MERVA for ESA

Diagnosis Guide

Version 4 Release 1

SH12-6382-01

MERVA for ESA

IBM

Diagnosis Guide

Version 4 Release 1

SH12-6382-01

Note!

Before using this information and the product it supports, be sure to read the general information under "Appendix. Notices" on page 99.

Second Edition, May 2001

This edition applies to Version 4 Release 1 of IBM MERVA for ESA (5648-B29) and to all subsequent releases and modifications until otherwise indicated in new editions.

Changes to this edition are marked with a vertical bar.

Contents

About This Book

This book is one in a series of books relating to the IBM licensed program Message Entry and Routing with Interfaces to Various Applications for ESA Version 4 Release 1 (abbreviated to MERVA ESA in this book). It assists IBM customer engineers and your company's system programmers in the identification and correction of problems experienced while using the MERVA ESA program product.

It is a step-by-step guideline on how to diagnose and report MERVA ESA program failures; if any of the other products used are at fault, refer to the corresponding manuals.

The bibliography lists other books relevant for MERVA ESA that provide preparatory or additional information.

Prerequisites for Using This Book

When using this book you should be familiar with *MERVA for ESA Concepts and Components*, which describes the functions, services, and utilities supplied. It is aimed at readers who want to get a general idea of the message concept, queues, routing, message handling, and the network links.

System programmers using this book should be familiar with:

- MERVA ESA
- The operating system MVS/ESA™, OS/390®, or VSE/ESA™ under which MERVA ESA is used
- The data communication system IMS™ or CICS® under which MERVA ESA is used.

Note: The term *CICS* is used to refer to the CICS/ESA®, CICS TS, and CICS/VSE® systems. The term *IMS* is used to refer to IMS/ESA® systems.

Chapter 1. The Structure of MERVA ESA

MERVA ESA is a message-processing system mainly used in a financial environment. It enables you to exchange business messages and to carry out international financial transactions using different network links.

MERVA ESA provides the following network links:

- The SWIFT Link
- The Telex Link
- The MERVA Link
- The MERVA-to-MERVA Financial Message Transfer/ESA

These network communication programs run under the control of MERVA ESA. In this book, they are referred to individually by name, or collectively as the MERVA ESA network links. “Chapter 2. The Structure of the MERVA ESA Network Links” on page 25 describes the structure of each network-link component.

Figure 1 shows the structure of MERVA ESA. In the following sections information about the processing of the main components of MERVA ESA is provided. *MERVA for ESA Concepts and Components* gives additional information about other programs and the data sets used in MERVA ESA.

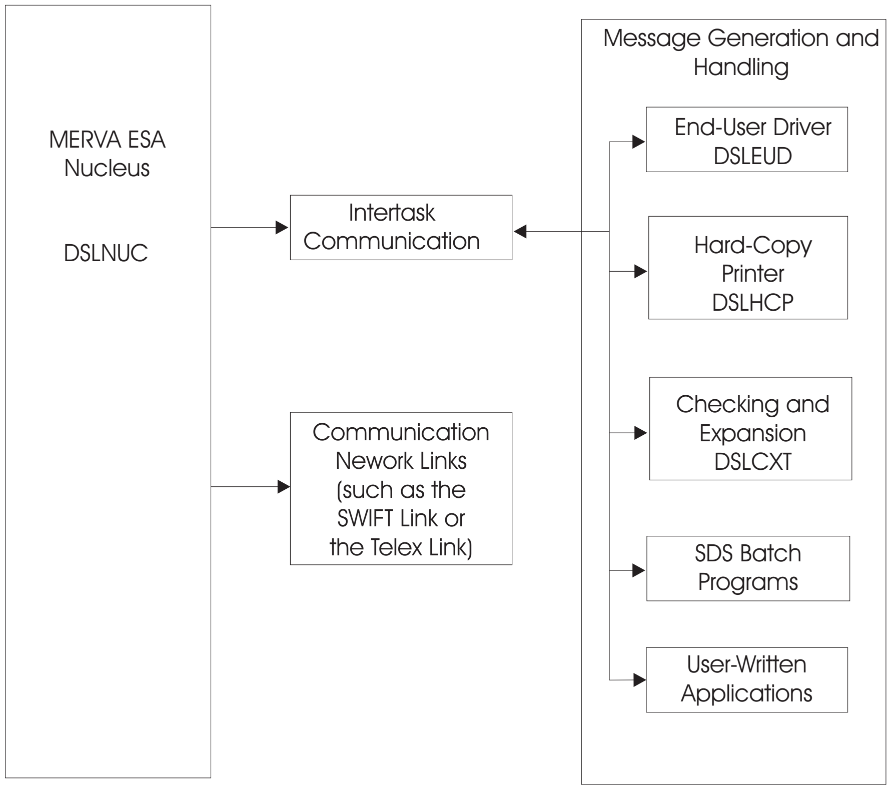

MERVA ESA Nucleus
DSLNUC
Intertask Communication
Communication Nework Links (such as the SWIFT Link or the Telex Link)
Message Generation and Handling
End-User Driver DSLEUD
Hard-Copy Printer DSLHCP
Checking and Expansion DSLCXT
SDS Batch Programs
User-Written Applications

Figure 1. MERVA ESA Components

The MERVA ESA Nucleus (DSLNUC)

Information about the components of the MERVA ESA nucleus can be found in *MERVA for ESA Concepts and Components*.

The nucleus is the central processing program of MERVA ESA:

- Under CICS, DSLNUC is a long running CICS transaction, which is started using the startup transaction program DSLCMO or DSLCAS.
- Under IMS, DSLNUC is an IMS batch message program (BMP), which is started using MVS™ job control statements.
- Under MVS, DSLNUC can be started as a native batch program using MVS job control statements.

The services provided by DSLNUC can run in separate tasks.

DSLNUC starts and stops MERVA ESA according to the commands you enter, and it controls:

- The MERVA ESA command server (DSLNCS), which processes the MERVA ESA operator commands.
- The MERVA ESA task server (DSLNTS), which provides for communication between the nucleus and the MERVA ESA applications.
- The communication link programs to external networks like the SWIFT network or the public telex network.

DSLNUC is the main program of MERVA ESA with the following major functions:

- Initializing MERVA ESA
- Processing of MERVA ESA
- Terminating MERVA ESA, normally and abnormally.

The parallel processing uses the Nucleus Server Shell programs (**DSLNSHEL** and **DSLNSHEC**) as the interface between the nucleus and the services performed by the nucleus servers running as separate tasks.

The Request Queue Handler (**DSLNRQH**) is the common component to queue and schedule service requests created by the nucleus or a nucleus server. Each time a service is requested, which is not provided by the nucleus or a nucleus server, a service request must be created for a nucleus server which provides the requested service.

The following chapters contain diagnostic information that describes the program flow of the main components of the MERVA ESA nucleus.

