	-- POINT OF ORIGIN
34	(22)	BITSTRING	1	MI_MRQEMSG_SRF	SOURCE TYPE FLAG FIELD
		....		MI_MRQEMSG_SRFNP	SOURCE FIELD NOT PRESENT
		1		MI_MRQEMSG_SRFTU	SOURCE USER FIELD PRESENT
		.... ..1.		MI_MRQEMSG_SRFCN	MCS CONSOLE ID FIELD PRESENT
		11		MI_MRQEMSG_SRFRF	REMOTE SYSTEM FORMAT PRESENT
35	(23)	CHARACTER	1		Reserved
36	(24)	FIXED	0	MI_MRQEMSGSOURCE	START OF SOURCE FIELD
36	(24)	BITSTRING	1	MI_MRQEMSG_SRCC	CONSOLE ID IF OPERATOR ISSUED
36	(24)	CHARACTER	8	MI_MRQEMSG_SRC	COMMAND SOURCE USER ID
36	(24)	BITSTRING	1	MI_MRQEMSG_SRRFL	REMOTE FORMAT LENGTH FIELD
37	(25)	BITSTRING	20	MI_MRQEMSG_SRRFD	REMOTE FORMAT SOURCE FIELD
		EQUATE	X'1A'	MI_MRQEMSGSRCLEN	LENGTH OF WHOLE SRC FIELD
57	(39)	CHARACTER	3		Reserved
70	(46)	FIXED	2	MI_MRQEMSG_UDL	MSG REQUEST USER DATA LENGTH
72	(48)	CHARACTER	0	MI_MRQEMSG_UD	MSG REQUEST USER DATA
		EQUATE	X'20'	MI_MRQEMSG_XLN	LENGTH OF FIXED PART

Offsets	Type	Length	Name	Description
	EQUATE	X'48'	MI_MRQEMSG_LEN	MSG REQUEST FIXED HEAD LENGTH
	EQUATE	X'1E'	MI_MRQEMSG_DLEN	MSG REQUEST DATA FIXED LEN

Extension to request element for abort request

32	(20)	FIXED	2	MI_MRQEABORT_UDL	ABORT REQUEST USER DATA LENG
34	(22)	CHARACTER	0	MI_MRQEABORT_UD	ABORT REQUEST USER DATA
		EQUATE	X'48'	MI_MRQEABORT_LEN	ABORT REQUEST FIXED HEAD LENGTH
		EQUATE	X'02'	MI_MRQEABORT_DLEN	ABORT RQEUEST DATA FIXED LEN

SC_CIOT

Common name: SUPERLINK Control Initialization Options Table

Macro ID: S@CICIOT

DSECT name: SC_CIOT

Created by: S@CC0000

Location: Common service area

Pointed to by: SC_SSVT

Serialization: None

Function: Contains a digest of the contents of the SUPERLINK subsystem's parameter library member.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	SC_CIOTID	CONTROL BLOCK ACRONYM 'CIOT'
		EQUATE	241	SC_CIOT_SPOOL	SUBPOOL FOR IOTS, INDIRECT AREAS
4	(4)	CHARACTER	1	SC_CIOTCHAR	SUBSYSTEM COMMAND CHARACTER
		.11. ...1		SC_CIOTCHAR_DFLT	DEFAULT FOR SC_CIOTCHAR
5	(5)	CHARACTER	4	SC_CIOTSSM	SUBSYSTEM SUPPORT MODULE PREFIX
9	(9)	CHARACTER	3		Reserved
12	(C)	A-ADDRESS	4	SC_CIOTNIOT	--> SLNET OPTIONS TABLE (NIOT)
16	(10)	A-ADDRESS	4	SC_CIOTJIOT	--> SLJP OPTIONS TABLE (JIOT)
20	(14)	A-ADDRESS	4	SC_CIOTVIOT	--> SLVT OPTIONS TABLE (VIOT)
24	(18)	A-ADDRESS	4	SC_CIOTOIOT	--> SLOP OPTIONS TABLE (OIOT)
28	(1C)	A-ADDRESS	4	SC_CIOTPFSS	--> AREA CONTAINING FSS DETAILS
32	(20)	A-ADDRESS	4	SC_CIOTPMIC	--> AREA CONTAINING MIC DETAILS
36	(24)	A-ADDRESS	4	SC_CIOTPATI	--> AREA CONTAINING APPLICATION
		EQUATE	10	SC_CIOT#JCL	NUMBER OF JCL AREAS
40	(28)	STRUCTURE	0	SC_CIOTPJCL	START -->S TO AREAS CONTAINING
40	(28)	FIXED	4	SC_CIOTPJCL_#01	--> 1ST AREA CONTAINING JCL
44	(2C)	FIXED	4	SC_CIOTPJCL_#02	--> 2ND AREA CONTAINING JCL
48	(30)	FIXED	4	SC_CIOTPJCL_#03	--> 3RD AREA CONTAINING JCL
52	(34)	FIXED	4	SC_CIOTPJCL_#04	--> 4TH AREA CONTAINING JCL

Offsets		Type	Length	Name	Description
56	(38)	FIXED	4	SC_CIOTPJCL_#05	--> 5TH AREA CONTAINING JCL
60	(3C)	FIXED	4	SC_CIOTPJCL_#06	--> 6TH AREA CONTAINING JCL
64	(40)	FIXED	4	SC_CIOTPJCL_#07	--> 7TH AREA CONTAINING JCL
68	(44)	FIXED	4	SC_CIOTPJCL_#08	--> 8TH AREA CONTAINING JCL
72	(48)	FIXED	4	SC_CIOTPJCL_#09	--> 9TH AREA CONTAINING JCL
76	(4C)	FIXED	4	SC_CIOTPJCL_#10	--> 10TH AREA CONTAINING JCL
80	(50)	BITSTRING	1	SC_CIOTNOFS	# SUPERLINK FSS ADDRESS SPACES
81	(51)	BITSTRING	1	SC_CIOTNMIC	# SUPERLINK MI CONNECT DETAILS
82	(52)	BITSTRING	1	SC_CIOTNATI	# SUPERLINK APPLICATION TITLE
83	(53)	STRUCTURE	0	SC_CIOTNJCL	START JCL STATEMENT COUNTS
83	(53)	BITSTRING	1	SC_CIOTNJCL_#01	# JCL STATEMENTS IN AREA 01
84	(54)	BITSTRING	1	SC_CIOTNJCL_#02	# JCL STATEMENTS IN AREA 02
85	(55)	BITSTRING	1	SC_CIOTNJCL_#03	# JCL STATEMENTS IN AREA 03
86	(56)	BITSTRING	1	SC_CIOTNJCL_#04	# JCL STATEMENTS IN AREA 04
87	(57)	BITSTRING	1	SC_CIOTNJCL_#05	# JCL STATEMENTS IN AREA 05
88	(58)	BITSTRING	1	SC_CIOTNJCL_#06	# JCL STATEMENTS IN AREA 06
89	(59)	BITSTRING	1	SC_CIOTNJCL_#07	# JCL STATEMENTS IN AREA 07
90	(5A)	BITSTRING	1	SC_CIOTNJCL_#08	# JCL STATEMENTS IN AREA 08
91	(5B)	BITSTRING	1	SC_CIOTNJCL_#09	# JCL STATEMENTS IN AREA 09
92	(5C)	BITSTRING	1	SC_CIOTNJCL_#10	# JCL STATEMENTS IN AREA 10
93	(5D)	BITSTRING	1	SC_CIOTSLRM	FLAG BYTE - REMOTE SERVICES
		1...		SC_CIOTSLRM_NR	REMOTE SERVICES NOT REQUIRED
94	(5E)	CHARACTER	8	SC_CIOT_SRB	FSS SRB
102	(66)	CHARACTER	8	SC_CIOT_FRR	FSS SRB FRR
110	(6E)	CHARACTER	8	SC_CIOT_LTASK	FSS LISTENER TASK
118	(76)	CHARACTER	8	SC_CIOTNMN	SLNET OPTIONS MEMBER NAME
126	(7E)	CHARACTER	8	SC_CIOTJMN	SLJP OPTIONS MEMBER NAME

Offsets	Type	Length	Name	Description
134 (86)	CHARACTER	8	SC_CIOTVMN	SLVT OPTIONS MEMBER NAME
142 (8E)	CHARACTER	8	SC_CIOTOMN	SLOP OPTIONS MEMBER NAME
150 (96)	FIXED	2	SC_CIOT#S	COUNT OF SITE MVS START CMDS
152 (98)	A-ADDRESS	4	SC_CIOTPS	--> AREA MVS START CMD DETAILS
156 (9C)	CHARACTER	8	SC_CIOTNAM	SUPERLINK SUBSYSTEM NAME
164 (A4)	CHARACTER	50	SC_CIOT_SITE	SITE NAME
214 (D6)	BITSTRING	0	SC_CIOT_LOGRCDE	LOG ROUTE CODES
214 (D6)	BITSTRING	1	SC_CIOT_LOGRCDE1	1ST BYTE LOG ROUTE CODES
215 (D7)	BITSTRING	1	SC_CIOT_LOGRCDE2	2ND BYTE LOG ROUTE CODES
216 (D8)	CHARACTER	8	SC_CIOT_CNMSGNME	SUB-SYSTEM MESSAGE TABLE NAME
224 (E0)	CHARACTER	8	SC_CIOT_DAMSGNME	DATA ACCESS MESSAGE TABLE NAME
232 (E8)	BITSTRING	1	SC_CIOT_#CDTKS	MAX NUMBER OF DATA ACCESS TASKS
	.... 1.1.		SC_CIOT_#CDTKSD	DEFAULT FOR SC_CIOT_#CDTKSD
233 (E9)	BITSTRING	0	SC_CIOT_ROUTCDE	WTO/WTOR ROUTE CODES
233 (E9)	BITSTRING	1	SC_CIOT_ROUTCDE1	1ST BYTE WTO/WTOR ROUTE CODES
234 (EA)	BITSTRING	1	SC_CIOT_ROUTCDE2	2ND BYTE WTO/WTOR ROUTE CODES
	EQUATE	X'8000'	SC_CIOT_ROUTCDED	DEFAULT ROUTCDES
235 (EB)	BITSTRING	1	SC_CIOT_SVC#	SUPERLINK SVC NUMBER
	111. .11.		SC_CIOT_SVC#D	DEFAULT SUPERLINK SVC NUMBER
236 (EC)	CHARACTER	1	SC_CIOT_MICATL	LENGTH OF SC_CIOT_MICAT
237 (ED)	CHARACTER	32	SC_CIOT_MICAT	APPLICATION ENTITY TITLE (LOCAL)
269 (10D)	CHARACTER	1		Reserved
270 (10E)	STRUCTURE	0	SC_CIOT_PROCVECS	Association Manager vectors
270 (10E)	FIXED	2	SC_CIOT_PROCINIT	MAX # INITIATORS
	.... 1.1.		SC_CIOT_PROCINID	DEFAULT MAX # INITIATORS
272 (110)	FIXED	2	SC_CIOT_PROCRESP	MAX # RESPONDERS
	.... 1.1.		SC_CIOT_PROCRESD	DEFAULT MAX # RESPONDERS
274 (112)	FIXED	2	SC_CIOT_PROCCELL	MAX # CELLS
276 (114)	BITSTRING	1	SC_CIOT_PROCTRCE	ASSOC MGR TRACING AND FLAGS
	1		SC_CIOT_TRACE1	BASIC AM-TRACE TO WTO
	.... ..1.		SC_CIOT_TRACE2	BASIC AM-TRACE TO WTO + LOG

Offsets	Type	Length	Name	Description
	.... .1..		SC_CIOT_TRACE3	FULL AM-TRACE TO WTO
	.... 1...		SC_CIOT_TRACE4	FULL AM-TRACE TO WTO + LOG
	1...		SC_CIOT_MULTIPLE	SKIP PSAP ID VERIFICATION
	....		SC_CIOT_SINGLE	DONT SKIP PSAP ID VERIFICATION
277 (115)	CHARACTER	0	SC_CIOT_CJUEXITS	COMPANION JOB USER EXITS
277 (115)	CHARACTER	8	SC_CIOT_DYNEXIT	DYNAMIC ALLOCATION EXIT
285 (11D)	CHARACTER	8	SC_CIOT_FTAMEXIT	PRE/POST FTAM EXIT
293 (125)	CHARACTER	3		Reserved
296 (128)	FIXED	4	SC_CIOT_OUTLIM	SYSOUT OUTPUT LIMIT
	EQUATE	X'7A120'	SC_CIOT_OUTLIMD	DEFAULT SYSOUT OUTPUT LIMIT
300 (12C)	FIXED	2	SC_CIOT_PRBLK	PRINT BLKSIZE
	EQUATE	X'1000'	SC_CIOT_PRBLKD	DEFAULT PRINT BLKSIZE
302 (12E)	FIXED	2	SC_CIOT_PRLRECL	PRINT LRECL
	1... 1..1		SC_CIOT_PRLRECLD	DEFAULT PRINT LRECL
304 (130)	BITSTRING	1	SC_CIOT_PRRECFM	PRINT RECFM
	.1.1 .1..		SC_CIOT_PRRECFMD	DEFAULT PRINT RECFM (VBA)
305 (131)	CHARACTER	1	SC_CIOT_PRCLASS	PRINT CLASS
	11.. ...1		SC_CIOT_PRCLASSD	PRINT CLASS
306 (132)	FIXED	2	SC_CIOT_PUBLK	PUNCH BLKSIZE
308 (134)	FIXED	2	SC_CIOT_PULRECL	PUNCH LRECL
310 (136)	BITSTRING	1	SC_CIOT_PURECFM	PUNCH RECFM
311 (137)	CHARACTER	1	SC_CIOT_PUCLASS	PUNCH CLASS
312 (138)	FIXED	2	SC_CIOT_PLBLK	PLOT BLKSIZE
314 (13A)	FIXED	2	SC_CIOT_PLLRECL	PLOT LRECL
316 (13C)	BITSTRING	1	SC_CIOT_PLRECFM	PLOT RECFM
317 (13D)	CHARACTER	1	SC_CIOT_PLCLASS	PLOT CLASS
318 (13E)	CHARACTER	0	SC_CIOT_AMUEXITS	ASSOCIATION MANAGER USER EXITS
318 (13E)	CHARACTER	8	SC_CIOT_AMUX1	VARIABLE VALIDATION 1
326 (146)	CHARACTER	8	SC_CIOT_AMUX2	JCL VALIDATION 2
334 (14E)	CHARACTER	8	SC_CIOT_AMUX3	JOB PROBLEM RESOLUTION 3
342 (156)	CHARACTER	8	SC_CIOT_AMUX4	SECURITY 4
350 (15E)	CHARACTER	8	SC_CIOT_AMUX5	(UNDEFINED) 5
358 (166)	CHARACTER	8	SC_CIOT_AMUX6	(UNDEFINED) 6
366 (16E)	BITSTRING	1	SC_CIOT_SAFPARML	
367 (16F)	BITSTRING	60	SC_CIOT_SAFPARM	INSTALLATION DATA FOR S.A.F.