Initialization of DSLNUC

The initialization of DSLNUC consists of the following steps:

- Load modules needed by DSLNUC and by the programs link-edited to DSLNUC
- Allocate storage
- Load the Nucleus Server Table (DSLNSVT) and build a dynamic nucleus server table in storage
- If request queue services are available:

- Invoke the request queue handler with the *INIT* function to allocate a Server Control Block (DSLNSCB) for the nucleus and to establish the request queue handler environment.
  - Initialize the SCB with data for the nucleus.
  - Update the Nucleus Server Table (DSLNSVT)
- For all nucleus servers defined in the Nucleus Server Table (DSLNSVT) as separate tasks:
  - Invoke the request queue handler with the *INIT* function to allocate a Server Control Block (DSLNSCB) for the nucleus server shell to be attached.
  - Initialize the SCB with data for the nucleus server.
  - Update the Nucleus Server Table (DSLNSVT).
  - Create a separate task for a service by calling:
    - **DSLNATTA** to attach an MVS subtask
    - **DSLNATTC** to start a CICS task.
- Establish the exit for an abnormal end.
- If remote nucleus servers are defined in the nucleus server table, start the MQSeries® nucleus server (DSLNMQS).
- Initialize the Traffic Reconciliation interface if requested.
- Initialize the MERVA ESA journal and write the MERVA ESA START message to the journal.
- Initialize the MERVA ESA queue management.
- Initialize the MERVA ESA user file program DSLNUSR.
- Scan the MERVA ESA program table DSLNPTT for programs that should start automatically when MERVA ESA starts. Programs defined as running as a separate task will be controlled by the nucleus server shell program and run independently of DSLNUC.
- The last step depends on the success of the previous steps:
  - If all initialization steps are successful, DSLNUC issues the following message to the MERVA ESA operators:

    ```
    DSL000A MERVA is ready
    ```

    MERVA ESA is marked active and DSLNUC enters the processing cycle.
  - If one of the initialization steps fails, a dump is taken, and DSLNUC issues the following message to the MERVA ESA operators:

    ```
    DSL006A MERVA startup failed in program pgm, RC is rc
    ```

    Refer to *MERVA for ESA Messages and Codes* for an explanation of this message.

    Then all successful initializing steps are reset, and DSLNUC terminates.
- Under CICS, DSLCMO or DSLCAS is informed of the result of the DSLNUC initialization.
- All functions defined with START=AUTO in the function table are automatically started.

Processing of DSLNUC

In the processing cycle, DSLNUC makes its resources available to all active programs of the MERVA ESA Nucleus Program Table (DSLNPTT). See “Initialization of DSLNUC” on page 2 for which programs are contained in DSLNPTT.

Each of these programs is defined in DSLNPTT with one or more event control blocks (ECB). After successful initialization of such a program the ECB addresses are given to DSLNUC. Posting of such an ECB indicates to DSLNUC that the relevant program wants to get control to perform its task. Control is given to the programs according to their priorities defined in DSLNPTT.

A timer service is provided for the programs in DSLNPTT using the MERVA ESA timer program **DSLTIMP**.

Termination of DSLNUC

The termination of DSLNUC is caused by one of the following:

- The MERVA ESA operator command **terminat** or **cancel** (normal termination)
- The decision of one of the programs link-edited to DSLNUC after an error that makes it impossible for MERVA ESA to continue running (for example, a queue management error)
- An abnormal end of one of the programs link-edited to DSLNUC (abnormal termination)
- An abnormal end of one of the nucleus servers running as a separate task for DSLNUC (abnormal termination).

Normal Termination

The termination of DSLNUC consists of the following steps:

- Mark MERVA ESA inactive.
- Stop all active programs of DSLNPTT.
- Terminate the MERVA ESA queue management.
- Terminate the MERVA ESA user file program.
- Issue the following message to the MERVA ESA operators:
  ```
  DSL012I MERVA has been terminated
  ```
- Write the MERVA ESA STOP message to the MERVA ESA journal and terminate the journal.
- Terminate the MQSeries nucleus server if it is active.
- Terminate Traffic Reconciliation if it was started earlier.
- Remove the exit for an abnormal end.
- Terminate subtasks. Scan through all nucleus server table entries and invoke the request queue handler with the *TERM* function to free the Server Control Block (DSLNSCB) for the nucleus server shell to be terminated.
- Wait for all nucleus servers to terminate. If a nucleus server terminates, detach the subtask (IMS/CICS batch only).
- Invoke the request queue handler with the *TERM* function to remove the request queue handler environment.
- Release allocated storage.
- Delete all modules loaded during the initialization of DSLNUC.
- Return control to CICS, IMS, or MVS.

Errors during termination steps are ignored.

Abnormal Termination

Abnormal termination takes place in two cases:

- After an abnormal end of one of the programs link-edited to DSLNUC. Then CICS or MVS gives control to DSLNUC in the abnormal-end exit. The following steps are performed in this exit:

- Under MVS batch and IMS, the error information of the System Diagnostic Work Area (SDWA) of MVS is saved in the storage of DSLNUC at the label NUCSTAE in order to have it available in the dump. Under CICS, the error information can be found in the task ABEND control block (DFHTACB) in the dump.
   - Provide a storage dump of the error with the identification 013.
   - Issue one of the messages DSL094I or DSL095I to the MERVA ESA operators. Refer to *MERVA for ESA Messages and Codes* for an explanation of these messages.
- After an abnormal end of one of the subtasks of DSLNUC.
   - The abnormal-end exit of the nucleus server shell produces a storage dump of the error.
   - Issue one of the messages DSL385I or DSL386I to the MERVA ESA operators. Refer to *MERVA for ESA Messages and Codes* for an explanation of these messages.

After the abnormal termination event is handled, a normal termination is tried.

The Queued Call Interface Stub

All calls for nucleus services are diverted to a stub code area in the nucleus server table entry. When the caller and the nucleus server run in the same task, a direct call is executed. When parallel processing is used and the caller of a service runs in a different task from the nucleus server, the request queue handler interface **DSLNRQH** is used. A request is added to the queue of the nucleus server and its Request Ready (RR) ECB is posted.

If the invoker specified synchronous request processing, the queued call interface stub waits on the Request Processed (RP) ECB to be posted and passes back the results to the caller which resumes processing.

If the invoker specified asynchronous request processing, the queued call interface stub does not wait until the created request has finished. Instead, a control element is created. This control element contains the address of the added RCE returned by the request queue handler. The caller resumes processing immediately.

When parallel processing is used and the caller of a service runs in a different system from the nucleus server, the created service request is routed to the request queue of the MQSeries **DSLNMQS**.

Nucleus Server Shell

The following applies to both modules **DSLNSHEL** and **DSLNSHEC**.

Initialization of the Nucleus Server Shell Main Module

The initialization of the nucleus server shell consists of the following steps:

- Establish the nucleus server shell error recovery environment.
- Allocate the ECB pointer list for this nucleus server.
- Call **DSLTIMP** to establish the timer environment and initialize it.
- Initialize the ECB pointer list by filling it with addresses known and needed at the beginning:
  - The termination ECB
  - The DSLTIMP timer ECB
  - The Request Ready ECB.
- Signal the Nucleus server Ready event to the maintask (DSLNUC).

Processing of the Nucleus Server Shell Main Module

Processing of the nucleus server shell is event driven. The following steps are executed in a loop until termination is requested or an error occurred:

- Request DSLTIMP timer service. If the nucleus server shell runs under CICS, update ECB pointer list with the new address of the DSLTIMP ECB.
- Wait until any of the following events occurs:
  - Nucleus server termination
  - DSLTIMP timer expiration
  - Request processed
  - Service processed
  - Posted program
  - Request ready.
- Examine the event list for posted ECBs in the following order and invoke the appropriate processing:
  - If the termination ECB is posted by the maintask **DSLNUC**, leave the loop and return.
  - If the **DSLTIMP** timer expiration ECB is posted, post all started programs as listed in the nucleus program table.
  - If the Service Processed (SP) ECB is posted, the service previously invoked when processing the Request Ready event has finished. This event is signaled by the Request Post Processing module **DSLNRPP**. Invoke the Service Processed event processor **DSLNSPP**.
  - If the Request Processed (RP) ECB is posted, the service invoked during Request Ready processing has subsequently created another service request asynchronously, which has completed. The Request Post Processor **DSLNRPP** is invoked.
  - If a program defined in the nucleus program table is posted, invoke the Posted Program event processor **DSLNPPP**.
  - If the Request Ready (RR) ECB is posted, another nucleus server has added a request to the request queue of this nucleus server for being processed. The Request Ready event processor **DSLNRRP** is invoked.

Request Ready Event Processing (DSLNRRP)

The following steps are executed in a loop until no more service requests are on the request queue for this nucleus server:

- Obtain a service request from the request queue.
- Find the address of the entry in the nucleus server table pertaining to the services of this nucleus server.
- Reserve an address in the ECB pointer list for the Service Processed event.
- Reserve an address in the ECB pointer list for the Request Processed event.
- Check for the request type and invoke the appropriate subroutine:
  - If the entry in the nucleus server table for this nucleus server specifies to process a program specified in the nucleus program table and:
    - If it is a start request, invoke module **DSLNSNPT**. On return, update the ECB pointer list with the ECB addresses returned. Copy the addresses into the appropriate NPT entry.
    - If it is a stop request, invoke module **DSLNPNPT**. On return, remove addresses no longer used from the ECB pointer list.
  - If the entry in the nucleus server table for this nucleus server specifies to process a nucleus task service, invoke module **DSLNNTS**.
  - If the entry in the nucleus server table for this nucleus server specifies to process a command, invoke module **DSLNNCS**.
- Update the parameters with the condition code.

Start an Application Program Link-Edited to DSLNUC (DSLNSNPT)

The following steps are executed in sequence:

- Look up the nucleus server table if the requested program is defined there. If this is the case, invoke the program with a start request either via the low-level or high-level call interface, depending on the parameter specified in the NPT entry.
- On return, the Request Post Processor (**DSLNRPP**) is invoked.

Stop an Application Program Link-Edited to DSLNUC (DSLNPNPT)

The following steps are executed in sequence:

- Look up the nucleus server table if the requested program is defined there. If this is the case, invoke the program with a stop request either via the low-level or high-level call interface, depending on the parameter specified in the NPT entry.
- On return, the Request Post Processor (**DSLNRPP**) is invoked.

Invoke a Central Service

The following steps are executed in sequence:

- Look up the nucleus server table if the requested service is defined there. If this is the case, invoke the central service via the low-level call interface.
- If the requested service specifies the MQSeries program (DSLNMQS), the nucleus server table specifies routing to another system within the sysplex. **DSLNMQS** is invoked via the low-level call interface. The service request with the parameters and data is passed for being transferred using the interservice communication.
- On return, the Request Post Processor (**DSLNRPP**) is invoked.

Invoke a Command Execution Routine (DSLNNCS)

The following steps are executed in sequence:

- Look up the nucleus server table if the requested command execution module is defined there. If this is the case, invoke it via the low-level call interface.
- If the requested command specifies the MQSeries program (DSLNMQS), the nucleus server table specifies routing to another system within the sysplex. **DSLNMQS** is invoked via the low-level call interface. The service request with the parameters and data is passed for being transferred using the interservice communication. Set the processing mode for special processing.
- On return, the Request Post Processor (**DSLNRPP**) is invoked.

Posted Program Processing (DSLNPPP)

Each time an ECB of a nucleus server is posted, this processor is invoked to perform the following actions:

- Identify the appropriate entry in the nucleus server table with the posted program ECB.
- Find the program associated with the identified entry in the nucleus server table.
- Invoke the program with an ECB request either via the low-level or high-level call interface, depending on the parameter specified in the NPT entry.
- If the invoked service returned with an error, issue an operator message.
- Update the parameters with the condition code.
- Update the nucleus server shell ECB list with the updated list from DSLNPTT.
- If DSLNPPP runs in the nucleus server shell of the MQSeries nucleus server program (DSLNMQS), and if it acts as a responder, invoke the request post-processor (**DSLNRPP**) with a processing mode for special processing.

Request Post Processing (DSLNRPP)

This program is invoked by the nucleus server shell each time a service request returns control after being processed synchronously. In the case of the MQSeries nucleus server, this program is also invoked when completed as a posted NPT program. It is also invoked by the nucleus server shell following a Request Processed (RP) event. This is the case after a subsequently invoked service was invoked asynchronously and has completed.

The following steps are executed in sequence:

- If nucleus server shell of the MQSeries nucleus server and special processing is indicated in the RP ECB post code:
  - Address the service request via the control element (CE) and update the RCE with service return code, parm and buffer address, and invoke **DSLNMQS** to create an asynchronous request response message.
  - Clear the Request Processed (RP) ECB and its address in the nucleus server's ECB pointer list.
  - Compress the ECB pointer list.
  - Signal the Service Processed (SP) event to the nucleus server shell main module. Forward the post code which determined RP processing to the SP ECB.
  - Delete the service request asynchronously created by DSLNMQS.
  - Free the dynamic RCE created by DSLNMQS.
- Otherwise

– Notify the request queue handler that the obtained service request has finished processing.
– Signal the Service Processed (SP) event to the nucleus server shell main module. Forward the post code which determined RP processing to the SP ECB.

Service Processed Event Processing (DSLNSPP)

The following steps are executed in sequence:

- If no special processing is indicated in the SP ECB post code, find out if there is a parent request to signal for a Request Processed (RP) event. If there is one, post the associated Request Processed (RP) ECB pointed to by the parent request.
- Clear the Service Processed (SP) ECB and its address in the nucleus server's ECB pointer list.
- Compress the ECB pointer list.
- If this nucleus server has subsequently added a request that is still in the finished state, delete the request.

Termination of the Nucleus Server Shell Main Module

The termination of the nucleus server shell consists of the following steps:

- Call **DSLTIMP** to terminate the timers and to remove the timer environment.
- Remove the nucleus server shell error recovery environment.

Request Queue Handler (DSLNRQH)

The request queue handler consists of the following modules:

DSLNRQH	Is the request queue handler main module. It analyzes the parameters passed and invokes the appropriate queuing function processors. It consists also of the *INIT* and *TERM* request queue handler services.
DSLNRQHA	*ADD* queuing function processor
DSLNRQHO	*OBTAIN* queuing function processor
DSLNRQHN	*NOTIFY* queuing function processor
DSLNRQHD	*DELETE* queuing function processor
DSLNRQHQ	*QUERY* queuing function processor. It processes the: • Administration query of the request queue and the nucleus servers • Single query of a specific service request • Multiple query of relations to a specific request.
DSLNRQHP	*PURGE* queuing function processor
DSLNRQHR	*REQUEUE* queuing function processor
DSLNRQHG	*GETNEXT* queuing function processor

Queuing Functions

Queuing functions can be invoked by calling the address stored in the field COMRQHPA in the DSLCOM, passing the parameters as described below. There are two types of queuing functions:

- Queuing functions that rely on the rules of the finite state machine:
  - ADD
  - OBTAIN
  - NOTIFY
  - DELETE

  Any service that requests a subsequent service automatically invokes the ADD queuing function if the nucleus server does not provide it. The nucleus server which invoked the ADD queuing function must also invoke the DELETE queuing function to remove the created service request from the request queue. A nucleus server which was signaled that a service request is ready on its request queue for being processed invokes the OBTAIN queuing function and then processes the services as described in this request.

  After processing of the requested service is finished, the nucleus server must invoke the NOTIFY queuing function to update the request queue and signal the Service Processed event.