Offsets	Type	Length	Name	Description
427 (1AB)	CHARACTER	0	SC_CIOT_DELIM	JCL PARM DELIMITERS
427 (1AB)	CHARACTER 11..	1	SC_CIOT_DELIMS SC_CIOT_DELIMSD	START START DEFAULT
428 (1AC)	CHARACTER 11.1	1	SC_CIOT_DELIME SC_CIOT_DELIMED	END END DEFAULT
429 (1AD)	CHARACTER	1		Reserved
430 (1AE)	STRUCTURE	0	SC_CIOT_JOBSVECS	Association Manager job scheduling vectors
430 (1AE)	FIXED	2	SC_CIOT_AMTTIME	ASSOC MGR TOTAL TIME (SECS)
432 (1B0)	FIXED	2	SC_CIOT_AMITIME	ASSOC MGR INTERVAL TIME (SECS)
434 (1B2)	BITSTRING 1... .... .1..	1	SC_CIOT_JACTION SC_CIOT_JACT_O SC_CIOT_JACT_C	ACTION TO BE TAKEN PROMPT THE OPERATOR CANCEL THE JOB
435 (1B3)	BITSTRING	1	SC_CIOT_ACCOUNTL	
436 (1B4)	CHARACTER	70	SC_CIOT_ACCOUNT	ACCOUNT
506 (1FA)	BITSTRING	1	SC_CIOT_MSGLEVLL	
507 (1FB)	CHARACTER	5	SC_CIOT_MSGLEVL	MSGLEVEL
512 (200)	BITSTRING	1	SC_CIOT_PASSWRDL	
513 (201)	CHARACTER	8	SC_CIOT_PASSWRD	PASSWORD
521 (209)	BITSTRING	1	SC_CIOT_PROGNMEL	
522 (20A)	CHARACTER	20	SC_CIOT_PROGNME	PROGRAMMER NAME
542 (21E)	BITSTRING	1	SC_CIOT_NOTIFYL	
543 (21F)	CHARACTER	7	SC_CIOT_NOTIFY	NOTIFY
550 (226)	BITSTRING	1	SC_CIOT_PROCL	
551 (227)	CHARACTER	8	SC_CIOT_PROC	PROC
559 (22F)	BITSTRING	1	SC_CIOT_CLASSL	
560 (230)	CHARACTER	8	SC_CIOT_CLASS	CLASS
568 (238)	BITSTRING	1	SC_CIOT_USERL	
569 (239)	CHARACTER	7	SC_CIOT_USER	USER
576 (240)	BITSTRING	1	SC_CIOT_GROUPL	
577 (241)	CHARACTER	8	SC_CIOT_GROUP	GROUP
585 (249)	BITSTRING	1	SC_CIOT_REGIONL	
586 (24A)	CHARACTER	8	SC_CIOT_REGION	REGION
594 (252)	BITSTRING	1	SC_CIOT_MSGCLSSL	
595 (253)	CHARACTER	1	SC_CIOT_MSGCLSS	MSGCLASS
596 (254)	BITSTRING	1	SC_CIOT_DATA1L	
597 (255)	CHARACTER	70	SC_CIOT_DATA1	SITE DATA 1
667 (29B)	BITSTRING	1	SC_CIOT_DATA2L	
668 (29C)	CHARACTER	70	SC_CIOT_DATA2	SITE DATA 2
738 (2E2)	BITSTRING	1	SC_CIOT_DATA3L	
739 (2E3)	CHARACTER	70	SC_CIOT_DATA3	SITE DATA 3
809 (329)	BITSTRING	1	SC_CIOT_TIMEL	
810 (32A)	CHARACTER	7	SC_CIOT_TIME	TIME

Offsets	Type	Length	Name	Description
817 (331)	BITSTRING	1	SC_CIOT_JOBNAMEL	
818 (332)	CHARACTER	8	SC_CIOT_JOBNAME	JOBNAME
	EQUATE	X'33A'	SC_CIOT_LEN	LENGTH OF DSECT

SC_CIOTATI

Common name: Application Titles Details

Macro ID: S@C1CIOT

DSECT name: SC_CIOTATI

Created by: Options Processor (S@C1000)

Location: CSA

Pointed to by: SC_CIOTPATI

Serialization: None

Function: Association Manager offer processing.

Offsets		Type	Length	Name	Description
0	(0)	FIXED	4	SC_CIOTATI_QTPDU	QUALITY OF SERVICE TPDU SIZE
4	(4)	BITSTRING	1	SC_CIOTATI_ATL	LENGTH OF SC_CIOT_AT
5	(5)	CHARACTER	32	SC_CIOTATI_AT	APPLICATION ENTITY TITLE (LOCAL)
37	(25)	CHARACTER	8	SC_CIOTATI_MOD	MODULE NAME
45	(2D)	CHARACTER	8	SC_CIOTATI_CTXT	PRESENTATION CONTEXT NAME
53	(35)	BITSTRING	1	SC_CIOTATI_QLVL	QUALITY OF SERVICE LEVEL
54	(36)	BITSTRING	1	SC_CIOTATI_T	FLAG BYTE - TYPE OF APPLICATION
		1...		SC_CIOTATI_TT	TASK
		.1..		SC_CIOTATI_TJ	JOB
55	(37)	BITSTRING	1	SC_CIOTATI_JCL	INDEX # CORRESPONDING JCL AREA
		EQUATE	X'38'	SC_CIOTATI_LEN	LENGTH OF DSECT

SC_CIOTFSS

Common name: Functional Subsystem Details

Macro ID: S@C1CIOT

DSECT name: SC_CIOTFSS

Created by: Options Processor

Location: CSA

Pointed to by: SC_CIOTPFSS

Serialization: None

Function: Functional subsystem management

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	8	SC_CIOTFSSN	NAME OF FSS (SLNET, SLJP, SLOP
8	(8)	CHARACTER	8	SC_CIOTFSSP	PROCNAME OF FSS (ABS EQU FSSN)
16	(10)	CHARACTER	8	SC_CIOTFSSS	NAME OF FSS SUPPORT MODULE
24	(18)	BITSTRING	1	SC_CIOTFSSA	FLAG BYTE - METHOD OF FSS START
		1...		SC_CIOTFSSA_A	AUTOMATICALLY ∂ SUPERLINK INIT
		.1..		SC_CIOTFSSA_O	BY OPERATOR COMMAND
		EQUATE	X'19'	SC_CIOTFSS_LEN	LENGTH OF DSECT

SC_CIOTMIC

Common name: Management interface connection details

Macro ID: S@C1CIOT

DSECT name: SC_CIOTMIC

Created by: Options Processor.

Location: CSA

Pointed to by: SC_CIOTPMIC

Serialization: None

Function: Management interface connection management

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	8	SC_CIOTMICN	NAME OF CONNECTION
8	(8)	BITSTRING	1	SC_CIOTMIC_ATL	LENGTH OF SC_CIOT_AT
9	(9)	CHARACTER	32	SC_CIOTMIC_AT	APPLICATION ENTITY TITLE(REMOTE)
41	(29)	BITSTRING	1	SC_CIOTMICA	FLAG BYTE - TYPE AND METHOD OF
		1...		SC_CIOTMICS_A	AUTOMATICALLY @ SUPERLINK INIT
		.1..		SC_CIOTMICS_O	BY OPERATOR COMMAND
		..1.		SC_CIOTMICR_A	AUTOMATICALLY BY SUPERLINK
		...1		SC_CIOTMICR_O	BY OPERATOR COMMAND
		.... 1...		SC_CIOTMICM_P	PRIMARY
		.... .1..		SC_CIOTMICM_S	SECONDARY
		.... ..1.		SC_CIOTMICT_ON	TRACE=ON (IF NEITHER SET
		1		SC_CIOTMICT_OFF	TRACE=OFF TRACE= OMITTED/NULL)
42	(2A)	CHARACTER	2	SC_CIOTMIC_MF	MAINFRAME ID
44	(2C)	FIXED	4	SC_CIOTMICI	RETRY INTERVAL (1/100 SECS)
48	(30)	CHARACTER	8	SC_CIOTMICC	CONTEXT NAME
		EQUATE	X'38'	SC_CIOTMIC_LEN	LENGTH OF DSECT

SC_CIOTS

Common name: MVS START command details

Macro ID: S@C1CIOT

DSECT name: SC_CIOTS

Created by: Options Processor (S@C1000)

Location: CSA

Pointed to by: SC_CIOTPS

Serialization: None

Function: Contains information for the automatic starting of site programs by the SUPERLINK subsystem.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	8	SC_CIOTSPRC	JCL PROCEDURE NAME FOR START CMD
8	(8)	FIXED	2	SC_CIOTSPLN	LEN OF PARAM TEXT FOR START CMD
		EQUATE	X'80'	SC_CIOTSPLN_MAX	MAX VALUE FOR SC_CIOTSPLN
10	(A)	STRUCTURE	0	SC_CIOTSPS	START OF PARAMS FOR START CMD
		EQUATE	X'8A'	SC_CIOTS_LENM	MAX LEN OF DSECT

SC_FRQE

Common name: FSS Manager Work-to-do Request Element

Macro ID: S@CFFRQE

DSECT name: SC_FRQE

Created by: Components wishing to communicate with the Functional Subsystem Manager component

Location: Common service area

Pointed to by: Request block chained from queue anchored from SC_SSVT

Serialization: None

Function: Contains requests from components of SUPERLINK to the Functional Subsystem Manager.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	SC_FRQE_ID	ACRONYM FOR FSSM REQUEST ELEMENT
4	(4)	A-ADDRESS	4	SC_FRQE_NEXT	--> NEXT REQUEST ELEMENT
8	(8)	FIXED	4	SC_FRQE_ORIGIN	REQUEST ORIGIN
12	(C)	CHARACTER	8	SC_FRQE_FSSNAME	TARGET FSS FOR REQUEST
20	(14)	BITSTRING	1	SC_FRQE_REQ	REQUESTED FUNCTION
		1...		SC_FRQE_REQ_INIT	START THE NAMED FSS *
		.1..		SC_FRQE_REQ_STOP	STOP THE NAMED FSS *
		..1.		SC_FRQE_REQ_SALL	STOP ALL FSS'S *
21	(15)	BITSTRING	1	SC_FRQE_QUAL	QUALIFIER FIELD FOR FUNCTION
		1...		SC_FRQE_QUAL_N	NORMAL TYPE STOP (IE DRAIN) *
		.1..		SC_FRQE_QUAL_Q	QUICK TYPE STOP *
		..1.		SC_FRQE_QUAL_A	ABORT TYPE STOP *
		EQUATE	241	SC_FRQE_SPOOL	SUBPOOL USED FOR FRQE
		EQUATE	X'16'	SC_FRQE_SIZE	LENGTH OF SC_FRQE

SC_FSSCB

Common name: Functional Subsystem Control Block

Macro ID: S@CFSSCB

DSECT name: SC_FSSCB

Created by: S@CF0000

Location: Common service area

Pointed to by: SC_SSVT

Serialization: Compare and swap logic should be used to manipulate fields.

Function: Describes an individual functional subsystem of SLCN.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	SC_FSSCB_ID	'FSCB' ACRONYM
4	(4)	CHARACTER	8	SC_FSSCB_NAME	FUNCTIONAL SUBSYSTEM NAME
12	(C)	A-ADDRESS	4	SC_FSSCB_SSVT	--> SC_SSVT
16	(10)	CHARACTER	4	SC_FSSCB_SNAM	NAME OF OWNING SUBSYSTEM
20	(14)	A-ADDRESS	4	SC_FSSCB_ASCB	--> ASCB FOR FSS
24	(18)	A-ADDRESS	4	SC_FSSCB_IPARMS	--> INITIALISATION TABLE FOR FSS
28	(1C)	FIXED	4	SC_FSSCB_RCONF	FSS ORDER RECEIPT CONFIRMED ECB
32	(20)	FIXED	4	SC_FSSCB_WTDECB	FSS WORK TO DO ECB
36	(24)	A-ADDRESS	4	SC_FSSCB_LISAECB	--> ECB POSTED BY LISTEN TASK
40	(28)	FIXED	4	SC_FSSCB_CDECB	ECB POSTED BY (CONN/DISCONN)ECT
44	(2C)	FIXED	2	SC_FSSCB_FSSID	FUNCTIONAL SUBSYSTEM ID
46	(2E)	BITSTRING	1	SC_FSSCB_STATUS	FSS STATUS INDICATORS #0
		1...		SC_FSSCB_STATUS_STASST	FSS HAS BEEN STARTED
		.1..		SC_FSSCB_STATUS_SFAIL	FSS START COMMAND FAILED
		..1.		SC_FSSCB_STATUS_CONCT	FSS CONNECTED SUCCESSFULLY
		...I		SC_FSSCB_STATUS_TERM	FSS TERM ORDER OUTSTANDING
		.... 1...		SC_FSSCB_STATUS_TFAIL	FSS TERM ORDER FAILED
		.... .1..		SC_FSSCB_STATUS_DISC	FSS DISCONNECTED SUCCESSFULLY
		.... ..1.		SC_FSSCB_STATUS_CTING	FSS BEING CONNECTED
		1		SC_FSSCB_STATUS_DTING	FSS BEING DISCONNECTED
		1.1.		SC_FSSCB_STATUS_ACTIV	FSS is active (STARTED and CONNECTED)

Offsets		Type	Length	Name	Description
		1111 1.1.		SC_FSSCB_STATUS_RESET	Reset all but DISCONNECTED and DICONNECTING
		1111 1111		SC_FSSCB_STATUS_ALL	ALL BITS ON - USED FOR CLEARING
47	(2F)	BITSTRING	1	SC_FSSCB_STATUS1	FSS STATUS INDICATORS #1
		1...		SC_FSSCB_STATUS1_EOM	FSS FLAGGED AS TERMINATED BY EOM
		1111 1111		SC_FSSCB_STATUS1_ALL	RESET ALL BITS FIELD
48	(30)	BITSTRING	2		RESERVED
50	(32)	CHARACTER	2		Reserved
52	(34)	A-ADDRESS	4	SC_FSSCB_FSSXB	--> FM_FSSXB
56	(38)	A-ADDRESS	4	SC_FSSCB_SERVTAB	--> SERVICE TABLE FOR FSS
60	(3C)	A-ADDRESS	4	SC_FSSCB_TCB	--> CURRENT TCB
64	(40)	A-ADDRESS	4	SC_FSSCB_AFSSM	--> FSS SUPPORT MODULE
68	(44)	FIXED	4	SC_FSSCB_FSSML	LENGTH OF FSS SUPPORT MODULE
72	(48)	A-ADDRESS	4	SC_FSSCB_MSGTAB	--> MSG TABLE CSECT
76	(4C)	CHARACTER	4		Reserved
80	(50)	STRUCTURE	0	SC_FSSCB_WORKQ	ANCHOR FOR WORK AREA Q
80	(50)	A-ADDRESS	4	SC_FSSCB_WORKQ_F	- WORK AREA Q (FIFO)
84	(54)	A-ADDRESS	4	SC_FSSCB_WORKQ_L	- WORK AREA Q (LIFO)
88	(58)	BITSTRING	16	SC_FSSCB_FUAREA	FUNCTIONAL UNIT AREA
104	(68)	STRUCTURE	0		ENSURE WE'RE MULTIPLE OF DWORD
		EQUATE	X'68'	SC_FSSCB_SIZE	LENGTH OF SC_FSSCB
		EQUATE	241	SC_FSSCB_SPOOL	SUBPOOL TO USE
		EQUATE	X'00'	SC_FSSCB_ACRONYM	KEPT FOR COMPATIBILITY
88	(58)	A-ADDRESS	4	SC_FSSCB_IST	INTERNAL SERVICE TABLE
92	(5C)	A-ADDRESS	4	SC_FSSCB_NST	NETWORK SERVICE TABLE
96	(60)	A-ADDRESS	4	SC_FSSCB_NSP	NETWORK STORAGE POOL
100	(64)	A-ADDRESS	4		RESERVED

SC_GST

Common name: Global Service Table

Macro ID: S@CC0GST

DSECT name: SC_GST

Created by: S@CC0000

Location: Common service area

Pointed to by: SC_SSVT

Serialization: None

Function: Provides a vector table for global SUPERLINK services.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	SC_GST_ID	'GST ' CONSTANT
4	(4)	A-ADDRESS	4	SC_GST_SSVT	--> SC_SSVT
8	(8)	A-ADDRESS	4	SC_GST_SBY	--> SLSUBSYS SERVICE
12	(C)	A-ADDRESS	4	SC_GST_LOGQ	--> LOG QUEUE SERVICE
16	(10)	A-ADDRESS	4	SC_GST_AMGR	--> ASSOC. MANAGER SERVICE
20	(14)	A-ADDRESS	4	SC_GST_MSG	--> MESSAGE PROCESSOR
24	(18)	A-ADDRESS	4	SC_GST_SCHEDSRB	--> S@CF0100 TO SCHEDULE SRB
28	(1C)	A-ADDRESS	4	SC_GST_CASE	--> CASE SERVICE
32	(20)	A-ADDRESS	4	SC_GST_TRANSLATE	--> GLOBAL TRANSLATE TABLES
36	(24)	A-ADDRESS	4	SC_GST_RESMAN	--> S@CC0URM RESOURCE MANAGER
40	(28)	A-ADDRESS	8	SC_GST_RESERVED	RESERVED FIELDS FOR LATER USE
		EQUATE	X'30'	SC_GST_SIZE	LENGTH OF SC_GST
		EQUATE	241	SC_GST_SPOOL	SUBPOOL WHERE HE LIVES

SC_OPCB

Common name: Operator Command Buffer

Macro ID: S@CCOPCB

DSECT name: SC_OPCB

Created by: Any component wishing to communicate with the Product Operator component

Location: Common service area

Pointed to by: Chained from a queue anchored from SC_SSVT

Serialization: None

Function: Conveys requests from SUPERLINK components to the Operator component of SLCN.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	SC_OPCB_ID	'OPCB' CONSTANT
4	(4)	A-ADDRESS	4	SC_OPCB_SPOOL	SUBPOOL WHERE COMMAND
8	(8)	A-ADDRESS	4	SC_OPCB_NXTC	--> NEXT COMMAND BUFFER IN Q
12	(C)	BITSTRING	1	SC_OPCB_FLAGS	SOME FLAG FIELDS
		1...		SC_OPCB_FLAGSMIC	ORIGIN IS MANGMT. I/F CONNECTN.
13	(D)	BITSTRING	3		RESERVED FOR LATER USE
16	(10)	STRUCTURE	0	SC_OPCB_ORIGIN	COMMAND ORIGIN - MVS FORM
16	(10)	BITSTRING	3	SC_OPCB_CONSFLAG	ZERO IF ORIGIN IS CONSOLE
19	(13)	BITSTRING	1	SC_OPCB_CONSOLID	CONSOLE ID IF FLAG = 0
16	(10)	BITSTRING	2	SC_OPCB_TSOASID	ASID IF ORIGIN IS TSO USER
18	(12)	BITSTRING	1	SC_OPCB_TSOFLAG	AUTHORISATION IF TSOUSER
		1		SC_OPCB_TSOAUTH	AUTH FLAG IN TSOFLAG FIELD
16	(10)	CHARACTER	8	SC_OPCB_MIFID	MANAGEMENT I/F CONNECTION ID
24	(18)	BITSTRING	1	SC_OPCB_MIFSULN	LENGTH OF SOURCE USER DATA FIELD
25	(19)	BITSTRING	20	SC_OPCB_MIFSUD	SOURCE USER DATA FIELD
45	(2D)	CHARACTER	1		Reserved
46	(2E)	FIXED	2	SC_OPCB_LEN	LENGTH OF OPERATOR COMMAND
48	(30)	CHARACTER	136	SC_OPCB_COMMAND	OPERATOR COMMAND
		EQUATE	X'B8'	SC_OPCB_SIZE	LENGTH OF SC_OPCB
		EQUATE	X'88'	SC_OPCB_CMDSIZE	LEN OF SC_OPCB COMMAND (MAX)
		EQUATE	241	SC_OPCB_POOL	SUBPOOL USED FOR OPCB

SC_SSVT

Common name: SUPERLINK Subsystem Vector Table

Macro ID: S@CCSSVT

DSECT name: SC_SSVT

Created by: S@CC0SSI

Location: Common service area

Pointed to by: SSCVT (IBM subsystem control block)

Serialization: Compare and swap logic is used to manipulate fields.