  For each single service request this sequence must be followed. Any sequence violation will be notified and the requested queuing function will not be performed.
- Queuing functions that bypass the finite state machine rules:
  - PURGE - to delete a service request regardless of its state
  - GETNEXT - to obtain the next service request in the request queue
  - REQUEUE - to put an eligible service request back to the end of the chain for waiting requests.

  Additionally, there are three different QUERY functions to process data for new display commands:
  - QRYS to query a specific request
  - QRYM to query relations of a specific request
  - QRYA to query administrative information about the request queue and the nucleus servers.

Queuing Functions Governed by the Finite State Machine Rules

ADD queuing function: This function can only be invoked by the queued call subroutine when the service call interface detects that the requested service is not provided by the nucleus server which currently processes a service request. Instead this queuing function is submitted to create a service request for another nucleus server which then should provide the requested service.

The following parameters must be specified:

parm1 ADD - the verb to request the ADD queuing function. Specification of this parameter is mandatory.

parm2 Pointer to the Server Control Block (DSLNSCB) of the nucleus server which should process the service request to be added. Specification of this parameter is mandatory.

parm3 Address of the user data area. The user data area contains the name of the nucleus server which has to obtain the service request, and the service call register values. This area is used by the nucleus server shell of the obtaining nucleus server as calling parameters when invoking the requested service. This is a mandatory parameter.

parm4 Pointer to the Request Processed (RP) ECB. Specification of this parameter is optional. However, it must be specified if an ECB other than the one in the RCE should be used.

parm5 Pointer to the Request Control Element (DSLNRCE) in the request queue which is currently processed by the nucleus server that adds the new service request. Specification of this parameter is optional. If specified, it allows to determine the request creation sequence. Otherwise, the display related request command (DRR) will contain no data.

parm6 Address returned by the request queue handler for the Request Control Element (DSLNRCE added in the request queue. It is needed to delete and query the added service request.

The processing is as follows:

- Access the SCB whose address is passed.
- Access the nucleus servers request queue and Request Queue Vector Table (DSLNRQVT), which are anchored in the SCB.
- Select first element from element chain in the common free element pool.
- Remove selected element from the free element chain and rechain.
- Insert selected element into the chain of waiting service requests.
- Build the request relationships and set the request processing type.
- Create Request Control Element (DSLNRCE by filling it with the request data, control information, and the time when the request was added.
- Signal the Request Ready event to the nucleus server which should obtain and process this request.
- Return address of RCE.

OBTAIN queuing function: This function is invoked by the nucleus or nucleus server after it receives a signal from the request queue handler that a service request is on its request queue.

The following parameters must be specified:

parm1 OBTAIN - verb to request the OBTAIN queuing function. Specification of this parameter is mandatory.

parm2 Pointer to the SCB of the nucleus server which received the Request Ready signal. Specification of this parameter is mandatory.

parm3 Pointer to a specific RCE. A nucleus server may have the need to obtain a specific request. Specification of this parameter is optional.

parm4 Address returned by the request queue handler for the Request Control Element (DSLNRCE obtained from the request queue. It is needed to notify the request queue when the service request has finished processing.

The processing is as follows:

- Access the SCB whose address is passed.
- Access the nucleus servers request queue and RQVT which are anchored in the SCB.
- Select the most eligible request in the request queue by scanning all chains of waiting requests according to their priority, starting from the highest (9) to the lowest (0) priority.
- Remove selected element from the chain of waiting requests and rechain it.
- Insert selected element into the chain of active requests.

- Update selected RCE with control information and the time it was obtained. Return address of the selected request.

NOTIFY queuing function: This function must be invoked by the issuer of the OBTAIN queuing function after the service request has completely finished processing. This includes the wait for subsequently added requests.

The following parameters must be specified:

parm1 NOTIFY - verb to request the NOTIFY queuing function. Specification of this parameter is mandatory.

parm2 Pointer to the RCB which was obtained and processed by the nucleus server. Specification of this parameter is mandatory.

The processing is as follows:
- Examine the RCE whose address is passed and extract the SCB address of the nucleus server owning the request queue.
- Access the nucleus servers request queue and RQVT which are anchored in the SCB.
- Select the element with the RCE address passed.
- Remove selected element from the chain of active requests and rechain it.
- Insert selected element into the chain of finished requests.
- Update selected RCE with control information and the time it was finished.

DELETE queuing function: This function must be invoked by the issuer of the ADD queuing function after the Service Processed event is signaled. This includes the wait for subsequently added requests.

The following parameters must be specified:

parm1 DELETE - verb to request the DELETE queuing function. Specification of this parameter is mandatory.

parm2 Pointer to the RCB to be deleted. Specification of this parameter is mandatory.

The processing is as follows:
- Examine the RCE whose address is passed and extract the SCB address of the nucleus server owning the request queue.
- Access the nucleus servers request queue and RQVT which are anchored in the SCB.
- Select the element with the RCE address to be deleted.
- Remove the selected element from the chain of finished requests and rechain it.
- Insert selected element into the chain of free elements in the free element pool.
- Update the request relationships.
- Update the selected RCE with control information and the time it was deleted.

Queuing Functions Bypassing the Finite State Machine Rules

QUERY administrative information about the request queue and nucleus servers: This function can be invoked to query administrative request queue and nucleus server data. It is invoked by module DSLNDRQA which processes the appropriate display command.

The following parameters must be specified:

parm1 QRYA - verb to request the administrative QUERY function. Specification of this parameter is mandatory.

parm2 Reserved.

parm3 Address of the SCB of the nucleus server which issues the administrative query function. Specification of this parameter is mandatory.

parm4 Address of the *request area*. This storage area is provided by the caller to hold the data returned by the administrative query function. Specification of this parameter is mandatory.

The processing is as follows:

- Access the Server Maintask Block (DSLNSMB) for common static request queue and nucleus server data.
- Loop through all SCBs and access anchored RQVTs to collect actual request queue and nucleus server data.
- Store the data into the area provided by the caller.

QUERY information about a specific service request: This function can be invoked to query a specific service request. It is invoked by module DSLNDR which processes the appropriate display command.

The following parameters must be specified:

parm1 QRYS - verb to request the single request QUERY function. Specification of this parameter is mandatory.

parm2 Address of the RCE representing the service request to query. Specification of this parameter is mandatory.

parm3 Address of the SCB of the nucleus server which issues the query function for a specific request. Specification of this parameter is mandatory.

parm4 Address of the storage area provided by the caller to hold the data (the RCE) returned by the specific request query function. Specification of this parameter is mandatory.

The processing is as follows:

- Access the SCB of the calling nucleus server.
- Access the request queue.
- Access the service request directly by the address passed in parm2.
- Store the RCE into the area provided by the caller.

QUERY information about relations of a specific service request: This function can be invoked to query relations of a specific service request. It is invoked by module DSLNDRR, which processes the appropriate display command.

The following parameters must be specified:

parm1 QRYM - verb to request the multiple request QUERY function. Specification of this parameter is mandatory.

parm2 Address of the RCE representing the service request whose relationship is to be queried Specification of this parameter is mandatory.

parm3 Address of the SCB of the nucleus server which issues the query function for a specific request. Specification of this parameter is mandatory.

parm4 Address of the storage area provided by the caller to hold the data returned by the specific request query function. Specification of this parameter is mandatory.

The processing is as follows:
- Access the SCB of the calling nucleus server.
- Access the request queue.
- Access the service request directly by the address passed in parm2.
- Scan the requests parent chain until the highest request is found.
- Scan the child chain until the lowest request is found. If a request has a brother, scan this chain until the lowest request is found. For each request found, extract data from the RCE and store it in the request area provided by the caller.

PURGE queuing function: This function is a service that allows to purge a service request. In contrast to the DELETE function, the rules of the finite state machine are bypassed; the RCE is moved from the current request queue into the chain of free elements.

The following parameters must be specified:

parm1 PURGE - verb to request the PURGE queuing function. Specification of this parameter is mandatory.

parm2 Pointer to the RCE to be purged. Specification of this parameter is mandatory.

The processing is as follows:
- Examine the RCE whose address is passed and extract the SCB address of the nucleus server owning the request queue.
- Access the nucleus servers request queue and RQVT which are anchored in the SCB.
- Select the element with the RCE address to be purged.
- Depending on the request state:
  - If waiting, remove selected element from the chain of waiting requests and rechain it.
  - If active, remove selected element from the chain of active requests and rechain it.
  - If finished, remove selected element from the chain of finished requests and rechain it.
- Insert selected element into the chain of free elements in the free element pool.
- Update the request relationships.
- Update selected RCE with control information and the time it was purged.

REQUEUE queuing function: This function is a service that can be invoked in the case DSLNUC or a nucleus server receives the Request Ready signal but is not yet able to obtain this service request. Instead it requests to suspend the processing until it is ready.

The following parameters must be specified:

parm1 REQUEUE - verb to request the REQUEUE queuing function. Specification of this parameter is mandatory.

parm2 SCB pointer of the nucleus server which requests the REQUEUE function. Specification of this parameter is mandatory.

parm3 Pointer to the RCE of the service request to be requeued. Specification of this parameter is mandatory.

The processing is as follows:

- Examine the RCE whose address is passed and extract the SCB address of the nucleus server owning the request queue.
- Access the nucleus servers request queue and RQVT which are anchored in the SCB.
- Access the specified request and check if it is still waiting.
- Remove selected element from the chain of waiting requests and rechain it.
- Insert selected element into the back of the chain of waiting requests.
- Update selected RCE with control information and the time it was requeued.
- Signal the Request Ready event to the nucleus server which should obtain and process this request.

GETNEXT queuing function: This function is a service that can be invoked to obtain the service request which is the next in the request queue.

The following parameters must be specified:

parm1 GETNEXT - verb to request the GETNEXT queuing function. Specification of this parameter is mandatory.

parm2 Pointer to the SCB of the nucleus server which requests the GETNEXT function. Specification of this parameter is mandatory.

parm3 Pointer to the previous RCE of the service request. Specification of this parameter is mandatory.

parm4 Address returned by the request queue handler for the Request Control Element (DSLNRCE next in the waiting request queue. This address can be used to obtain a specific service request.

The processing is as follows:

- Examine the RCE whose address is passed and extract the SCB address of the nucleus server owning the request queue.
- Access the nucleus servers request queue and RQVT which are anchored in the SCB.
- Access the specified request and check if it is still waiting.
- Access the next request in the waiting request queue.
- Return the address of the RCE.

Additional Services

INITIALIZE the request queue handler environment for a nucleus server: During startup, the nucleus also initializes the nucleus server and request queue handler environments. The Server Maintask Block (DSLNSMB) is allocated if the nucleus server table could be loaded. The entry definitions of the NSVT are scanned and analyzed. For the main task (the nucleus) and for each main entry in the nucleus server table where a nucleus server is defined as a separate task this function invoked.

The following parameters must be specified:

parm1 INIT - verb to request the INIT service. Specification of this parameter is mandatory.

parm2 Pointer to the allocated SMB. Specification of this parameter is mandatory.

parm3 Address where the nucleus has stored the total amount of request queue elements. Specification of this parameter is mandatory.

parm4 Address where the nucleus has stored the number to be assigned to the nucleus server. Specification of this parameter is mandatory.

parm5 Address returned by the request queue handler where the nucleus servers SCB has been allocated. Specification of this parameter is mandatory to address a nucleus server via the SCB.

The processing is as follows:

- Allocate the Server Control Block (DSLNSCB) and initialize it. All SCBs are chained in the sequence of their allocation. Thus, the first SCB in chain is always the one for the nucleus, which has the number zero.
- Allocate the Request Queue Vector Table (DSLNRQVT) for this nucleus server, anchor it in the SCB, and initialize it.
- Calculate the size of the common free element pool using the number addressed in parm3, allocate the storage, and initialize it by chaining the elements and assigning contiguous request numbers in ascending order. Update the SCB.
- Return the nucleus server number.

TERMINATE the request queue handler environment for a nucleus server: During termination, the nucleus removes the nucleus server and request queue handler environments. Each time the termination service is invoked, the SCB, whose address is passed, is analyzed, all request queues of all nucleus servers are scanned to free possible allocated areas anchored in an RCE, and to free the RQVT, the request queue, and the SCB. After all environments are terminated, the nucleus can free the SMB.

The following parameters must be specified:

parm1 TERM - verb to request the TERM service. Specification of this parameter is mandatory.

parm2 Pointer to the allocated SMB. Specification of this parameter is mandatory.

parm3 Address where the nucleus has stored the total amount of request queue elements. Specification of this parameter is mandatory.

parm4 Address where the nucleus has stored the number assigned to the nucleus server. Specification of this parameter is mandatory.

The processing is as follows:

- Access the SCB whose address and number is passed.
- Access the RQVT for this nucleus server.
- Scan all chains and check RCEs if there is an address of a request area that is not yet freed. Free the area.
- Free storage of the RQVT anchored to the SCB.
- If the SCB is for the nucleus, then free the storage of the request queue.
- Remove the SCB from the chain and free its storage.

MQSeries Queue Handler Program (DSLNMQH)

The MQSeries queue handler has the purpose to allow requesting MQSeries functions without having a need to know about the various control blocks of the message queuing interface (MQI), the application program interface of MQSeries. The MQSeries queue handler provides only the interface for a function subset of MQI which is needed by MERVA ESA.

The MQSeries queue handler consists of the following modules:

DSLNMQH	The MQSeries queue handler main module. It analyzes the parameters passed and invokes the function processors for MQI requests.
DSLNMQHC	*MQCONN* function processor
DSLNMQHO	*MQOPEN* function processor
DSLNMQHQ	*MQINQ* function processor
DSLNMQHS	*MQSIGNAL* function processor
DSLNMQHG	*MQGET* function processor
DSLNMQHP	*MQPUT* function processor
DSLNMQHL	*MQCLOSE* function processor
DSLNMQHD	*MQDISC* function processor

The following functions are specific:

DSLNMQHI	*INIT* function processor
DSLNMQHT	*TERM* function processor

MQI Function Processors

MQCONN Function Processor

This function processor allows to connect to the local message queue manager (MQM).

The following parameters must be specified:

Input parameters:

parm1 NMQTCONN - the verb to request the MQCONN MQI function. Specification of this parameter is mandatory.

parm2 MQMNM - the name of the local message queue manager (MQM). Specification of this parameter is mandatory.

Output parameter:

parm3 The returned connection handle.

Processing is as follows:

- Take the MQMNM value and call MQCONN.
- Store the returned connection handle in CHNDL.
- Return completion and reason code.

INIT Function Processor

This function processor initializes a message queue. It opens the message queue and inquires its attributes if local, creates the necessary storage areas and chains them. The address of this storage area is returned as a storage handle.