Function: This is the main control block of the SUPERLINK product, identifying it as an MVS subsystem.

Offsets	Type	Length	Name	Description
0 (0)	FIXED	2	SC_SSVT_RSV1	RESERVED
2 (2)	FIXED	2	SC_SSVT_FNUM	# FUNCTIONS SUPPORTED BY SUBSYS
4 (4)	BITSTRING	256	SC_SSVT_MATRIX	FUNCTION CODE MATRIX
	EQUATE	X'104'	SC_SSVT_FSIZE	LENGTH OF SC_SSVT FIXED PART
260 (104)	A-ADDRESS	4	SC_SSVT_ADDRS	FIRST ADDRESS ENTRY IN TABLE
264 (108)	A-ADDRESS	1020		Rest of table
	EQUATE	X'504'	SC_SSVT_MAX_SIZE	MAXIMUM SIZE OF SSVT HEADER
SUPERLINK supported functions header				
2 (2)	Y-ADDRESS	2		
SUPERLINK supported functions matrix				
7 (7)	A-ADDRESS	1		Address offset #04 - End of task
11 (B)	A-ADDRESS	1		Address offset #08 - End of memory
13 (D)	A-ADDRESS	1		Address offset #10 - SVC 34
56 (38)	A-ADDRESS	1		Address offset #53 - FSS processing
SUPERLINK supported functions vector table				
	EQUATE	X'100'	SC_SSVT_ORIGIN	MATRIX BASE OFFSET

Offsets	Type	Length	Name	Description
260 (104)	A-ADDRESS	4	SC_SSVT_S@CC0EOT	END OF TASK PROCESSING
264 (108)	A-ADDRESS	4	SC_SSVT_S@CC0EOM	END OF MEMORY PROCESSING
268 (10C)	A-ADDRESS	4	SC_SSVT_S@CC0S34	SVS34 - OPERATOR COMMAND PROC.
272 (110)	A-ADDRESS	4	SC_SSVT_S@CC0FSS	FSS CONNECT/DISCONNECT PROCING
	EQUATE	X'110'	SC_SSVT_LAST	LAST FUNCTION DEFINED
276 (114)	A-ADDRESS	40		SOME SPARE LOCATIONS
316 (13C)	CHARACTER	4	SC_SSVT_ID	SC_SSVT CONTROL BLOCK ACRONYM
320 (140)	A-ADDRESS	4	SC_SSVT_SSCT	--> SSCVT FOR SUBSYSTEM
324 (144)	A-ADDRESS	4	SC_SSVT_ASCB	--> ASCB FOR SUBSYSTEM
328 (148)	A-ADDRESS	4	SC_SSVT_CIOT	--> SC_CIOT
332 (14C)	A-ADDRESS	4	SC_SSVT_GST	--> SC_GST
336 (150)	A-ADDRESS	4	SC_SSVT_OPCT	--> SI_OPCT
340 (154)	A-ADDRESS	4	SC_SSVT_AMT	--> ASSOCIATION MANAGER TABLE
344 (158)	FIXED	4	SC_SSVT_COND	CONDITION OF SUBSYSTEM
	....		SC_SSVT_COND_UP	SUBSYSTEM ACTIVE
	EQUATE	X'80000000'	SC_SSVT_COND_AB	----"---- ABENDED OR ABENDING
	EQUATE	X'40000000'	SC_SSVT_COND_RP	----"---- RECOVERY IN PROGRESS
	EQUATE	X'0FFFFFFF'	SC_SSVT_COND_TR	----"---- SL TERM. REQUESTED
348 (15C)	STRUCTURE	0	SC_SSVT_STATUS	SUBSYSTEM STATUS INDICATORS
348 (15C)	BITSTRING	1	SC_SSVT_STATUS1	*1 STATUS 1ST BYTE (SUBSYSTEM)
	1...		SC_SSVT_STATUS1_INIT	SUBSYS IN INITIALISATION PHASE
	.1..		SC_SSVT_STATUS1_TERM	SUBSYS IN TERMINATION PHASE
	..1.		SC_SSVT_STATUS1_REDY	SUBSYS IN READY STATE
349 (15D)	BITSTRING	1	SC_SSVT_STATUS2	*2 STATUS 2ND BYTE (GENERAL)
	1...		SC_SSVT_STATUS2_DUMP	GENERAL DUMP ON ERROR ENABLED
350 (15E)	BITSTRING	1	SC_SSVT_STATUS3	*3 STATUS 3RD BYTE (FSS SUPPORT)
	1...		SC_SSVT_STATUS3_CDF	FSS CONNECT/DISCONNECT FAILURE
	.1..		SC_SSVT_STATUS3_FMT	FSS MANAGER TO TERMINATE FLAG
	..1.		SC_SSVT_STATUS3_FAC	FSS MANAGER IS ACTIVE
	...1		SC_SSVT_STATUS3_LAC	LOG MANAGER IS ACTIVE
	.... 1...		SC_SSVT_STATUS3_OAC	OPR MANAGER IS ACTIVE

Offsets	Type	Length	Name	Description
	.... .1..		SC_SSVT_STATUS3_AAC	ASSOC. MANAGER IS ACTIVE
	.... ..1.		SC_SSVT_STATUS3_MAC	MANAGEMENT I/F IS ACTIVE
351 (15F)	BITSTRING	1	SC_SSVT_STATUS4	*4 STATUS 4TH BYTE (RESERVED)
352 (160)	CHARACTER	1	SC_SSVT_CMDCHAR	SUBSYSTEM COMMAND CHARACTER
353 (161)	BITSTRING	1	SC_SSVT_TERMTYPE	TERMINATION TYPE
	....		SC_SSVT_NORMAL	NORMAL TERMINATION
	1		SC_SSVT_QUICK	QUICK TERMINATION
	.... ..1.		SC_SSVT_ABORT	ABORT TERMINATION
354 (162)	BITSTRING	6		RESERVED
360 (168)	STRUCTURE	0	SC_SSVT_CMDQ	ANCHORS FOR OP COMMAND QUEUES
360 (168)	A-ADDRESS	4	SC_SSVT_CMDQ_LI	-> OP CMD QUEUE (LIFO)
364 (16C)	A-ADDRESS	4	SC_SSVT_CMDQ_FI	-> RE-ORDERED CMD QUEUE (FIFO)
368 (170)	STRUCTURE	0	SC_SSVT_FSMQ	ANCHORS FOR FSS MNGR.REQ. QUEUES
368 (170)	A-ADDRESS	4	SC_SSVT_FSMQ_LI	-> REQ. QUEUE (LIFO)
372 (174)	A-ADDRESS	4	SC_SSVT_FSMQ_FI	-> RE-ORDERED REQ. QUEUE (FIFO)
376 (178)	STRUCTURE	0	SC_SSVT_SNDQ	ANCHORS FOR 'SEND' WORK QUEUES
376 (178)	A-ADDRESS	4	SC_SSVT_SNDQ_LI	-> SEND REQ. QUEUE (LIFO)
380 (17C)	A-ADDRESS	4	SC_SSVT_SNDQ_FI	-> RE-ORDERED REQ. QUEUE (FIFO)
384 (180)	BITSTRING	8	SC_SSVT_LOGQ_F	ANCHOR FOR SUBSYS LOG FREE Q
	1...		SC_SSVT_LOGQ_T	LOG PROCESSOR TERMINATING
392 (188)	BITSTRING	8	SC_SSVT_LOGQ_R	ANCHOR FOR SUBSYS LOG REQ Q
400 (190)	BITSTRING	8	SC_SSVT_LOGQ_W	ANCHOR FOR SUBSYS LOG WORK Q
408 (198)	A-ADDRESS	4	SC_SSVT_ECB_LIST	MAIN WAIT ECB LIST
	.... .1..		SC_SSVT_ECB_ELEN	AN ENTRY LENGTH
412 (19C)	A-ADDRESS	80		ECBLIST POINTERS
	.1.1 .1..		SC_SSVT_ECB_LEN	SIZE OF LIST IN BYTES
	...1 .1.1		SC_SSVT_ECB_SIZE	# IN LIST
492 (1EC)	FIXED	4	SC_SSVT_PECB	SUBSYS TERMINATION ECB
496 (1F0)	FIXED	4	SC_SSVT_OPR_PECB	OPERATOR COMPONENT TASK END ECB
500 (1F4)	FIXED	4	SC_SSVT_OPR_WECB	OPERATOR COMPT. WORK TO DO ECB
504 (1F8)	FIXED	4	SC_SSVT_OPR_IECB	OPERATOR INIT. COMPLETE ECB
508 (1FC)	A-ADDRESS	4	SC_SSVT_OPR_TCB	-> TCB OF OPERATOR COMPONENT

Offsets	Type	Length	Name	Description
512 (200)	FIXED	4	SC_SSVT_LOG_PECB	LOG COMPONENT TASK ENDED ECB
516 (204)	FIXED	4	SC_SSVT_LOG_IECB	LOG COMPONENT INITIALISED ECB
520 (208)	A-ADDRESS	4	SC_SSVT_LOG_TCB	-> TCB OF LOG COMPONENT
524 (20C)	STRUCTURE	0	SC_SSVT_LOG_ORDR	LOG COMPONENT ORDER FIELD
524 (20C)	BITSTRING	3		RESERVED FOR LOG COMPONENT
527 (20F)	BITSTRING	1	SC_SSVT_LOG_CMND	LOG COMPONENT REQUEST FIELD
528 (210)	A-ADDRESS	4	SC_SSVT_LOG_ADDN	LOG COMPONENT -> SWITCH DDNAME
532 (214)	CHARACTER	8	SC_SSVT_LOG_SWDD	LOG SWITCH NEW DDNAME
540 (21C)	FIXED	4	SC_SSVT_FSS_PECB	FSS MANAGER TASK ENDED ECB
544 (220)	FIXED	4	SC_SSVT_FSS_IECB	FSS MANAGER INIT.COMPLETE ECB
548 (224)	FIXED	4	SC_SSVT_FSS_CECB	FSS CONTINUE ECB - SLCN POSTS
552 (228)	FIXED	4	SC_SSVT_FSS_RECB	FSS MANAGER REQUEST TERM. ECB
556 (22C)	FIXED	4	SC_SSVT_FSS_WECB	FSS MANAGER WORK TO DO ECB
560 (230)	FIXED	4	SC_SSVT_FSS_FECB	FSS MANAGER FSS TERMINATED ECB
564 (234)	FIXED	4	SC_SSVT_NET_IECB	SLNET COMPNT. INIT.COMPLETE ECB
568 (238)	A-ADDRESS	4	SC_SSVT_LIST_TCB	-> TCB OF LISTEN TASK
572 (23C)	A-ADDRESS	4	SC_SSVT_LIST_ECB	-> TERM. REQUEST ECB FOR LISTEN
576 (240)	A-ADDRESS	4	SC_SSVT_SND_ECB	-> 'SEND' REQUEST ARRIVED ECB
580 (244)	A-ADDRESS	4	SC_SSVT_RECSRBEP	-> ENTRY POINT FOR RECEIVE SRB
584 (248)	A-ADDRESS	4	SC_SSVT_RECFRREP	-> EP FOR RECEIVE SRB FRR
588 (24C)	A-ADDRESS	4	SC_SSVT_OPT_EP	-> EP FOR OPTIONS PROCESSOR
592 (250)	A-ADDRESS	4	SC_SSVT_FSSM_TCB	-> TCB OF FSS MANAGER TASK
596 (254)	CHARACTER	4	SC_SSVT_SNAM	SUBSYS NAME AS KNOWN TO MVS
600 (258)	CHARACTER	8	SC_SSVT_ANAM	SUBSYS NAME AS KNOW TO APPL PROG
608 (260)	A-ADDRESS	4	SC_SSVT_FSCQ	ANCHOR FOR FSS CONTROL BLOCKS
612 (264)	FIXED	4	SC_SSVT_FSCL	TOTAL LENGTH OF SC_FSSCB TABLE
616 (268)	FIXED	4	SC_SSVT_FSCN	# FSSCB'S IN CHAIN
620 (26C)	A-ADDRESS	4	SC_SSVT_FSCELIST	--> ECBLIST FOR FSS TERMINATION

Offsets	Type	Length	Name	Description
624 (270)	A-ADDRESS	4	SC_SSVT_SBUF	--> FSS START COMMAND BUFFER
628 (274)	A-ADDRESS	4	SC_SSVT_MSGTAB	--> MESSAGE TABLE CSECT
632 (278)	A-ADDRESS	4	SC_SSVT_XMEMTECB	--> X-MEMORY PREMATURE TERM ECB
636 (27C)	A-ADDRESS	4	SC_SSVT_XMEMCECB	--> X-MEMORY CONTINUE ECB
640 (280)	FIXED	4	SC_SSVT_SLASVTLN	LENGTH OF SC_SLASVT TABLE
644 (284)	A-ADDRESS	4	SC_SSVT_ASLASVT	--> SC_SLASVT CONTROL BLOCK
648 (288)	FIXED	20		RESERVED FOR LATER USE
668 (29C)	FIXED	4	SC_SSVT_ASM_PECB	ASSOC. MGR. TASK ENDED ECB
672 (2A0)	FIXED	4	SC_SSVT_ASM_IECB	ASSOC. MGR. INITIALISED ECB
676 (2A4)	A-ADDRESS	4	SC_SSVT_ASM_TCB	-> TCB OF ASSOC. MGR. TASK
680 (2A8)	FIXED	4	SC_SSVT_MIF_PECB	MANAGMENT I/F TASK ENDED ECB
684 (2AC)	FIXED	4	SC_SSVT_MIF_IECB	MANAGEMENT I/F INITIALISED ECB
688 (2B0)	A-ADDRESS	4	SC_SSVT_MIF_TCB	-> TCB OF MANAGEMENT I/F TASK
692 (2B4)	A-ADDRESS	4	SC_SSVT_MICT	-> MANAGEMENT I/F CONTROL TABLE
696 (2B8)	FIXED	4	SC_SSVT_FSS_TECB	FSS TERMINATE CONTINUE ECB
700 (2BC)	A-ADDRESS	4	SC_SSVT_CPOOLID	CPOOL ID FOR SC_URE CPOOL GETS
704 (2C0)	FIXED	4	SC_SSVT_NAMAVAIL	NAM AVAILABLE FOR USE ECB
708 (2C4)	FIXED	32	SC_SSVT_RESERVED	RESERVED FOR EXPANSION
740 (2E4)	CHARACTER	4		Reserved
744 (2E8)	STRUCTURE	0		JUST ALIGN
	EQUATE	X'504'	SC_SSVT_SIZE	LENGTH OF SC_SSVT INC VAR PART
	EQUATE	241	SC_SSVT_SPOOL	SUBPOOL TO BE USED ON GETMAIN

SC_SLASVT

Common name: SUPERLINK Address Space Vector Table

Macro ID: S@CCASVT

DSECT name: SC_SLASVT

Created by: S@CC0000

Location: Common service area

Pointed to by: SC_SSVT

Serialization: The CMS lock must be held when manipulating this control block.

Function: Provides a list of the SUPERLINK resources used by address spaces, on an address space basis.

Offsets	Type	Length	Name	Description
0 (0)	STRUCTURE	0	SC_SLASVT_START	START OF THE VECTOR TABLE
	EQUATE	241	SC_SLASVT_SPOOL	SUBPOOL USED FOR TABLE

SC_URE

Common name: User Resource Element

Macro ID: S@CC0URE

DSECT name: SC_URE

Created by: S@CC0URM

Location: Common service area

Pointed to by: SC_SLASVT

Serialization: The CMS lock must be held when manipulating the SC_URE queues and when updating the SC_URE.