The following parameters must be specified:

Input parameters:

parm1 NMQTINIT - the verb to request the INIT function. Specification of this parameter is mandatory.

parm2 CHNDL - the connection handle as returned from MQCONN. Specification of this parameter is mandatory.

parm3 RMTQMGRNAMP - address of the locally defined name of the remote MQM.

parm4 QNAMEP - address of the locally defined name of a normal message queue or of a model queue.

parm5 DYNQNAMP - address of the dynamic message queue name if the name specified in QNAMEP is a model queue.

parm6 QTYP - message queue type:

NMQTRCV	specifies the local receive queue
NMQTRTQ	specifies the local reply-to queue
NMQTSND	specifies the local send queue
NMQTRMT	specifies a remote queue

Specification of this parameter is mandatory.

parm7 WAITINTVL - response wait time interval

parm8 RTQ_STORP - address of local reply-to queue data area

parm9 RTQ_ECBP - address of ECB to be posted by MQSeries to signal message arrival.

Output parameters:

parm10
STORP - the storage handle for the initialized message queue.

parm11
MAXQL - the maximum message length of the local reply-to queue.

Processing is as follows:

- Set up MQSeries environment for a message queue:
  - Allocate storage for the object descriptor, message descriptor, message options, and inquiry values.
  - Initialize the object descriptor with the local MQM name and the name of the message queue.
  - Initialize the message descriptor with the name of the local MQM and reply-to queue names if for a send queue.
  - Initialize the message options with the specified wait interval. If an event is to be signaled by MQSeries if a message arrives on the queue, the appropriate ECB address is set.

- Save the connection handle in the object descriptor.
- If no remote queue:
  - Open the message queue for inquiry by calling DSLNMQHO.
  - Inquire the message queue attributes by calling DSLNMQHQ:
    - The maximum message length the queue can handle
    - The queue type
    - The queue shareability.

    Store the values into the inquiry parm block.
  - Close the message queue for inquiry by calling DSLNMQHL.
- Open the message queue for the next operation. The local receive queue and the local reply-to queue are opened for get, the local send/receive queue or the remote send queue are opened for put.
- Set up the output parameters.
- Return completion and reason code.

MQOPEN Function Processor

This function processor allows to open MQSeries objects such as message queues.

The following parameters must be specified:

Input parameter:

parm1 NMQTOPEN - the verb to request the MQOPEN MQI function. Specification of this parameter is mandatory. STORP - the storage handle as returned by INIT. Specification of this parameter is mandatory.

Output parameter:

parm3 The returned object handle.

Processing is as follows:

- Take the STORP value and address the object descriptor and the option descriptor.
- Retrieve the stored connection handle, object descriptor and open options, and call MQOPEN.
- Store the returned connection handle in CHNDL.
- Return completion and reason code.

MQINQ Function Processor

This function processor allows to inquire the attributes of a local message queue.

The following parameters must be specified:

Input parameters:

parm1 NMQTINQ - the verb to request the MQINQ MQI function. Specification of this parameter is mandatory.

parm2 STORP - the storage handle as returned by INIT. Specification of this parameter is mandatory.

This function returns no output parameters.

Processing is as follows:

- Take the STORP value and address the object descriptor and the option descriptor.
- Retrieve the stored connection handle, object descriptor, open options and inquiry selectors, and call MQINQ.
- Store the returned selector attributes in the MQINQ control block.
- Return completion and reason code.

MQPUT Function Processor

This function processor allows to store a complete message regardless of its size onto a queue opened for put.

The following parameters must be specified:

Input parameters:

parm1 NMQTPUT - the verb to request the MQPUT MQI function. Specification of this parameter is mandatory.

parm2 STORP - the storage handle as returned by INIT. Specification of this parameter is mandatory.

parm3 BUFLN - length of available message buffer.

parm4 BUFFP - address of available message buffer.

parm5 FORMP - address of the format indicator as received.

parm6 RMTQS - maximum message length of remote queue.

Processing is as follows:
- Initialize the message ID and correlation ID.
- Set message type and message format, and request an exception report in the message descriptor.
- Determine the total message length and store it into the message header.
- If the data buffer to send is greater than the maximum queue size, the data is sent in segments. Loop until all segments are sent:
  - Determine segment size and segment address in data buffer.
  - Determine segment sequence number and put it into the message descriptor as application identity. Put it also into the message header.
  - Put the data onto the send queue by calling MQPUT.

  Otherwise put the available data buffer onto the send queue by calling MQPUT.
- Return completion and reason code.

MQGET Function Processor

This function processor allows to retrieve a complete message regardless of its size from a queue opened for get.

The following parameters must be specified:

Input parameters:

parm1 NMQTGET - the verb to request the MQGET MQI function. Specification of this parameter is mandatory.

parm2 STORP - the storage handle as returned by INIT. Specification of this parameter is mandatory.

Input/Output parameters:

parm3 BUFLN - length of available message buffer. If too small, the new length is returned.

parm4 BUFFP - address of available message buffer. If too small, the new address is returned.

Output parameters:

parm5 DLEN - length of received data in message buffer.

parm6 RTQP - address of remote reply-to queue name as provided by the local MQM.

parm7 RTMP - address of remote MQM name which manages the remote reply-to queue.

parm7 FORMP - address of the format indicator as received.

Processing is as follows:

- Set message ID and correlation ID to accept all messages.
- Set get message options to wait for a message, no syncpoint processing, and fail if queue manager is quiescing.
- Loop until end of data:
  - Retrieve data from the queue by calling MQGET.
  - If the total message length as indicated in the message header is greater than the available get message buffer:
    - Allocate a new get message buffer in the required length.
    - Copy the already received data into the new message buffer.
  - If the total message length is greater than the maximum queue size, the data is received in segments.
- Return sender’s reply-to queue name and MQM name.
- Return received data length and message format.
- Return completion and reason code.

MQSIGNAL Function Processor

This function processor allows MQSeries to set a signal if a message arrives on a queue opened for get. However, if a message is already on the queue, it is retrieved regardless of its size.

The following parameters must be specified:

Input parameters:

parm1 NMQTSIG - the verb to request the MQSIGNAL MQI function. Specification of this parameter is mandatory.

parm2 STORP - the storage handle as returned by INIT. Specification of this parameter is mandatory.

Input/Output parameters:

parm3 BUFLN - length of available message buffer. If too small, the new length is returned.

parm4 BUFFP - address of available message buffer. If too small, the new address is returned.

Output parameters:

parm5 DLEN - length of received data in message buffer

parm6 RTQP - address of remote reply-to queue name as provided by the local MQM.

parm7 RTMP - address of remote MQM name which manages the remote reply-to queue.

parm7 FORMP - address of the format indicator as received.

Processing is as follows:
- Set message ID and correlation ID to accept all messages.
- Set get message options to set a signal, no syncpoint processing, and fail if queue manager is quiescing.
- Loop until end of data:
  – Retrieve data from the queue by calling MQGET.
  – If the total message length as indicated in the message header is greater than the available get message buffer:
    - Allocate a new get message buffer in the required length.
    - Copy the already received data into the new message buffer.
  – If the total message length is greater than the maximum queue size, the data is received in segments.
- Return sender's reply-to queue name and MQM name.
- Return received data length and message format.
- Return completion and reason code.

TERM Function Processor

This function processor terminates a message queue. It closes the message queue and frees the storage areas created during INIT.

The following parameters must be specified:

Input parameters:

parm1 NMQTTERM - the verb to request the TERM function. Specification of this parameter is mandatory.

parm2 STORP - the storage handle as returned by INIT. Specification of this parameter is mandatory.