Function: Represents a task, in a particular address space, using SUPERLINK resources.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	SC_URE_ID	'URE ' EYECATCHER
4	(4)	A-ADDRESS	4	SC_URE_FPTR	--> NEXT SC_URE ON THE QUEUE
8	(8)	A-ADDRESS	4	SC_URE_BPTR	--> PREVIOUS SC_URE ON THE QUEUE
12	(C)	A-ADDRESS	4	SC_URE_ATCB	--> TCB ASSOC. WITH THIS ENTRY
16	(10)	FIXED	4	SC_URE_AID	ASSOCIATION IDENTIFIER
20	(14)	BITSTRING	1	SC_URE_FLAG	STATUS FLAGS FOR THIS SC_URE
		1...		SC_URE_FLAG_EOT	END OF TASK FLAGGED FOR THIS URE
21	(15)	BITSTRING	1		RESERVED FOR LATER USE
22	(16)	FIXED	2	SC_URE_ASID	ADDRESS SPACE ID FOR THIS ENTRY
24	(18)	A-ADDRESS	4	SC_URE_ASCB	-> ASCB FOR THIS ENTRY
28	(1C)	FIXED	4		RESERVED FOR LATER USE
32	(20)	STRUCTURE	0		ROUND UP TO A DOUBLE WORD
		EQUATE	X'20'	SC_URE_SIZE	LENGTH OF SC_URE
		EQUATE	241	SC_URE_SPOOL	SUBPOOL OF CPOOL FOR SC_URE

SC_URM

Common name: User Resource Manager Entry Parameter List and Feedback Area

Macro ID: S@CC0URM

DSECT name: SC_URM

Created by: ACSE

Location: Private storage

Pointed to by: Register 1 on entry

Serialization: None

Function: Provides the parameter list for requesting services from the User Resource Manager and is used to hold the return code.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	SC_URM_ID	'URM ' EYECATCHER
4	(4)	A-ADDRESS	4	SC_URM_SSVT	--> SC_SSVT
8	(8)	BITSTRING	1	SC_URM_RFC	REQUESTED FUNCTION CODE
		....		SC_URM_RFC_LOC	LOCATE
		1		SC_URM_RFC_Q	QUEUE
		.... ..1.		SC_URM_RFC_DEQ	DEQUEUE
		11		SC_URM_RFC_SWI	SWITCH
9	(9)	BITSTRING	1		RESERVED
10	(A)	FIXED	2	SC_URM_ASID	ADDRESS SPACE ID FOR THIS ENTRY
12	(C)	A-ADDRESS	4	SC_URM_ASCB	--> ASCB FOR THIS ENTRY
16	(10)	A-ADDRESS	4	SC_URM_TCB	--> TCB ASSOC. WITH THIS ENTRY
20	(14)	A-ADDRESS	4	SC_URM_AID	ASSOCIATION IDENTIFIER
24	(18)	A-ADDRESS	4	SC_URM_URE	--> MODEL URE
28	(1C)	BITSTRING	1	SC_URM_RC	RETURN CODE
		.... 1...		SC_URM_RC_SSVT	SC_SSVT NOT FOUND
		.... 11..		SC_URM_RC_SLASVT	SC_SLASVT NOT FOUND
		...1		SC_URM_RC_URM	INVALID SC_URM
		...1 .1..		SC_URM_RC_RFC	INVALID RFC
		...1 1...		SC_URM_RC_LOC	NO SC_URE LOCATED
		...1 11..		SC_URM_RC_URE	INVALID SC_URE
		..1.		SC_URM_RC_GETM	GETMAIN FAILURE AC000071
29	(1D)	BITSTRING	3		RESERVED
		EQUATE	X'20'	SC_URM_SIZE	LENGTH OF SC_URM

SI_CMD

Common name: Command Definition Table

Macro ID: S@COCMAP

DSECT name: SI_CMD

Created by: Built at assembly time

Location: SLCN private area

Link-edited into the Product operator component. Not available

Serialization: None

Function: Maps operator command definitions built at assembly time

Offsets		Type	Length	Name	Description
0	(0)	FIXED	2	SI_CMD_LENGTH	LENGTH OF AN ENTRY
2	(2)	BITSTRING	1	SI_CMD_FORMAT	FORMAT OF THIS ENTRY
		1...		SI_CMD_POSITION	POSITIONAL PARAMETER
		.1..		SI_CMD_KEYWORD	KEYWORD PARAMETER
		..1.		SI_CMD_LITERAL	LITERAL OPERAND
		...1		SI_CMD_ALTERNATE	ALTERNATE OPERAND LIST
		.... 1...		SI_CMD_COMMAND	COMMAND DEFINITION
3	(3)	BITSTRING	1	SI_CMD_FLAGS	FLAG BYTE
		1...		SI_CMD_TSOUSER	MAY BE ISSUED BY TSO USER
		.1..		SI_CMD_OPERATOR	MAY BE ISSUED BY OPERATOR
4	(4)	STRUCTURE	0	SI_CMD_HEAD_END	END OF COMMON FIXED HEADER
		EQUATE	X'4'	SI_CMD_HEAD_SIZE	LENGTH OF SI_CMD FIXED HEADER

Extension to SI_CMD for command table entries themselves

4	(4)	FIXED	2	SI_CMD_C_OPERNUM	NUMBER OF OPERAND TABLES PRESENT
6	(6)	FIXED	2	SI_CMD_C_MAXLEN	LENGTH OF COMMAND NAME
8	(8)	FIXED	2	SI_CMD_C_MINLEN	LENGTH OF MINIMUM ABREVIATION
10	(A)	CHARACTER	20	SI_CMD_C_NAME	COMMAND NAME ITSELF
		.... .1..		SI_CMD_C_OFFSET	
30	(1E)	CHARACTER	2		Reserved
32	(20)	A-ADDRESS	4	SI_CMD_C_MODULE	-> PROCESSING MODULE FOR COMMAND
36	(24)	A-ADDRESS	0	SI_CMD_C_OPANDS	START OF OPERAND TABLE ADDRESSES

Offsets		Type	Length	Name	Description
Extension to SI_CMD for alternate operand list table entries					
4	(4)	FIXED	2	SI_CMD_A_ALTNUM	NUMBER OF ALTERNATE LISTS
6	(6)	FIXED	2	SI_CMD_A_DESCLEN	LENGTH OF DESCRIPTION
8	(8)	FIXED	2		RESERVED
10	(A)	CHARACTER	28	SI_CMD_A_DESC	DESCRIPTION
38	(26)	CHARACTER	2		Reserved
40	(28)	STRUCTURE	0		ALIGN ON WORD BOUNDARY
40	(28)	FIXED	2	SI_CMD_A_OPERLEN	LENGTH OF OPERAND LIST
42	(2A)	FIXED	2	SI_CMD_A_OPERNUM	NUMBER OF OPERANDS IN LIST
44	(2C)	A-ADDRESS	0	SI_CMD_A_OPERAND	START OF OPERAND TABLE ADDRESSES
Extension to SI_CMD for keyword operand definition entries					
4	(4)	CHARACTER	4	SI_CMD_K_TYPE	KEYWORD OPERAND TYPE
8	(8)	FIXED	2	SI_CMD_K_MAXLEN	MAX LENGTH OF KEYWORD
10	(A)	FIXED	2	SI_CMD_K_MINLEN	MIN LENGTH OF KEYWORD
12	(C)	CHARACTER	20	SI_CMD_K_NAME	NAME OF KEYWORD
32	(20)	FIXED	2	SI_CMD_K_DESCLEN	LENGTH OF DESCRIPTION TEXT
34	(22)	FIXED	2		RESERVED
36	(24)	CHARACTER	28	SI_CMD_K_DESC	DESCRIPTION TEXT ITSELF
64	(40)	BITSTRING	8	SI_CMD_K_OINST	INSTRUCTION TO INDICATE KEYWORD
72	(48)	FIXED	4	SI_CMD_K_XVALUE	X-value: For ._K_TYPE='N' this is minimum value
76	(4C)	FIXED	4	SI_CMD_K_YVALUE	Y-value: For ._K_TYPE='N' this is maximum value, otherwise this is maximum string length
80	(50)	BITSTRING	8	SI_CMD_K_INST	INST. TO MOVE VALUE TO SYM TAB
88	(58)	FIXED	2	SI_CMD_K_DEFLEN	LENGTH OF DEFAULT VALUE
90	(5A)	CHARACTER	2		Reserved
92	(5C)	STRUCTURE	0		ENSURE ALIGNMENT OF DEFVAL
92	(5C)	CHARACTER	28	SI_CMD_K_DEFVAL	DEFAULT OPERAND VALUE
120	(78)	A-ADDRESS	0	SI_CMD_K_ENTEND	END OF KEYWORD ENTRY

Offsets		Type	Length	Name	Description
Extension to SI_CMD for positional operand definition entries					
4	(4)	CHARACTER	4	SI_CMD_P_TYPE	POSITIONAL OPERAND TYPE
8	(8)	FIXED	2	SI_CMD_P_DESCLEN	LENGTH OF DESCRIPTION TEXT
10	(A)	FIXED	2		RESERVED
12	(C)	CHARACTER	28	SI_CMD_P_DESC	DESCRIPTION TEXT ITSELF
40	(28)	BITSTRING	8	SI_CMD_P_OINST	INSTRUCTION TO INDICATE PRESENCE
48	(30)	FIXED	4	SI_CMD_P_XVALUE	X-VALUE
52	(34)	FIXED	4	SI_CMD_P_YVALUE	Y-VALUE
56	(38)	BITSTRING	8	SI_CMD_P_INST	INST. TO MOVE VALUE TO SYM TAB
64	(40)	FIXED	2	SI_CMD_P_DEFLEN	LENGTH OF DEFAULT VALUE
66	(42)	CHARACTER	2		Reserved
68	(44)	STRUCTURE	0		ENSURE ALIGNMENT OF DEFVAL
68	(44)	CHARACTER	28	SI_CMD_P_DEFVAL	DEFAULT OPERAND VALUE
96	(60)	A-ADDRESS	0	SI_CMD_P_ENTEND	END OF POSITIONAL ENTRY
Extension to SI_CMD for literal operand definition entries					
4	(4)	CHARACTER	4	SI_CMD_L_TYPE	LITERAL OPERAND TYPE
8	(8)	FIXED	2	SI_CMD_L_MAXLEN	MAXIMUM LITERAL LENGTH
10	(A)	FIXED	2	SI_CMD_L_MINLEN	MINIMUM LITERAL LENGTH
12	(C)	CHARACTER	20	SI_CMD_L_NAME	LITERAL NAME
32	(20)	BITSTRING	8	SI_CMD_L_OINST	INSTRUCTION TO INDICATE LITERAL
40	(28)	A-ADDRESS	0	SI_CMD_L_ENTEND	END OF LITERAL ENTRY
		EQUATE	X'78'	SI_CMD_MAXLENGTH	MAXIMUM LENGTH OF A TABLE

SI_OPCT

Common name: Operator Control Table

Macro ID: S@COOPCT

DSECT name: SI_OPCT

Created by: S@CO0000

Location: SLCN private storage

Pointed to by: SC_SSVT_OPCT

Serialization: None

Function: Controls the processing of an individual command by the Product Operator component.

Offsets		Type	Length	Name	Description
0	(0)	CHARACTER	4	SI_OPCT_ID	'OPCT' CONSTANT ACRONYN
4	(4)	FIXED	4	SI_OPCT_TERM	ECB FOR TERMINATION OF OPERATOR
		.... 1...		SI_OPCT_CLEAR	CLEAR FROM HERE ON RESET
8	(8)	A-ADDRESS	4	SI_OPCT_CMDBUF	--> COMMAND BUFFER
12	(C)	A-ADDRESS	4	SI_OPCT_BUFP	--> INTO COMMAND BUFFER DATA
16	(10)	FIXED	4	SI_OPCT_CHARS	# CHARS REMAINING IN BUFFER
20	(14)	FIXED	4	SI_OPCT_MSGNUM	MESSAGE NUMBER TO BUILD MSG TEXT
24	(18)	A-ADDRESS	4	SI_OPCT_DESCP	--> CURRENT DESCRIPTION DATA
28	(1C)	BITSTRING	1	SI_OPCT_WHO	ISSUER OF COMMAND
		1...		SI_OPCT_WHO_OPER	OPERATOR
		.1..		SI_OPCT_WHO_AUSR	AUTHORISED USER
		..1.		SI_OPCT_WHO_TSOU	TSO USER
		...1		SI_OPCT_WHO_MICN	MANAGEMENT INTERFACE CONNECTION
29	(1D)	BITSTRING	1	SI_OPCT_FLAG	SOME FLAGS
		1...		SI_OPCT_FLAGTEXT	FLAG LAST OPERAND AS 'TEXT' TYPE
		.1..		SI_OPCT_FLAGKMAT	FLAG LAST OPERAND AS KWORD TYPE
		..1.		SI_OPCT_FLAGCMNT	FLAG COMMENT PRESENT ON COMMAND
30	(1E)	CHARACTER	2		Reserved
32	(20)	FIXED	4	SI_OPCT_CONSOLE	CONSOLE ID OF COMMAND ORIGIN
36	(24)	CHARACTER	8	SI_OPCT_TSOUSER	TSO ID OF COMMAND ORIGIN
44	(2C)	CHARACTER	8	SI_OPCT_TERMID	TERMINAL ID OF ORIGIN
52	(34)	CHARACTER	8	SI_OPCT_MICONID	MANAGEMENT I/F CONNECTION

Offsets	Type	Length	Name	Description
60 (3C)	BITSTRING	1	SI_OPCT_MICONUL	MGMT. I/F SOURCE USER DATA LEN.
61 (3D)	BITSTRING	20	SI_OPCT_MICONUD	MGMT. I/F SOURCE USER DATA
81 (51)	CHARACTER	3		Reserved
84 (54)	A-ADDRESS	4	SI_OPCT_SYNTX	--> SYNTAX GRAPH OF THIS COMMAND
88 (58)	A-ADDRESS	4	SI_OPCT_OPLIST	--> CURRENT OPERAND LIST
92 (5C)	FIXED	2	SI_OPCT_OPLEFT	# OPERANDS LEFT TO PROCESS
94 (5E)	FIXED	2	SI_OPCT_COMMENTL	LENGTH OF COMMENT IF ANY
96 (60)	A-ADDRESS	4	SI_OPCT_COMMENTP	--> COMMENT IN COMMEND IF ANY
100 (64)	A-ADDRESS	4	SI_OPCT_HEADSTAK	--> HEAD OF SI_STAK STACK
104 (68)	A-ADDRESS	4	SI_OPCT_CURRSTAK	--> CURRENT STACK ENTRY
108 (6C)	A-ADDRESS	4	SI_OPCT_FREESTAK	--> NEXT FREE STACK ELEMENT
112 (70)	BITSTRING	8	SI_OPCT_PACKWORK	WORK AREA FOR PACK
120 (78)	A-ADDRESS	4	SI_OPCT_MSGPTR	-> MESSAGE FOR S@C00050
124 (7C)	A-ADDRESS	4	SI_OPCT_PAGBUF	-> PAGE BUFFER FOR MULTILINE MSG
128 (80)	A-ADDRESS	4	SI_OPCT_PAGBUFP	-> CURRENT LOCATION IN BUFFER
132 (84)	FIXED	4	SI_OPCT_OLDEST	OLDEST DISPLAY AREA TIME
136 (88)	FIXED	4	SI_OPCT_LINECNT	CURRENT COUNT OF LINES IN MSG
140 (8C)	FIXED	4	SI_OPCT_WTOMNUM	WTO RETURNED MULT-LINE MSG #
144 (90)	FIXED	0	SI_OPCT_MSG	FORMATTED MESSAGE BUFFER
144 (90)	FIXED	2	SI_OPCT_MSGTLEN	LENGTH FIELD FOR MESSAGE TEXT
146 (92)	CHARACTER	125	SI_OPCT_MSGTEXT	FORMATTED MESSAGE FIELD
271 (10F)	BITSTRING	1	SI_OPCT_FUNC	FUNCTION CODE FOR S@C00050
272 (110)	BITSTRING	1	SI_OPCT_DISPLAY	DISPLAY AREA IDENTIFIER
273 (111)	CHARACTER	1		Reserved
274 (112)	FIXED	0	SI_OPCT_WTOPARM	ALIGN'PARM BUFFER
274 (112)	CHARACTER	125	SI_OPCT_WTOBUF	BUFFER FOR WTO PARM LIST
399 (18F)	CHARACTER	1		Reserved
400 (190)	STRUCTURE	0	SI_OPCT_SYM	START OF SYMBOL TABLE
400 (190)	STRUCTURE	0	SI_OPCT_P_XXXXXX	POSITIONAL OPERANDS