Processing is as follows:
- Take the MQMNM value and call MQCMIT.
- Return completion and reason code.

MQCLOSE Function Processor

This function processor allows to close MQSeries objects such as message queues.

The following parameters must be specified:

Input parameters:

parm1 NMQTCLOSE - the verb to request the MQCLOSE MQI function. Specification of this parameter is mandatory.

parm2 STORP - the storage handle as returned by INIT. Specification of this parameter is mandatory.

Processing is as follows:

- Take the STORP value and address the object descriptor and the option descriptor.
- Retrieve the stored connection handle, object descriptor and close options, and call MQCLOSE.
- Return completion and reason code.

MQDISC Function Processor

This function processor allows to disconnect from the local message queue manager (MQM).

The following parameter must be specified:

Input parameter:

parm1 NMQTDISC - the verb to request the MQDISC MQI function. Specification of this parameter is mandatory. CHNDL - the connection handle as returned from MQCONN. Specification of this parameter is mandatory.

Processing is as follows:

- Take the CHNDL value and call MQDISC.
- Return completion and reason code.

MQSeries Nucleus Server Program (DSLNMQS)

The MQSeries nucleus server is invoked by the queued call stub in the case a requested service is available on another system within a sysplex.

The MQSeries nucleus server is special in several ways:

- It is invoked only if interservice communication is specified in the MERVA ESA customization parameter module DSLPRM.
- It must run as a subtask and must therefore be specified in the nucleus server table.
- It is activated by the nucleus (DSLNUC) before any other service.
- On the requesting side it performs request processing, on the responding side it performs event processing. The requesting and the responding side act as one entity. They communicate via the MQSeries queue handler using an internal protocol.
- Its nucleus server shell forces asynchronous request processing to avoid deadlock situations.

For a requesting or responding service it is completely transparent if the request transfer is conducted by the MQSeries nucleus server. However, elapsed time increases if MQSeries does not fully exploit the capabilities of a parallel sysplex.

Chapter 2. The Structure of the MERVA ESA Network Links

MERVA ESA can connect to external and internal networks using the following network links:

- The SWIFT Link can link to the SWIFT network. The connection to the SWIFT network via X.25 uses the separate product MERVA Extended Connectivity running on a 37xx controller under ACF/NCP.
- The Telex Link allows for linking to the public telex network. There are two ways to do that:
  - Telex Link via the workstation
  - Telex Link via a fault-tolerant front-end system
- The MERVA Link can connect different MERVA installations.

This chapter summarizes the structure of each network component. For more information refer to *MERVA for ESA Concepts and Components*.

The Structure of the SWIFT Link

The SWIFT Link is the component of MERVA ESA that provides the link between MERVA ESA and the SWIFT network. The SWIFT Link supports X.25 connections to the SWIFT network.

Figure 2 shows the basic structure of the SWIFT Link for the SWIFT network.

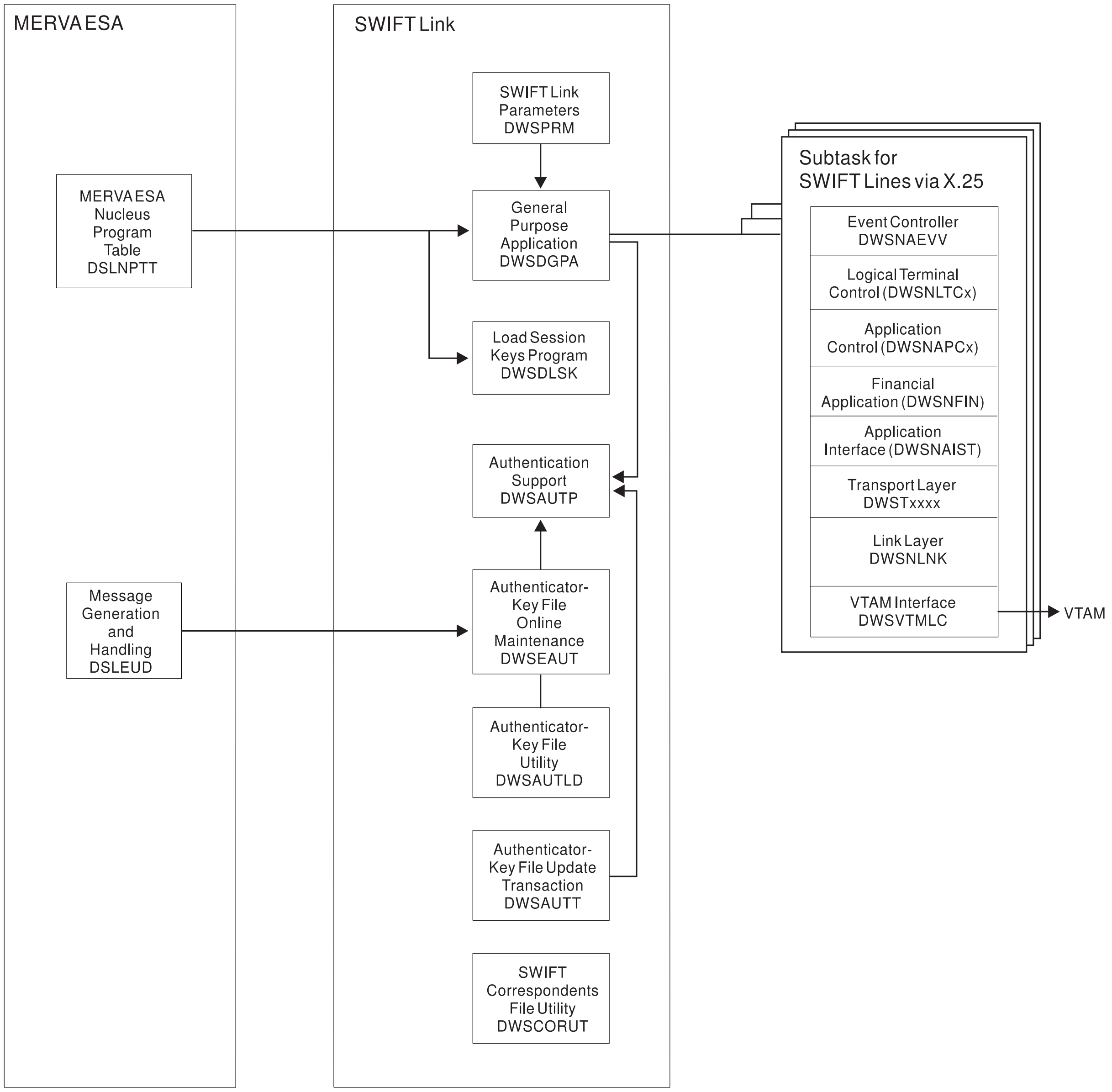

MERVA ESA
MERVA ESA Nucleus Program Table DSLNPTT
Message Generation and Handling DSLEUD
SWIFT Link
SWIFT Link Parameters DWSPRM
General Purpose Application DWSDGPA
Load Session Keys Program DWSDLSK
Authentication Support DWSAUTP
Authenticator-Key File Online Maintenance DWSEAUT
Authenticator-Key File Utility DWSAUTLD
Authenticator-Key File Update Transaction DWSAUTT
SWIFT Correspondents File Utility DWSCORUT
Subtask for SWIFT Lines via X.25
Event Controller DWSNAEVV
Logical Terminal Control (DWSNLTCx)
Application Control (DWSNAPCx)
Financial Application (DWSNFIN)
Application Interface (DWSNAIST)
Transport Layer DWSTxxxx
Link Layer DWSNLNK
VTAM Interface DWSVTMLC
VTAM

Figure 2. The SWIFT Link Structure

The main program of the SWIFT Link is the General Purpose Application program DWSDGPA. It is controlled by the MERVA ESA nucleus program DSLNUC via the nucleus program table DSLNPTT. It is possible to define the server program DWSDGPA more than once in the nucleus program table. This allows the use of several SWIFT Link servers in parallel. Each server has its own logical terminal table (DWSLTTx). Obviously, multiple servers must run as subtasks using the parallel processing feature of MERVA ESA to improve the throughput of the

SWIFT Link. The access to the authentication support is serialized through enqueue and dequeue functions provided by system services.