Offsets	Type	Length	Name	Description
400 (190)	CHARACTER	8	SI_OPCT_P_MICON	SEND MICON PARAMETER
408 (198)	BITSTRING	0	SI_OPCT_Q_XXXXXX	POSITIONAL FLAGS
	EQUATE	X'10'	SI_OPCT_Q_MICON	'MICON' PRESENT
408 (198)	STRUCTURE	0	SI_OPCT_K_XXXXXX	KEYWORD OPERANDS
408 (198)	CHARACTER	8	SI_OPCT_K_FU	FU NAME VALUE
416 (1A0)	CHARACTER	8	SI_OPCT_K_OPTIONS	OPTIONS NAME VALUE
424 (1A8)	CHARACTER	8	SI_OPCT_K_APL	APL NAME VALUE
432 (1B0)	CHARACTER	8	SI_OPCT_K_MI	MI NAME VALUE
440 (1B8)	CHARACTER	8	SI_OPCT_K_JOB	JOBNAME VALUE
448 (1C0)	CHARACTER	8	SI_OPCT_K_DDNAME	DDNAME NAME VALUE
	EQUATE	X'1C1'	SI_OPCT_K_DD	ALIAS FOR DDNAME KEYWORD
456 (1C8)	CHARACTER	22	SI_OPCT_K_JOBS	JOBS=(NNNNN,NNNNN,XXXXXXXX)
478 (1DE)	CHARACTER	15	SI_OPCT_K_AMLIMIT	AMLIMIT=(NNN,NNN,NNN,N)
493 (1ED)	BITSTRING	1	SI_OPCT_F_XXXXXX	KEYWORD FLAGS
	1...		SI_OPCT_F_FU	'FU=' KEYWORD PRESENT
	....		SI_OPCT_F_OPTIONS	VALUE=40'. 'OPTIONS=' KEYWORD PRESENT
	....		SI_OPCT_F_APL	VALUE=20'. 'APL=' KEYWORD PRESENT
	....		SI_OPCT_F_DDNAME	VALUE=10'. 'DDNAME=' KEYWORD PRESENT
	....		SI_OPCT_F_DD	VALUE=10'. 'DDNAME=' KEYWORD PRESENT
	....		SI_OPCT_F_MI	VALUE=08'. 'MI=' KEYWORD PRESENT
	....		SI_OPCT_F_JOB	VALUE=04'. 'JOB=' KEYWORD PRESENT
	....		SI_OPCT_F_JOBS	VALUE=02'. 'JOBS=' KEYWORD PRESENT
	....		SI_OPCT_F_AMLIMIT	VALUE=01'. 'AMLIMIT=' KEYWORD PRESENT
494 (1EE)	BITSTRING	1		WHERE FLAGS LIVE
495 (1EF)	BITSTRING	1	SI_OPCT_L_XXXXXX	LITERAL FLAGS BYTE 1
	....		SI_OPCT_L_REFRESH	VALUE=80'. 'REFRESH' LITERAL PRESENT
	....		SI_OPCT_L_QUICK	VALUE=40'. 'QUICK' LITERAL PRESENT
	....		SI_OPCT_L_ABORT	VALUE=20'. 'ABORT' LITERAL PRESENT
	....		SI_OPCT_L_ALL	VALUE=10'. 'ALL' LITERAL PRESENT
	....		SI_OPCT_L_LINKS	VALUE=08'. 'LINKS' LITERAL PRESENT
	....		SI_OPCT_L_OFFERS	VALUE=04'. 'OFFERS' LITERAL PRESENT
	....		SI_OPCT_L_AM	VALUE=02'. 'AM' LITERAL PRESENT

Offsets	Type	Length	Name	Description
	....		SI_OPCT_L_SESSIONS	VALUE=01'. 'SESSIONS' LITERAL PRESENT
496 (1F0)	BITSTRING	1		LITERAL FLAGS BYTE 2
	....		SI_OPCT_L_NULL	VALUE=80'. 'NULL' LITERAL PRESENT
	....		SI_OPCT_L_STORAGE	VALUE=40'. 'STORAGE' LITERAL PRESENT
	....		SI_OPCT_L_STATUS	VALUE=20'. 'STATUS' LITERAL PRESENT
	....		SI_OPCT_L_PROCS	VALUE=10'. 'PROCS' LITERAL PRESENT
	....		SI_OPCT_L_RESP	VALUE=08'. 'RESP' LITERAL PRESENT
	....		SI_OPCT_L_MIC	VALUE=04'. 'MIC' LITERAL PRESENT
	....		SI_OPCT_L_FSS	VALUE=02'. 'FSS' LITERAL PRESENT
	....		SI_OPCT_L_SLUSERS	VALUE=01'. 'SLUSERS' LITERAL PRESENT
497 (1F1)	BITSTRING	1		LITERAL FLAGS BYTE 3
	....		SI_OPCT_L_TRACE	VALUE=80'. 'TRACE' LITERAL PRESENT
	....		SI_OPCT_L_NOTRACE	VALUE=40'. 'NOTRACE' LITERAL PRESENT
	....		SI_OPCT_L_DUMP	VALUE=20'. 'DUMP' LITERAL PRESENT
	....		SI_OPCT_L_NODUMP	VALUE=10'. 'NODUMP' LITERAL PRESENT
	....		SI_OPCT_L_SNAP	VALUE=08'. 'SNAP' LITERAL PRESENT
	....		SI_OPCT_L_NOSNAP	VALUE=04'. 'NOSNAP' LITERAL PRESENT
	....		SI_OPCT_L_SESSION	VALUE=02'. 'SESSION' LITERAL PRESENT
	....		SI_OPCT_L_CASE	VALUE=01'. 'CASE' LITERAL PRESENT
498 (1F2)	BITSTRING	1		LITERAL FLAGS BYTE 4
	....		SI_OPCT_L_ECB	VALUE=80'. 'ECB' LITERAL PRESENT
	....		SI_OPCT_L_NODES	VALUE=40'. 'NODES' LITERAL PRESENT
	....		SI_OPCT_L_TABLES	VALUE=20'. 'TABLES' LITERAL PRESENT
	EQUATE	X'63'	SI_OPCT_SYM_SIZE	SIZE OF SYMBOL TABLE
	EQUATE	X'1F3'	SI_OPCT_SIZE	SIZE OF SI_OPCT CONTROL BLOCK
	EQUATE	X'1EB'	SI_OPCT_CLEARLEN	AMOUNT TO CLEAR FOR RESET
	EQUATE	0	SI_OPCT_SPOOL	SUBPOOL TO USE FOR SI_OPCT

SI_STAK

Common name: Alternate Operand Stack Element

Macro ID: S@COSTAK

DSECT name: SI_STAK

Created by: S@CO0030

Location: SLCN private storage

Pointed to by: SI_OPCT

Serialization: None

Function: Provides a recursive work element within the Product Operator component.

Offsets		Type	Length	Name	Description
0	(0)	A-ADDRESS	4	SI_STAK_NEXT	-> NEXT SI_STAK ON THE STACK
4	(4)	A-ADDRESS	4	SI_STAK_AOPAND	-> ALTERNATE OPERAND TABLE
8	(8)	A-ADDRESS	4	SI_STAK_AOPLIST	-> ALTERNATE OPERAND LIST
12	(C)	A-ADDRESS	4	SI_STAK_AOPENTRY	-> ALTERNATE OPERAND LIST ENTRY
16	(10)	FIXED	2	SI_STAK_LISTNUM	# LISTS LEFT TO PROCESS
18	(12)	FIXED	2	SI_STAK_ENTRYNUM	# ENTRIES LEFT IN LIST
20	(14)	A-ADDRESS	4	SI_STAK_RETURN	RETURN ADDRESS FROM PROCESSOR
		EQUATE	X'18'	SI_STAK_SIZE	LENGTH OF SI_STAK

SV_ESTW

Common name: SVC ESTAE Work Area

Macro ID: S@CCSVEW

DSECT name: SV_ESTW

Created by: S@CC0SVC

Location: Private area

Serialization: None

Function: Work area for the ESTAE recovery routine.

Offsets		Type	Length	Name	Description
0	(0)	STRUCTURE	0		ALIGN PARAMETER LIST
0	(0)	FIXED	72	SV_ESTW_REG	STANDARD 18-WORD SAVE AREA
72	(48)	FIXED	72	SV_ESTW_SAVE	STANDARD 18-WORD SAVE AREA
144	(90)	BITSTRING	8	SV_ESTW_ZPACK	DOUBLE WORD FOR PACK
152	(98)	A-ADDRESS	4	SV_ESTW_ARETRY	RETRY ROUTINE ENTRY POINT ADDRESS
156	(9C)	FIXED	64	SV_ESTW_RETREGS	REGISTERS FOR RESUME POINT
156	(9C)	FIXED	4	SV_ESTW_RETR0	R0
160	(A0)	FIXED	4	SV_ESTW_RETR1	R1
164	(A4)	FIXED	4	SV_ESTW_RETR2	R2
168	(A8)	FIXED	4	SV_ESTW_RETR3	R3
172	(AC)	FIXED	4	SV_ESTW_RETR4	R4
176	(B0)	FIXED	4	SV_ESTW_RETR5	R5
180	(B4)	FIXED	4	SV_ESTW_RETR6	R6
184	(B8)	FIXED	4	SV_ESTW_RETR7	R7
188	(BC)	FIXED	4	SV_ESTW_RETR8	R8
192	(C0)	FIXED	4	SV_ESTW_RETR9	R9
196	(C4)	FIXED	4	SV_ESTW_RETR10	R10
200	(C8)	FIXED	4	SV_ESTW_RETR11	R11
204	(CC)	FIXED	4	SV_ESTW_RETR12	R12
208	(D0)	FIXED	4	SV_ESTW_RETR13	(R13) --> STANDARD SAVE AREA
212	(D4)	FIXED	4	SV_ESTW_RETR14	(R14) --> RESUME POINT
216	(D8)	FIXED	4	SV_ESTW_RETR15	(R15) = RETURN CODE FROM RETRY
220	(DC)	A-ADDRESS	4	SV_ESTW_SSVT@	SC_SSVT ADDRESS
224	(E0)	FIXED	4	SV_ESTW_ABENDCC	ABEND COMPLETION CODE
228	(E4)	FIXED	4	SV_ESTW_SYSTEMCC	SYSTEM COMPLETION CODE
232	(E8)	FIXED	4	SV_ESTW_USERCC	USER COMPLETION CODE
236	(EC)	CHARACTER	1		

Offsets	Type	Length	Name	Description
237 (ED)	CHARACTER	3	SV_ESTW_SCC	SYSTEM COMPLETION CODE, HEX 'XXX'
240 (F0)	CHARACTER	4	SV_ESTW_UCC	USER COMPLETION CODE, DECIMAL 'DDDD'
244 (F4)	CHARACTER	8	SV_ESTW_REASON	REASON CODE
252 (FC)	CHARACTER	8	SV_ESTW_TIME	TIME HH.MM.SS
	1111 11..		SV_ESTW_HH	HH
	1111 1111		SV_ESTW_MM	MM
	EQUATE	X'102'	SV_ESTW_SS	SS
260 (104)	BITSTRING	1	SV_ESTW_FLAG	FLAG BYTE
	1...		SV_ESTW_ERECURS	1 = RECURSION
261 (105)	CHARACTER	1		Reserved
262 (106)	FIXED	0	SV_ESTW_DMSGL	HALFWORD LENGTH FIELD FOR S@@MSG
262 (106)	BITSTRING	1		FILLER - SHOULD CONTAIN ZERO
263 (107)	BITSTRING	1	SV_ESTW_DDUMPHDR	LENGTH OF THE DUMP TITLE IN BYTES
264 (108)	CHARACTER	100	SV_ESTW_DDUMPTIT	DUMP TITLE (MAXIMUM 100 CHARACTERS)
364 (16C)	STRUCTURE	0	SV_ESTW_DSDUMP	SDUMP PARAMETER LIST
364 (16C)	A-ADDRESS	1		FLAG BYTE
365 (16D)	A-ADDRESS	1		FLAG BYTE
366 (16E)	A-ADDRESS	1		FLAG BYTE
367 (16F)	A-ADDRESS	1		FLAG BYTE
368 (170)	A-ADDRESS	4		ADDRESS OF DCB
372 (174)	A-ADDRESS	4		ADDRESS OF STORAGE LIST
376 (178)	A-ADDRESS	4		ADDRESS OF USER DATA
380 (17C)	A-ADDRESS	4		ADDRESS OF ECB
384 (180)	A-ADDRESS	2		CURRENT ASID
386 (182)	A-ADDRESS	2		OTHER ASID
388 (184)	A-ADDRESS	4		ADDRESS OF ASID LIST
392 (188)	A-ADDRESS	4		ADDRESS OF SUMLIST/SUMLSTA LIST @G38
396 (18C)	A-ADDRESS	4		RESERVED @G33VPHD
400 (190)	A-ADDRESS	4		RESERVED @G33VPHD
404 (194)	STRUCTURE	0	SV_ESTW_DMSG	ALIGN
404 (194)	A-ADDRESS	4		MESSAGE NUMBER
408 (198)	CHARACTER	2		COMPONENT IDENTIFIER
410 (19A)	CHARACTER	1		TYPE OF PROCESSING REQUIRED
411 (19B)	A-ADDRESS	1		WTORLTH LENGTH OF WTORPLY
412 (19C)	A-ADDRESS	4		WTORPLY POINTER
416 (1A0)	A-ADDRESS	4		WTORECB POINTER

Offsets	Type	Length	Name	Description
420 (1A4)	A-ADDRESS	4		SSVT POINTER
424 (1A8)	A-ADDRESS	4		MESSAGE TABLE POINTER
428 (1AC)	A-ADDRESS	4		RETURN MESSAGE AREA
432 (1B0)	BITSTRING	1		ROUTING CODES
433 (1B1)	CHARACTER	1		Reserved
434 (1B2)	BITSTRING	1		DESCRIPTOR CODES
435 (1B3)	CHARACTER	1		Reserved
436 (1B4)	A-ADDRESS	1		NUMBER OF VARIABLES
437 (1B5)	CHARACTER	3		Reserved
440 (1B8)	A-ADDRESS	4		ADDR OF A MESSAGE VARIABLE
444 (1BC)	A-ADDRESS	4		ADDR OF A MESSAGE VARIABLE
	EQUATE	X'1C0'	SV_ESTW_SIZE	

Appendix B. SLCN Macros

Command Syntax Macros

S@COADEF Macro

The S@COADEF macro allows sets of alternate operands to be available on any command. Alternate operand entries may themselves point to other alternate operand lists so that complex syntaxes may be defined.

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@COADEF.
S@COADEF	
b	One or more blanks must follow S@COADEF.

desc	*desc:* character string. Enclose the string in quotes.
param group(s)	*param group(s):* One or more groups separated by commas. Each group consists of one or more RX-type addresses; if more than one address is coded, the addresses must be enclosed in parentheses and separated by commas.

The parameters are explained as follows:

desc
Specifies a textual description which can be included in error messages generated by the parser while it is processing this operand definition.

param group(s)
Specifies one or more addresses of operand definitions generated by other operand definition macros.

Each group of addresses represents one valid operand group for a command, from a choice of operand groups.

S@COCDEF Macro

The S@COCDEF macro defines each of the commands available to the command parser and its attributes. This macro also directs the parser to one or more operand entries which define the operands that are available on the command.

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@COCDEF.
S@COCDEF	
b	One or more blanks must follow S@COCDEF.

---

cmd	*cmd:* character string.
,*minlen*	*minlen:* decimal digit.
,*oplist*	*oplist:* RX-type address or list of RX-type addresses enclosed in parentheses and separated by commas.
,TSO = NO ,TSO = YES	**Default:** TSO = NO
,OPER = NO ,OPER = YES	**Default:** OPER = YES
,MODULE = *module name*	*module name:* character string.

If this is the first S@COCDEF macro in the syntax table:

,OPCTREG = *opct register*	*opct register:* decimal digits.

The parameters are explained as follows:

cmd
Specifies the name of the command being defined.

minlen
Specifies the minimum length of the command name which will be recognized by the command parser.

oplist
Specifies the address or addresses of operand definitions, generated by the operand definition macros, which define valid operands for this command.

TSO = NO
TSO = YES
Specifies whether or not the command is available for use by TSO users.

OPER = NO
OPER = YES
Specifies whether or not the command is available for use by an operator via an MCS console or by an authorized TSO user.

MODULE = *module name*
Specifies the name of the module to be entered to process the command once the command has been correctly parsed.

OPCTREG = *opct register*
Specifies a register. This register must be the same as the one used to address the SI_OPCT in the command and operand parsing routines.

It is used to generate the instructions which will be executed in the syntax table from the parsing routines to move data to the symbol table in the SI_OPCT. It must be present on the first and only on the first S@COCDEF macro in the syntax table.

S@COKDEF Macro

The S@COKDEF macro allows definitions of keyword operands to be made. The attributes of the keyword itself and the values that the keyword may take are also capable of being defined.

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@COKDEF.
S@COKDEF	
b	One or more blanks must follow S@COKDEF.

keyword	*keyword:* character string.
,minlen	*minlen:* decimal digit.
,desc	*desc:* a character string enclosed in quotes.
,type	*type:* one of the following values: **N AN ALL TEXT**
,lower bound	*lower bound:* decimal digits.
,upper bound	*upper bound:* decimal digits.
,default value	*default value:* depends on the *type* value or may be specified as one of the following values: **NONE NULL**
,TSO = NO ,TSO = YES	**Default:** TSO = NO
,OPER = NO ,OPER = YES	**Default:** OPER = YES

The parameters are explained as follows:

keyword
Specifies the name of the keyword being defined.

minlen
Specifies the minimum length of the keyword name that will be recognized by the operand parser.

desc
Gives a short description of the operand. This may be used in error messages when the operand parser encounters an error while processing this operand.

type
Specifies the type of value which may be coded with this operand; the valid types are listed below:

N This operand will accept only numeric values. Conversion to binary will be done by the parser after checking the validity of the value presented.

AN This operand will accept only alphanumeric strings of characters. Validity checking is done by the parser.

ALL This operand will accept strings including "national" characters. Validity checking is done by the parser.

TEXT This operand will accept strings containing arbitrary characters (including blanks), provided that the text is enclosed in quotes. Validity checking is done by the parser.

lower bound
Specifies the lower bound of the value allowed in the case of numeric operand values.

upper bound
Specifies the upper bound of the value allowed in the case of numeric operands. The maximum string length allowed in the case of the other operand types.

default value
Specifies the default value to be used if this operand is not specified on the command line.

This may be an actual value or one of the following literals:

NONE No default value is to be generated if this operand is not specified on the command line.

NULL A null (X'FFFF....FF') default value is to be generated if this operand is not specified on the command line.

TSO = NO
TSO = YES
Specifies whether the command is available for use by TSO users.

OPER = NO
OPER = YES
Specifies whether or not the command is available for use by an operator via an MCS console or by an authorized TSO user.

S@COLDEF Macro

The S@COLDEF macro defines literal operands that may be used on a specific command.

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@COLDEF.
S@COLDEF	
b	One or more blanks must follow S@COLDEF.

literal	*literal:* character string.
,minlen	*minlen:* decimal digit.
,TSO = NO ,TSO = YES	**Default:** TSO = NO
,OPER = NO ,OPER = YES	**Default:** OPER = YES

The parameters are explained as follows:

literal
Specifies the name of the keyword being defined.

minlen

Specifies the minimum length of the keyword name that will be recognized by the operand parser.

TSO = NO
TSO = YES
Specifies whether or not the command is available for use by TSO users.

OPER = NO
OPER = YES
Specifies whether or not the command is available for use by an operator via an MCS console or by an authorized TSO user.

S@COPDEF Macro

The S@COPDEF macro allows positional operands to be defined. The attributes of the operand itself and the values that the operand may take are also capable of being defined.

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@COPDEF.
S@COPDEF	
b	One or more blanks must follow S@COPDEF.

desc	*desc:* A character string enclosed in quotes.
,type	*type:* one of the following values: **N AN ALL TEXT**
,lower bound	*lower bound:* decimal digits.
,upper bound	*upper bound:* decimal digits.
,default value	*default value:* Depends on the *type* value or may be specified as one of the following values: **NONE NULL**
,TSO = NO ,TSO = YES	**Default:** TSO = NO
,OPER = NO ,OPER = YES	**Default:** OPER = YES

The parameters are explained as follows:

desc
Gives a short description of the operand. This may be used in error messages when the operand parser encounters an error while processing this operand.

type
Specifies the type of value which may be coded with this operand; the valid types are listed below:

N This operand will accept only numeric values. Conversion to binary will be done by the parser after checking the validity of the value presented.

AN This operand will accept only alphanumeric strings of characters. Validity checking is done by the parser.

ALL This operand will accept strings including "national" characters. Validity checking is done by the parser.

TEXT This operand will accept strings containing arbitrary characters including blanks provided the text is enclosed in quotes. Validity checking is done by the parser.

lower bound
Specifies the lower bound of the value allowed in the case of numeric operand values.

upper bound
Specifies the upper bound of the value allowed in the case of numeric operands. The maximum string length allowed in the case of the other operand types.

default value
Specifies the default value to be used if this operand is not specified on the command line.

This may be an actual value or one of the following literals:

NONE No default value is to be generated if this operand is not specified on the command line.

NULL A null (X'FFFF....FF') default value is to be generated should this operand not be specified on the command line.

TSO = NO
TSO = YES
Specifies whether or not the command is available for use by TSO users.

OPER = NO
OPER = YES
Specifies whether or not the command is available for use by an operator via an MCS console or by an authorized TSO user.

S@C1SYNX Macro

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@C1SYNX.
S@C1SYNX	
b	One or more blanks must follow S@C1SYNX.

DSECT = YES TABLE = START TABLE = END NAME = *keyword name*	*keyword name:* character string.
For NAME requests:	
,CONV = *conversion*	*conversion:* Type one of the following: `NUM HEX CHARxxxx FLAG SUBSCAN VECTOR ALIAS`
,SUBENT = *table entry*	*table entry:* symbol.
,CB = *control block name or reference*	*control block name or reference:* Type one of the following: `CIOT NIOT JIOT VIOT or PARENT`
,FIELD = (*name of field,name of length*	*name of field:* symbol.

field)	*name of length field:* symbol.
,RANGE = (*lower limit,upper limit*)	*lower limit:* symbol or decimal digits, 0-4294967295. *upper limit:* symbol or decimal digits, 0-4294967295.
,VALUE = (*value,value*)	*value:* character string.
,PRE = *pre-scan exit name*	*pre-scan exit name:* symbol.
,AFT = *post-scan exit name*	*post-scan exit name:* symbol.
,OBS = YES ,OBS = NO	**Default:** OBS = NO
,SUBSCRP = (*lower subscript,upper subscript,anchor name,total length*)	*lower subscript:* character string. A-Z,0-9 *upper subscript:* character string. A-Z,0-9 *anchor name:* symbol. *total length:* symbol or decimal digits, 1-32767.
,VCOUNT = *count of vectors*	*count of vectors:* symbol or decimal digits, 1-255. **Default:** 1

The parameters are explained as follows:

DSECT = YES
specifies that a mapping of the data areas be generated and not a look-up table.

TABLE = START
specifies that a new table is to be begun. The name field on the macro call is mandatory and is used to identify the table in a SUBENT parameter or on an assembler ENTRY statement.

TABLE = END
specifies that the current table is finished.

NAME = *keyword name*
specifies the name of the keyword being defined in the table.

CONV = *conversion*
specifies the conversion required for the parameter. The following possible alternatives available:

- NUM - numeric stored as binary
- HEX - hexadecimal stored as binary
- CHARxxxx - value stored in character form and must conform to given character set(s) x or explicit values. Possible character set(s) x are:
  - A - Alphabetic
  - N - Numeric
  - J - JCL rules (first character must be alphabetic or national)
  - S - Special national @$#
  - F - First must be alphabetic

  Note: Explicit values follow the CHARxxxx parameter as additional subparameters of CONV= eg CONV = (CHAR,%,6B). The explicit values are denoted by either a single character or two hex digits.
- FLAG - value is represented by flag bits in a flag byte
- SUBSCAN - keyword requires further subscanning using other table entries

- VECTOR - keyword requires vector subscanning using further table entries
- ALIAS - keyword is an ALIAS of another keyword

SUBENT = *table entry*
specifies the name of a further table entry to be used for scanning.

CB = *control block name or reference*
specifies the name of the control block to be updated or refers to a previous table entry.

FIELD = (*name of field,name of length field*)

- Name of field specifies the name of the control block field to be updated.
- Name of length field specifies the name of the control block field to contain the length of input in bytes for CONV = CHAR.

RANGE = (*lower limit,upper limit*)

- Lower limit specifies a minimum acceptable value except when CONV = CHAR, when it specifies a minimum length in bytes.
- Upper limit specifies a maximum acceptable value except when CONV = CHAR, when it specifies a maximum length in bytes.

Note: RANGE = and VALUE = are mutually exclusive.

VALUE = (*value,value*)
specifies a list of acceptable values unless CONV = FLAG. For CONV = FLAG specifies a series of triples as follows:

- Keyword name or null
- Value to OR
- Value to AND

Note: RANGE = and VALUE = are mutually exclusive.

PRE = *pre-scan exit name*
specifies the name of an internal routine which may perform specialized syntax analysis and control block updating.

AFT = *post-scan exit name*
specifies the name of an internal routine which performs specialized functions once the value of a parameter is known, for example obtaining storage for sub-scripted items after the maximum has been defined.

OBS = YES
OBS = NO
specifies whether a parameter is obsolete or current.

SUBSCRP = (*lower subscript,upper subscript,anchor name,total length*)

- Lower subscript specifies the first sub-script in a series, normal EBCDIC collating sequence.
- Upper subscript specifies the last sub-script in a series, normal EBCDIC collating sequence.

- Anchor name specifies the name of the control block field containing the address of the appendage to hold the subscripted item(s). This sub parameter is omitted for inline subscripted items.
- Total length specifies the length of each subscripted item to be used when obtaining storage, etc. It is particularly useful when the subscripted item is a complex data structure and the length of the symbol given in FIELD= is inappropriate.

Cross Memory Communications Macros

S@@CSERV Macro

The S@@CSERV macro handles communications between the control address space (SLCN) and the Functional Subsystem Interface (FSI) routines in the FSI part of the Functional Subsystem (FSS) address space. The macro SCHEDULEs the SRB to the FSS address space upon an ORDER-type communication, and cross memory POSTs the control address space when a SEND type communication from the FSS address space is requested.

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@@CSERV.
S@@CSERV	
b	One or more blanks must follow S@@CSERV.

TARGET=FSS TARGET=SLCN	**Default:** TARGET=SLCN
,TYPE=ORDER ,TYPE=SEND	**Default:** TYPE=SEND
,FUNC=(TERM,*term type*) ,FUNC=STATUS ,FUNC=MSG ,FUNC=INIT	*term type:* One of the following values: **QUICK NORMAL ABORT**
,PARM=*plist addr*	*plist addr:* RX-type address or register (2)-(12).
,FSID=*fss id*	*fss id:* RX-type address, register (2)-(12) or decimal digit.
,WAIT=YES ,WAIT=NO	**Default:** WAIT=YES
,ECB=*ECB addr*	*ECB addr:* RX-type address or register (2)-(12).
,DATA=*data addr*	*data addr:* RX-type address or register (2)-(12).
,LDATA=*data length*	*data length:* RX-type address or register (2)-(12).
,SSVT=*SSVT addr*	*SSVT addr:* RX-type address or register (2)-(12).

The parameters are explained as follows:

TARGET=FSS
TARGET=SLCN
Specifies the subsystem that is the target of this request. Two values are possible:

SLCN. The control address space (SLCN) is the target of this request. This is the default if TARGET is not specified.

FSS The Functional Subsystem (FSS) address space is the target of this request and is identified by the FSID parameter.

TYPE = ORDER
TYPE = SEND

Specifies the type of request. One of the following may be coded:

ORDER The type of request carried by the parameter list is a Functional Subsystem ORDER primitive. This must be accompanied by TARGET = FSS

SEND The type of request carried by the parameter list is a Functional Subsystem SEND primitive. This is the default if TYPE is not specified. This must be accompanied by TARGET = SLCN (default value).

FUNC = (TERM,*term type*)
FUNC = STATUS
FUNC = MSG
FUNC = INIT

Indicates the type of ORDER or SEND request to perform as follows:

(TERM,QUICK)	This is a TERMINATE ORDER request for a QUICK shutdown.
(TERM,NORMAL)	This is a TERMINATE ORDER request for a NORMAL shutdown.
(TERM,ABORT)	This is a TERMINATE ORDER request for an ABORT shutdown.
STATUS	This is a STATUS type ORDER request.
MSG	This is a MSG type ORDER request.
INIT	This is an INIT type ORDER request.

PARM = *plist addr*

Specifies the address of the parameter list (SC_SERV) which contains data relevant to the service requested. If this parameter is not specified, the address of the SC_SERV must have been placed previously in register 1.

FSID = *fss id*

Identifies the FSS that is the target of this request.

WAIT = YES
WAIT = NO

Specifies whether or not the the macro will be exited before the target FSS address space has notified the control address space via a cross memory post of the successful receipt of the request. YES is the default option if this parameter is not specified.

If YES is coded, control will not be returned until the notification has been received.

If NO is coded, the macro will schedule the SRB, but will not wait for notification of successful receipt. Care should be exercised when specifying this option, because the invoker is responsible for recovery processing should the SRB fail or ABEND.

ECB = *ECB addr*

Specifies the address of the ECB to be POSTed as specified by the WAIT = parameter.

DATA = *data addr*

Specifies the address of data associated with the specific request.

LDATA = *data length*
Specifies the length of data identified by the DATA= parameter.

This length is specified in a fullword or in a general register.

SSVT = *SSVT addr*
Specifies the address of the SC_SSVT control block.

Return Codes

A return code passed in register 15 indicates the success or failure of the request. This return code is the same as that returned from routine S@CF0100.

Notes

All parameters on the S@@CSERV macro are optional; however, if all of the parameters are omitted, the address of the SC_SERV must be specified in register 1, and the SC_SERV data area must be formatted with all the relevant information required to complete the request.

S@@FIREQ Macro

The S@@FIREQ macro handles the communications between the FSI component of the FSS address space and the FSS-specific routines. It also allows the FSS to CONNECT and DISCONNECT itself from the controlling address space (SLCN) via Subsystem Interface function request 53.

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@@FIREQ.
S@@FIREQ	
b	One or more blanks must follow S@@FIREQ.

REQUEST = *requested service*	*requested service:* register (2)-(12). Alternatively, one of the following values may be coded: **FSICON FSIDCON FSISEND FSIORDER**
,TARGET = SLCN ,TARGET = FSS	**Default:** TARGET = SLCN
,PARM = *plist addr*	*plist addr:* RX-type address or register (2)-(12).
,FSID = *fss id*	*fss id:* RX-type address, register (2)-(12) or decimal digit.
,FSSCB = *FSSCB addr*	*FSSCB addr:* RX-type address or register (2)-(12).
,WORK = *work area*	*work area:* RX-type address or register (2)-(12).

The parameters are explained as follows:

REQUEST = *requested service*
Specifies the requested service.

Valid services are:

FSICON CONNECT notifies the SLCN that the FSS has initialized and is available.

FSIDCON DISCONNECT notifies SLCN that the FSS is terminating and is no longer available.

FSISEND SEND enables the FSS to send a response to the SLCN.

FSIORDER ORDER enables the SLCN to send a command or order to the FSS address space.

If this parameter is not specified, one of the EQUated values for the required service must have been stored previously in the SC_FSIP_FUNC field of the SC_FSIP parameter list. For CONNECT and DISCONNECT requests, the REQUEST parameter must be specified.

TARGET = SLCN
TARGET = FSS
Specifies the target of this request. Two values are possible:

SLCN SLCN is the target of this request. This is the default if TARGET is not specified.

FSS The FSS address space identified by the FSID parameter is the target of this request.

PARM = *plist addr*
Specifies the address of the parameter list (SC_FSIP) which contains data relevant to the service requested.

If this parameter is not specified, the address of the SC_FSIP must be in register 1.

FSID = *fss id*
Identifies the FSS that is the target of this request.

If this parameter is not specified, the FSS-identifier must be in the SC_FSIP_FSID field in the SC_FSIP.

FSSCB = *FSSCB addr*
Specifies the address of the SC_FSSCB of the FSS identified by the FSID parameter.

This parameter is required only on ORDER and SEND requests. It is not required on CONNECT or DISCONNECT requests.

WORK = *work area*
Specifies the address of a 100-byte work area for use by the FSI routines.

This parameter is required only on ORDER and SEND requests. It is not required on CONNECT or DISCONNECT requests.

Notes

All parameters on the S@@FIREQ macro are optional, however if all of the parameters are omitted, the address of the SC_FSIP must be specified in register 1, and the SC_FSIP must be formatted with all the relevant information required to process the request.

S@@LOG Macro

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@@LOG.
S@@LOG	

b	One or more blanks must follow S@@LOG.

GET QUEUE	
For GET requests:	
,COUNT = *number of LOGEs*	*number of LOGEs:* decimal digits, 0-10 or register (2)-(12).
,SLNAME = *subsystem name* ,SSVT = *ssvt address*	*subsystem name:* RX-type address or register (2)-(12). *ssvt address:* RX-type address or register (2)-(12). **Default:** SSVT = 0
For QUEUE requests:	
,LEAD = (R*n*)	*n:* decimal digits, 2-12.
,SLNAME = *subsystem name* ,SSVT = *ssvt address*	*subsystem name:* RX-type address or register (2)-(12). *ssvt address:* RX-type address or register (2)-(12). **Default:** SSVT = 0

The parameters are explained as follows:

GET
QUEUE
Specifies the type of request.

GET signifies a request for Log Elements (LOGEs) to be allocated and chained. On return from a successful GET request, register 1 points to the first LOGE of the chain.

QUEUE signifies a request for the LOGEs obtained through a GET request to be queued for processing.

COUNT = *number of LOGEs*
Specifies the number of LOGEs required.

LEAD = (R*n*)
Specifies a register which contains the address of the LOGEs as returned by a GET request.

SLNAME = *subsystem name*
SSVT = *ssvt address*
Identifies which SUPERLINK subsystem is to perform the processing:

SLNAME This option specifies the address of an area containing the name of the subsystem.

SSVT This option specifies the address of the subsystem's SSVT control block. Components within the SLCN address space may specify SSVT = 0.

Return Codes

Code	*Description*
00	Successful completion
04	GET multiple request was only partially satisfied; register 0 contains a count of the LOGEs returned (at least 1; see return code 08). QUEUE message length(s) outside range, using maximum
08	Insufficient storage

12 Bad parameter; register 0 indicates the relevant parameter:

0 SLNAME or SSVT
4 COUNT
8 LEAD

16 SUPERLINK logging is inactive.

Notes

Components issuing S@@LOG do so in 31-bit addressing mode. S@@LOG returns a 31-bit address.

The message length in all returned LOGEs (after processing GET) is set to zero.

S@@SUBSY Macro

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@@SUBSY.
S@@SUBSY	
b	One or more blanks must follow S@@SUBSY.

SLNAME = *superlink name*	*superlink name:* character string. Length must be 1 to 8 characters.
,VERS = *version id*	*version id:* two EBCDIC characters. These must be in quotes if a blank is included. **Default:** VERS = ' ' (two blanks, hexadecimal '4040')

The parameters are explained as follows:

SLNAME = *superlink name*
Specifies the name of the SUPERLINK subsystem that is to provide the requested service.

This name is not necessarily the MVS subsystem name. It is the name known to the application program and provided to SUPERLINK by the installation as a parameter in the initialization options.

VERS = *version id*
Specifies a particular version of the SUPERLINK subsystem.

The specified characters are appended to the string "SL" to form the full string which identifies a subsystem as a SUPERLINK subsystem. The full string is compared with the user field in the SSCVT.

Association Manager Macros

S@@MREQ Macro

The Association Manager interface code is invoked by external modules using the S@@MREQ macro to request a function. The S@@MREQ macro format is as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@@MREQ.
S@@MREQ	
b	One or more blanks must follow S@@MREQ.

FUNC = LISTEN FUNC = PROCESS FUNC = NOTIFY FUNC = DELETE FUNC = *function*	*function:* register (2)-(12).
,SLNAME = *system name* ,SSVT = *ssvt addr*	*system name:* RX-type address or register (2)-(12). *ssvt addr:* RX-type address or register (2)-(12).
,ECB = *ecb addr*	*ecb addr:* RX-type address or register (2)-(12).
,TYPE = EP ,TYPE = ANY ,TYPE = *type*	*type:* register (2)-(12).
,ID = *identifier*	*identifier:* RX-type address or register (2)-(12).
,WKAREA = *work area*	*work area:* RX-type address or register (2)-(12).
,MF = L ,MF = (E,*ctrl addr*)	*work area:* RX-type address or register (2)-(12).
,BRANCH = YES ,BRANCH = NO	*type:* SVC or branch entry **Default:** BRANCH = NO.

The parameters are explained as follows:

FUNC = LISTEN
FUNC = PROCESS
FUNC = NOTIFY
FUNC = DELETE
FUNC = *function*

Specifies the type of function that the caller requires.

LISTEN specifies that events may be queued for the address space of the requester. It sets the user from either STATE = IDLE or STATE = STOP-NEW-DATA into STATE = LISTEN. Initially, when the STATE = IDLE and LISTEN is invoked, register 1 (upon return) will contain an identifier that the Association Manager component assigned to the address space of the requester. This ID should be quoted on all other function requests that use the ID parameter.

PROCESS requests processing of the first queued event, if any.

NOTIFY requests the Association Manager component to post an ECB when an event occurs. Notification is immediate if one or more events are already queued. The ECB must be reinitialized and the NOTIFY function reinvoked in order to be notified again.

DELETE deletes the LISTEN request for one of the following:

- Everything but termination requests (TYPE = EP; the state is set from STATE = LISTEN to STATE = STOP-NEW-DATA)
- Everything (TYPE = ANY; the state is set from STATE = LISTEN to STATE = IDLE)

FUNC = *function* may be used to specify the function from a code within a register. This code must be contained in the low-order byte of the register. The following codes are valid:

- X'01' - LISTEN
- X'02' - PROCESS
- X'04' - NOTIFY
- X'08' - DELETE

SLNAME = *system name*
SSVT = *ssvt addr*

Identifies the SUPERLINK system that is to process the request.

SLNAME specifies the address of an 8-character area in storage containing the SUPERLINK name. In-line code generated by the macro and stored in the parameter list resolves this into an SSVT address.

SSVT specifies a fullword containing the address of the SUPERLINK SSVT address.

ECB = *ecb addr*

Specifies the address of the full-word event control block to be posted when an event pertaining to association management occurs. This ECB must be initialized before the NOTIFY function is invoked. This parameter is only required when FUNC = NOTIFY is requested.

TYPE = EP
TYPE = ANY
TYPE = *type*

Specifies the type of DELETE being requested. This parameter is required only when FUNC = DELETE is coded.

EP specifies the end-point given event.

ANY specifies any event.

TYPE = *type* may be used to specify the type from a code within a register. This event type must be contained in the low-order byte of the register. The following types are valid:

- X'01' - Connection end point given
- X'02' - Any

ID = *identifier*

Specifies the address of a full-word containing the identifier associated with the address space of the requester. This identifier is assigned by the Association Manager and is used as an index in a table to locate the correct entry. The JOBNAME and JOB ID of the requester are checked against those contained in the entry for validity.

WKAREA = *work area*

Specifies the address of a 2-word work area to be used when a PROCESS request is made. The low-order byte of the first word contains an event code. The previous byte may contain a subcode as follows:

Hex Code	*Description*
00	No event was queued.
01	Connection end point was given; the second word contains the AID.

02 Termination requested; the subcode specifies the type of termination:

- X'00' - 'Graceful'
- X'01' - Quick
- X'02' - Abort

BRANCH = YES
BRANCH = NO
Specifies that an SVC entry is to be performed (NO) or that a branch entry is to be performed (YES) to the interface code.

Return Codes

The following is a list of return codes generated by the Association Manager Interface when it is called by the S@@MREQ macro. The return code is returned in register 15; the qualifier is returned in register 0:

Code	*Description*
00	Successful completion
04	Missing parameter

Qualifier:

Code	*Meaning*
04	ECB address required for FUNC = NOTIFY
08	TYPE required for FUNC = DELETE
0C	ID required
10	WKAREA required for FUNC = PROCESS
14	SLNAME/SSVT required

08 Invalid parameter

Qualifier:

04	Invalid code specified on FUNC
08	Invalid code specified on TYPE
0C	Invalid identifier specified on ID
10	Invalid SSVT address
14	Invalid Association Manager Vector Table address in SSVT

0C Function requested has been rejected

Qualifier:

00	STATE is not recognized
04	STATE = IDLE
08	STATE = STOP-NEW-DATA
0C	Association Manager component is terminating
10	STATE = LISTEN
14	STATE = LISTEN-PENDING

10 Association Manager failed or is inactive

Qualifier:

Code	*Meaning*
00	Null address found when trying to link to Association Manager Interface

	04	Association Manager STATE = INACTIVE
	08	Association Manager GWA address in Association Manager vector table is null
	0C	Resp Dir header address in GWA is null
	10	Resp Dir pointer base address in header is null
	14	Resp Dir entries queue cell address is null
	18	Responder not initiated by Association Manager
14	Insufficient space available for Association Manager	

Qualifier:

Code	*Meaning*
00	All entries in responder directory in use
04	No more elements for queue management cells
08	No more elements for RED entries

S@C@QADD Macro

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@C@QADD.
S@C@QADD	
b	One or more blanks must follow S@C@QADD.
CPID = *cpid*	*cpid:* RX-type address or register (2)-(12).
,DATA = *data*	*data:* RX-type address or register (2)-(12).
,QUEUE = *queue*	*queue:* RX-type address or register (2)-(12).

The parameters are explained as follows:

CPID = *cpid*
Specifies the address of a 4-byte area containing the cell pool identifier of the queue element cell pool. A queue element will be obtained from this cell pool to hold the data in the request queue.

DATA = *data*
Specified the address of the area that contains the queue element that is to be added to the queue. Its layout is defined by the S@C@QUE macro.

QUEUE = *queue*
Specifies the address of the request queue anchor. The data specified is inserted into a new cell pool element. The cell pool element is then chained onto the request queue.

S@C@QREM Macro

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@C@QREM.
S@C@QREM	

b	One or more blanks must follow S@C@QREM.

CPID = *cpid*	*cpid:* RX-type address or register (2)-(12).
,TARGET = *target*	*target:* RX-type address or register (2)-(12).
,QUEUE = *queue*	*queue:* RX-type address or register (2)-(12).

The parameters are explained as follows:

CPID = *cpid*
Specifies the address of a 4-byte area containing the cell pool identifier of the queue element cell pool. The queue element is returned to this cell pool once the data has been inserted into the target area.

TARGET = *target*
Specifies the address of the the area that is to contain the element removed off the queue. The layout of queue elements is defined by the S@C@QUE macro.

QUEUE = *queue*
Specifies the address of the work queue anchor.

S@C@QSWI Macro

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@C@QSWI.
S@C@QSWI	
b	One or more blanks must follow S@C@QSWI.

RQUEUE = *request queue*	*request queue:* RX-type address or register (2)-(12).
,WQUEUE = *work queue*	*work queue:* RX-type address or register (2)-(12).

The parameters are explained as follows:

RQUEUE = *request queue*
Specifies the address of the request queue anchor.

WQUEUE = *work queue*
Specifies the address of the work queue anchor.

S@C@TIMR Macro

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@S@TIMR.

S@S@TIMR

b	One or more blanks must follow S@S@TIMR.
FUNC = SET FUNC = CANCEL FUNC = *function*	*function:* register (0)-(10).
,ECB = *timer ecb*	*timer ecb:* RX-type address or register (2)-(12).
,TIME = *interval*	*interval:* RX-type address or register (2)-(12).
,SLNAME = *system name* ,SSVT = *ssvt address*	*system name:* RX-type address or register (2)-(12). *ssvt address:* RX-type address or register (2)-(12).
,MF = L ,MF = (E,*ctrl addr*)	*ctrl addr:* RX-type address or register (2)-(12).

The parameters are explained as follows:

FUNC = SET
FUNC = CANCEL
FUNC = *function*
Specifies the time interval is to be set or cancelled.

If SET is requested, the specified ECB is posted when the specified TIME has expired if it is not cancelled. If a register is specified, the function code must be contained in the low-order byte of the register. The following codes are valid:

- C'S' - SET
- C'C' - CANCEL

ECB = *timer ecb*
Specifies the address of the ECB that is to be posted when the time interval expires. This parameter is only required when FUNC = SET is coded.

TIME = *interval*
Specifies the time interval in seconds that is to expire before the user's ECB is posted. This parameter is only required when FUNC = SET is coded.

SLNAME = *system name*
SSVT = *ssvt address*
Identifies the SUPERLINK system that is to process the request.

SLNAME specifies the address of an 8 character area in storage that contains the SUPERLINK name. In-line code generated by the macro and stored in the parameter list resolves this into an SSVT address.

SSVT specifies a fullword containing the address of the SUPERLINK SSVT address.

MF = L
MF = (E,*ctrl addr*)
Specifies alternative forms of the macro.

L is used to specify the list form and (E,*ctrl addr*) specifies the execute form.

Message Processor Macros

S@@MDEF Macro

When the GML-encoded messages are formatted by DCF/SCRIPT they are converted to specifications of S@@MDEF macros. All macro parameters are positional.

The S@@MDEF macro instruction is specified as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@@MDEF.
S@@MDEF	
b	One or more blanks must follow S@@MDEF.

msg number	*msg number:* decimal digits. **Range:** 1 to 9999
,severity	*severity:* Single character
,message line	*message line:* This is of the form (*text,var*), where *text* is a quoted character string and *var* is a decimal integer enclosed in parentheses. The *var* subparameter may be omitted. **Note:** The two subparameters, as well as this parameter, may be repeated.

The parameters are explained as follows:

msg number
Specifies a unique 4-digit identifier for the message.

severity
Specifies a 1-digit severity indicator associated with the message. It is recommended that the traditional values (I, W, E, and so on) be used.

message line
Specifies a message line in terms of literal text and variable portions.

Each complete parameter enclosed in parentheses represents a complete line of the message. If a message line contains variable parts, the whole line is defined by one or more fixed text portions separated by subparameters specifying the length of a variable part. The position of the variable lengths in the macro identifies the position of the variable text in the message relative to the surrounding fixed text.

If a message consists of more than one line, more than one parameter is coded.

Notes

The S@@MDEF macro expands to a message data structure; the message structure consists of an index section and a data section. The two sections are generated as two CSECTS. The name of the CSECT that contains the index section is the label used on the first S@@MDEF macro encountered in the list. If the first macro has no label, a default label of SLNDEX is used.

Restrictions

The following is a list of restrictions that apply when using the S@@MDEF macro:

- The messages formatted by DCF/SCRIPT should be contained in sequential order within the members in the "Messages" library.
- Each line of text is restricted to 60 bytes. This applies to both single and multi-line messages.
- There is a limit of 50 variables per message.

Example

The following example shows a two line message with a variable in the first line of the message:

```
S@@MDEF  1234,I,                                                    X
           ('THE VARIABLE IS ',(3),' THREE BYTES LONG'),            X
           ('THIS IS THE SECOND LINE OF THE MESSAGE')
```

S@@MSG Macro

The S@@MSG macro is available to all components within the SUPERLINK/MVS product and is used to invoke the message processing service. During initialization processing within SLCN, the message processing service routines are loaded and anchored from the Global Service Table (GST). There are four different forms of this macro: standard, list, format and execute. Each of these formats is described in detail in the following subsections.

S@@MSG (Standard Form)

The standard form of the S@@MSG macro instruction is specified as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@@MSG.
S@@MSG	
b	One or more blanks must follow S@@MSG.

message number	*message number:* register (2)-(12).
,COMPID = *component id*	*component id:* character string of length 2
,TABLE = *message table*	*message table:* RX-type address or register (2)-(12).
,SLNAME = *system name* ,SSVT = *ssvt address*	*system name:* RX-type address or register (2)-(12). *ssvt address:* RX-type address or register (2)-(12).
,TYPE = LOCATE ,TYPE = FORMAT ,TYPE = WTO ,TYPE = WTO + LOG ,TYPE = WTOR ,TYPE = WTOR + LOG ,TYPE = LOG	**Default:** TYPE = WTO + LOG
,VAR = (*variable text*)	*variable text:* RX-type address or register (2)-(12). One or more values may be coded, separated by commas.
,ROUTCDE = (*routing code*)	*routing code:* decimal digits, 1-16 or register (2)-(12). A list of integers may

,ROUTCDE = *routcde addr*	be specified, separated by commas. *routcde addr:* RX-type address.
,DESC = (*desc code*) ,DESC = *desc addr*	*descriptor code.:* decimal digits, 1-16 or register (2)-(12). A list of integers may be specified, separated by commas. *desc addr:* RX-type address.

If TYPE = WTOR or TYPE = WTOR + LOG is coded:

,WTORPLY = *reply area*	*reply area:* RX-type address or register (2)-(12).
,WTORLTH = *reply length*	*reply length:* A-type address or register (2)-(12). **Default:** L'*reply area*
,WTORECB = *reply ecb*	*reply ecb:* RX-type address or register (2)-(12).

If TYPE = FORMAT is coded:

,RETAREA = *msg area*	*msg area:* RX-type address or register (2)-(12).

If the TYPE parameter value is not FORMAT or LOCATE:

,RETAREA = *msg area*	*msg area:* RX-type address or register (2)-(12).

The parameters are explained as follows:

message number
Specifies a unique 1- to 4-digit decimal number or specifies a register that contains the message number in binary. The message number must identify a message in the message table.

COMPID = *component id*
Specifies a 2-digit identifier of the component that is using this message macro. (If TYPE = LOCATE is specified, this parameter is ignored).

TYPE = LOCATE
TYPE = FORMAT
TYPE = WTO
TYPE = WTO + LOG
TYPE = WTOR
TYPE = WTOR + LOG
TYPE = LOG
Specifies the type of processing to be carried out.

LOCATE returns the address of the message entry in register 1. For the rest of the options, the message is formatted and output to the requested target.

FORMAT specifies that the user's return area is the requested target.

LOG specifies that the SUPERLINK LOG is the requested target.

WTO and WTOR specify that a WTO or WTOR is to be issued using the routing codes specified on the ROUTCDE parameter.

TABLE = *message table*
Specifies a full-word containing the address of the message table that contains the message identified by the message number parameter.

SLNAME = *system name*
SSVT = *ssvt address*
Identifies the SUPERLINK system that is to process the request.

SLNAME specifies the address of an 8-character area in storage that contains the SUPERLINK name. In-line code generated by the macro and stored in the parameter list resolves this into an SSVT address.

SSVT specifies a full-word containing the address of the SUPERLINK SSVT address.

VAR = (*variable text*)
Specifies the addresses of message variables that are to be inserted into the message.

The values coded on the parameter are positional; each variable represents the corresponding variable field within the message text. The user's variable data may be preceded by a halfword containing the length of the variable data. If the data length is zero, the parameter area in the message is padded with spaces. If the data length is not specified, the default length is the maximum permissible size of the variable defined in the message table entry. The maximum number of variables permitted is 50.

ROUTCDE = (*routing code*)
ROUTCDE = *routcde addr*
Specifies the routing code or codes to be used for the WTO or WTOR.

This parameter may be specified in one of three ways:

- A list of decimal integers (each between 1 and 16) enclosed in parentheses; the list may contain only one integer.
- An RX-type address which points to a halfword containing a routing code.
- A register containing the address of a halfword; the halfword contains the routing code.

The routing codes are as follows:

```
 1 Master console action          9 System security
 2 Master console information    10 System error/maintenance
 3 Tape pool                     11 Programmer information
 4 Direct access pool            12 Emulators
 5 Tape library                  13 Reserved for customer use
 6 Disk library                  14 Reserved for customer use
 7 Unit record pool              15 Reserved for custorer use
 8 Teleprocessing control        16 Reserved for future expansion
```

DESC = (*desc code*)
DESC = *desc addr*
Specifies the descriptor code or codes to be used for the WTO.

This parameter may be specified in one of three ways:

- A list of decimal integers (each between 1 and 16) enclosed in parentheses; the list may contain only one integer.
- An RX-type address which points to a halfword containing a descriptor code.
- A register containing the address of a halfword; the halfword contains the descriptor code.

The descriptor codes are as follows:

```
 1    System failure                 7    Application program/processor
 2    Immediate action required      8    Out-of-line message
 3    Eventual action required       9    Operator request
 4    System status                 10    Dynamic status requests
```

```
5    Immediate command response      11    Critical eventual action required
6    Job status                      12-16 Reserved for furure use
```

Note: Descriptor codes 1 through 6 and code 11 are mutually exclusive. Codes 7 through 10 can be assigned in combination with any other code.

WTORPLY = *reply area*
Specifies the address of an area into which the control program is to place the reply to the WTOR.

WTORLTH = *reply length*
Specifies the length of the area pointed to by WTORPLY. Valid values are 1 through 115.

WTORECB = *reply ecb*
Specifies the address of the ECB that the control program is to use to indicate the completion of the reply and the ID of the replying console. After the control program receives the reply, the ECB is updated as follows:

Offset	Length (in bytes)	Contents
+0	1	Completion code
+1	2	Reserved
+3	1	Console ID in hexadecimal

RETAREA = *msg area*
Specifies the address of an area in which the formatted message will be returned to the user. The area must begin with a halfword containing the maximum length of the user's return area. Upon return, the halfword contains the actual length of the message within the buffer. If the halfword contains zero, no data has been returned. This parameter is ignored if TYPE = LOCATE and is mandatory if TYPE = FORMAT. For all other types, this parameter is optional.

The macro will create a parameter list and call the message processing service to perform the necessary action.

Return Codes

The following is a list of return codes that will be generated by the message processing service when it is called by the S@@MSG macro. The return code is set in register 15 and the qualifier is set in register 0. (If a WTO/WTOR call is to be made, register 1 will contain the WTO/WTOR identification number.)

Code	*Description*
00	Successful completion
04	Partial completion

Qualifiers:

Code	*Meaning*
04	Message to LOG was truncated
08	Message to WTOR was truncated
12	Message to WTOR + LOG was truncated
16	Logging is inactive (where TYPE = WTO + LOG, the WTO was performed but the logging was not)

08 The requested processing failed; register 0 indicates reason

Qualifiers:

00	Unable to locate the specified message number
12	S@@LOG GET call failed
16	S@@LOG QUEUE call failed
20	Logging is inactive (TYPE = LOG only)

12 Bad parameter; register 0 indicates parameters in error

Qualifiers:

Code	*Meaning*
00	FUID
04	COMPID
08	VAR
12	ROUTCDE
16	WTORPLY
20	WTORLTH
24	WTORECB
28	SLNAME / SSVT
32	TABLE

16 Bad return code from WTO/WTOR; register 0 contains the WTO/WTOR return code.

S@@MSG Restrictions

The following restrictions apply when using the S@@MSG macro.

- Each line of text is restricted to 60 bytes. This applies to both single and multi-line messages.
- Each text message has a maximum of 50 variables.
- For TYPE = WTOR and TYPE = WTOR + LOG, the total text permitted is 112 bytes. Multi-line messages are compressed into a single-line WTOR. When text is being compressed and the limit is reached, the rest of the text is truncated and an appropriate return code is set. (If TYPE = WTOR + LOG and text for WTOR is truncated, the text in the log is also truncated).
- For TYPE = WTO and TYPE = WTO + LOG (also TYPE = LOG and TYPE = WTOR + LOG, if log output is going to the console), there is a limit of 10 lines of text in multi-line messages for all unauthorized programs. For authorized programs (supervisor state, protection key 0-7, or APF authorized), the limit is 255 lines of text.

S@@MSG (List Form)

The list form of the S@@MSG macro instruction is used to construct a control program parameter list.

The list form of the S@@MSG macro instruction is specified as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@@MSG.
S@@MSG	
b	One or more blanks must follow S@@MSG.

message number	*message number:* register (2)-(12).
,COMPID = *component id*	*component id:* character string of length 2
,TYPE = LOCATE ,TYPE = FORMAT ,TYPE = WTO ,TYPE = WTO + LOG ,TYPE = WTOR ,TYPE = WTOR + LOG ,TYPE = LOG	**Default:** TYPE = WTO + LOG
,VAR = (,...)	**Note:** This parameter is used to reserve space for one or more parameter values.
,ROUTCDE = (*routing code*)	*routing code:* decimal digits, 1-16. A list of integers, separated by commas, may be specified.
,DESC = (*desc code*)	*descriptor code:* decimal digits, 1-16 or register (2)-(12). A list of integers may be specified, separated by commas. *desc addr:* RX-type address.
,MF = L	

The parameters are described under the standard form of the macro instruction, with the following exception:

MF = L
Specifies the list form of the S@@MSG macro instruction.

S@@MSG (Format Form)

The format form of the S@@MSG macro instruction is used to insert fixed run-time parameters, such as SSVT or TABLE, into a control program parameter list generated by the list form of the macros.

The format form of the S@@MSG macro instruction is specified as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@@MSG.
S@@MSG	
b	One or more blanks must follow S@@MSG.

message number	*message number:* register (2)-(12).
,COMPID = *component id*	*component ID:* character string of length 2
,TABLE = *message table*	*message table:* RX-type address or register (2)-(12).
,SLNAME = *system name* ,SSVT = *ssvt address*	*system name:* RX-type address or register (2)-(12). *ssvt address:* RX-type address or register (2)-(12).
,TYPE = LOCATE ,TYPE = FORMAT ,TYPE = WTO ,TYPE = WTO + LOG ,TYPE = WTOR ,TYPE = WTOR + LOG	**Default:** TYPE = WTO + LOG

,TYPE = LOG	
,VAR = (*variable text*)	*variable text:* RX-type address or register (2)-(12). One or more values, separated by commas, may be specified.
,ROUTCDE = (*routing code*) ,ROUTCDE = *routcde addr*	*routing code:* decimal digits, 1-16 or register (2)-(12). A list of integers, separated by commas, may be specified. A list of registers may not be specified.
,DESC = (*desc code*) ,DESC = *desc addr*	*descriptor code:* decimal digits, 1-16 or register (2)-(12). A list of integers may be specified, separated by commas. A list of registers may not be specified. *desc addr:* RX-type address. *routcde addr:* RX-type address.
,WTORPLY = *reply area*	*reply area:* RX-type address or register (2)-(12).
,WTORLTH = *reply length*	*reply length:* A-type address or register (2)-(12). **Default:** L'*reply area*
,WTORECB = *reply ecb*	*reply ecb:* RX-type address or register (2)-(12).
,RETAREA = *msg area*	*msg area:* RX-type address or register (2)-(12).
,MF = (F,*ctrl addr*)	*ctrl area:* RX-type address or register (2)-(12).

The parameters are described under the standard form of the macro instruction with the following exception:

MF = (F,*ctrl addr*)
Specifies the format form of the S@@MSG macro instruction using a remote control program parameter list.

S@@MSG (Execute Form)

The execute form of the S@@MSG macro instruction uses a remote control program parameter list. The parameter list can be generated by the list form of the macro and modified by the format form of the macro. The parameter list may also be modified by the execute macro form.

The execute form of the S@@MSG macro instruction is specified as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@@MSG.
S@@MSG	
b	One or more blanks must follow S@@MSG.

message number	*message number:* register (2)-(12).
,COMPID = *component id*	*component ID:* character string of length 2
,TABLE = *message table*	*message table:* RX-type address or register (2)-(12).
,SLNAME = *system name* ,SSVT = *ssvt address*	*system name:* RX-type address or register (2)-(12). *ssvt address:* RX-type address or register (2)-(12).
,TYPE = LOCATE ,TYPE = FORMAT ,TYPE = WTO ,TYPE = WTO - LOG ,TYPE = WTOR ,TYPE = WTOR - LOG ,TYPE = LOG	**Default:** TYPE = WTO - LOG

,VAR = (*variable text*)	*variable text:* RX-type address or register (2)-(12). One or more values, separated by commas, may be specified.
,ROUTCDE = (*routing code*) ,ROUTCDE = *routcde addr*	*routing code:* decimal digits, 1-16 or register (2)-(12). A list of integers, separated by commas, may be specified. A list of registers may not be specified.
,DESC = (*desc code*) ,DESC = *desc addr*	*descriptor code:* decimal digits, 1-16 or register (2)-(12). A list of integers may be specified, separated by commas. A list of registers may not be specified. *desc addr:* RX-type address. *routcde addr:* RX-type address.
,WTORPLY = *reply area*	*reply area:* RX-type address or register (2)-(12).
,WTORLTH = *reply length*	*reply length:* A-type address or register (2)-(12). **Default:** L'*reply area*
,WTORECB = *reply ecb*	*reply ecb:* RX-type address or register (2)-(12).
,RETAREA = *msg area*	*msg area:* RX-type address or register (2)-(12).
,MF = (E,*ctrl addr*)	*ctrl area:* RX-type address or register (2)-(12).

The parameters are explained under the standard form of the macro instruction with the following exception:

MF = (E,*ctrl addr*)
Specifies the execute form of the S@@MSG macro instruction using a remote control program parameter list.

S@@MSGS Macro

The S@@MSGS macro is available to all components within the SUPERLINK/MVS product and is used to locate a specified message entry within a message table. It generates in-line code to locate the entry, using only registers (14) through (1), and it does not require a save area. Upon return, registers (2) through (13) remain unchanged.

The S@@MSGS macro instruction is specified as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@@MSGS.
S@@MSGS	
b	One or more blanks must follow S@@MSGS.
message number	*message number:* decimal digits or register (2)-(12).
,*message table*	*message table:* RX-type address or register (2)-(12).

The parameters are explained as follows:

message number
Specifies a unique 4-digit identifier of the message contained within the specified message table.

message table

Specifies a full-word containing the address of the message table that is to be searched for the message.

Return Codes

The following is a list of return codes that are returned upon exit from this macro call. The return code is set in register 15.

Code	*Description*
00	Successful completion, the address of the located message is in register 1
08	Failed to locate the message within the table
16	Invalid table address

S@@SVC Macro

This macro is used to pass the parameters required by the SVC. The first parameter is positional. The others are keyword parameters.

The SC_CIOT (S@C1CIOT mapping macro) must be addressable when the S@@SVC macro is issued because the SVC number is obtained from this control block.

The S@@SVC macro instruction is specified as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede S@@SVC.
S@@SVC	
b	One or more blanks must follow S@@SVC.

P R	
,SERV = *service*	*service:* symbol.
,SSVT = *ssvt address*	*ssvt address:* register (0)-(15). **Default:** SSVT = (15)
	Note: This parameter is only relevant if R is also coded.
,OFFSET = *plist offset*	*plist offset:* symbol or decimal digits. **Range:** 0 to 255.
	Note: This parameter is valid only if P is also coded.
,ESTAE = YES ,ESTAE = NO	**Default:** ESTAE = YES

The parameters are explained as follows:

P
R

Indicates the location of the SC_SSVT address.

Coding "R" specifies that the SC_SSVT address is contained in a register.

Coding "P" specifies that the SC_SSVT address is contained within a parameter list.

SERV = *service*
Specifies the service name. The following names are currently defined:

- AMGR
- CASE

SSVT = *ssvt address*
Specifies the register which contains the SC_SSVT address.

OFFSET = *plist offset*
Specifies the offset in the entry parameter list of the SC_SSVT pointer.

ESTAE = YES
ESTAE = NO
Specifies whether or not an ESTAE environment should be established by the SVC.

Note: The S@@SVC macro expands into in-line code that loads a one-word parameter list into register 0. Optionally, register 15 is also loaded with the address of the SC_SSVT. Register 1 is left unchanged.

Return Codes

When control returns to the program that issued the SVC, byte 0 in register 15 contains one of the following return codes:

Code	*Meaning*
0	Successful completion
4	SVC routine unable to obtain virtual storage for its work area. Byte 3 in R15 contains the return code from GETMAIN.
8	SVC routine unable to establish an ESTAE environment. Byte 3 in Register 15 contains the return code from ESTAE.
C	The SC_SSVT control block could not be located.
10	The entry point of the global service routine could not be located.
14	The ESTAE routine has successfully recovered an abnormal termination.

INDEX

Index

A

C

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

U

V

W

READER'S COMMENT FORM

SUPERLINK/MVS Logic Library Volume 2: Control Functional Unit SI-0182

Your reactions to this manual will help us provide you with better documentation. Please take a moment to check the spaces below, and use the blank space for additional comments.

1) Your experience with computers: ____ 0-1 year ____1-5 years ____5+ years
2) Your experience with Cray computer systems: ____0-1 year ____ 1-5 years ____5+ years
3) Your occupation: ____ computer programmer ____ non-computer professional
   ____ other (please specify): _____
4) How you used this manual: ____ in a class ____as a tutorial or introduction ____ as a reference guide
   ____ for troubleshooting

Using a scale from 1 (poor) to 10 (excellent), please rate this manual on the following criteria:

5) Accuracy _____
6) Completeness ____
7) Organization ____
8) Physical qualities (binding, printing) ____
9) Readability ____
10) Amount and quality of examples ____

Please use the space below, and an additional sheet if necessary, for your other comments about this manual. If you have discovered any inaccuracies or omissions, please give us the page number on which the problem occurred. We promise a quick reply to your comments and questions.

Name _____
Title _____
Company _____
Telephone _____
Today's Date _____

Address _____
City _____
State/ Country _____
Zip Code _____

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention: PUBLICATIONS
1345 Northland Drive
Mendota Heights, MN 55120

FOLD

STAPLE

READER'S COMMENT FORM

SUPERLINK/MVS Logic Library Volume 2: Control Functional Unit SI-0182

Your reactions to this manual will help us provide you with better documentation. Please take a moment to check the spaces below, and use the blank space for additional comments.

1) Your experience with computers: ____ 0-1 year ____1-5 years ____5+ years
2) Your experience with Cray computer systems: ____0-1 year ____ 1-5 years ____5+ years
3) Your occupation: ____ computer programmer ____ non-computer professional
____ other (please specify): _____
4) How you used this manual: ____ in a class ____as a tutorial or introduction ____ as a reference guide
____ for troubleshooting

Using a scale from 1 (poor) to 10 (excellent), please rate this manual on the following criteria:

5) Accuracy _____
6) Completeness ____
7) Organization ____
8) Physical qualities (binding, printing) ____
9) Readability ____
10) Amount and quality of examples ____

Please use the space below, and an additional sheet if necessary, for your other comments about this manual. If you have discovered any inaccuracies or omissions, please give us the page number on which the problem occurred. We promise a quick reply to your comments and questions.

Name _____
Title _____
Company _____
Telephone _____
Today's Date _____

Address _____
City _____
State/ Country _____
Zip Code _____

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention: PUBLICATIONS
1345 Northland Drive
Mendota Heights, MN 55120

FOLD

STAPLE