For each line to the SWIFT network, a subtask is attached with the main program DWSNAEVV for X.25. This program controls the layers defined by SWIFT:
- Logical terminal control (LTC, programs DWSNLTCx)
- Application control (APC, programs DWSNAPCx)
- Financial application (FIN, program DWSNFIN)
- Application interface (DWSNAIST)
- Transport layer (DWST....)
- Link layer (DWSNLNK).

DWSNLNK calls the X.25 specific program DWSNLNKV that accesses the X.25 line to the SWIFT network via the VTAM® control program DWSVTMLC, which communicates with the program MERVA Extended Connectivity running on a 37xx controller.

The Authenticator-Key File, which is used in the authentication of SWIFT messages, is maintained online by users of the function program DWSEAUT.

In addition, the Authenticator-Key File can be updated offline by the SWIFT Link utility program DWSAUTLD. If the SWIFT bilateral key exchange (BKE) is used, the Authenticator-Key File is updated by the MERVA ESA USE feature running on a workstation. MERVA Link is used to connect the workstation with MERVA ESA. The transaction program DWSAUTT receives the update from the USE workstation and updates the Authenticator-Key File. The authentication support is called by DWSDGPA to authenticate incoming and outgoing messages.

The utility program DWSCORUT is used to load the SWIFT addresses from tape to the SWIFT Correspondents File. Online maintenance of the SWIFT Correspondents File uses the facilities of the MERVA ESA General File Services.

The utility program DWSCURUT is used to load the SWIFT currency codes from tape to the SWIFT Currency Code File. Online maintenance of the SWIFT Currency Code File uses the facilities of the MERVA ESA General File Services.

The Structure of the Telex Link

The Telex Link is the component of MERVA ESA that provides the facilities to create and process Telex messages, to calculate test keys using a standard interface to a test-key calculation product.

Telex Link also provides the link between MERVA ESA and a front-end computer to access the public Telex network. There are two ways to link to the public Telex network:

- Telex Link via the workstation In this case workstation based telex functions of MERVA ESA are running on the workstation and providing the link to a telex box. MERVA Link is used to connect MERVA ESA with the workstation. MERVA Link uses an APPC connection between the host computer and the workstation.
- Telex Link via a fault-tolerant frontend The front-end computer is connected to the public telex network.

Figure 3 shows the basic structure of the Telex Link via a fault-tolerant system.

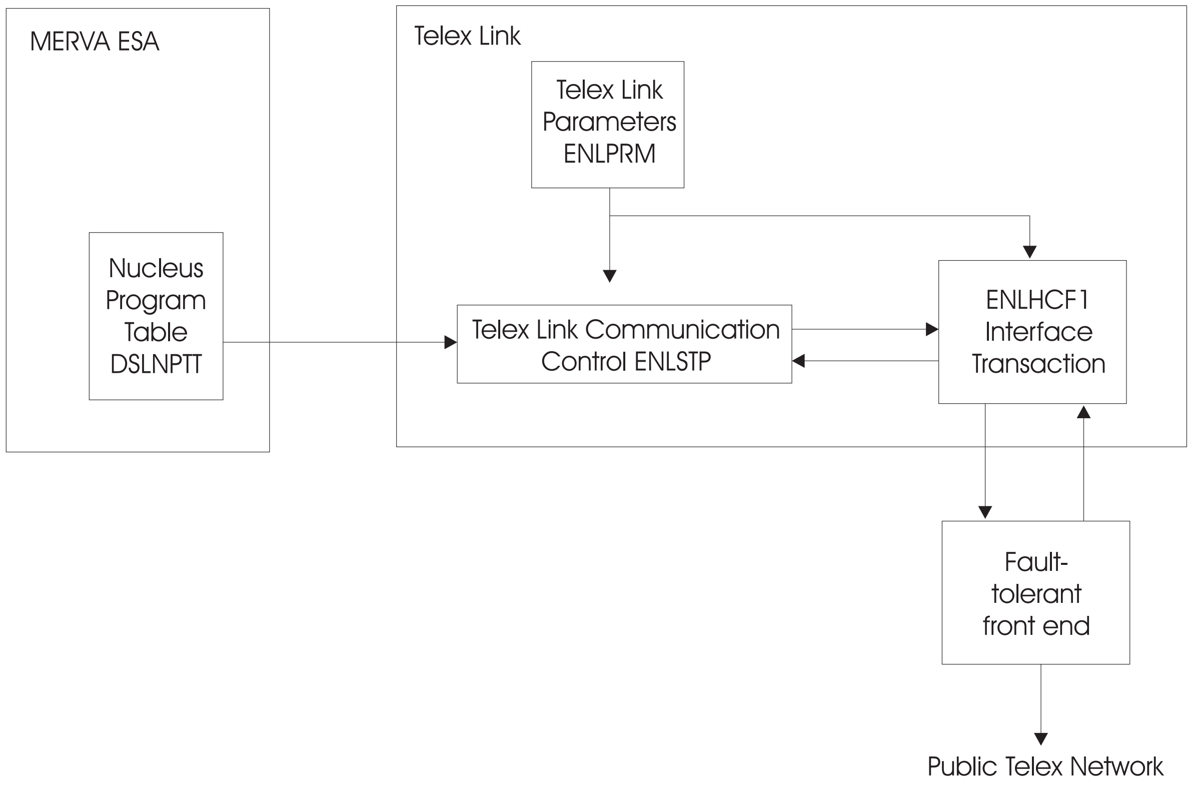

MERVA ESA
Nucleus Program Table DSLNPTT
Telex Link
Telex Link Parameters ENLPRM
Telex Link Communication Control ENLSTP
ENLHCF1 Interface Transaction
Fault-tolerant front end
Public Telex Network

Figure 3. The Telex Link via a Fault-Tolerant System Structure

The main program of the Telex Link is the station program ENLSTP. It is controlled by the MERVA ESA nucleus program DSLNUC via the nucleus program table DSLNPTT. ENLSTP communicates via MERVA ESA queues with the interface transaction ENLHCF1 when sending and receiving telex messages. ENLHCF1 communicates via a telecommunication line controlled by either CICS or IMS with the interface program in the fault tolerant front-end computer.

The Structure of the MERVA Link

MERVA Link of MERVA ESA has two subcomponents:

- MERVA Link ESA is associated with a particular MERVA ESA installation and executes in a CICS or IMS environment.
- MERVA Link USS is not associated with a particular MERVA ESA installation and executes in an OS/390 UNIX System Services (USS) environment. Detailed information about MERVA Link USS can be found in *MERVA for ESA Concepts and Components*.

The following information refers to the MERVA Link ESA subcomponent.

The MERVA Link is designed to enable MERVA ESA users to transmit and receive messages via a private network to other branches of their financial institution, or to other financial institutions that are MERVA Link users. Figure 4 shows the basic structure of the MERVA Link.
The main programs of the MERVA Link are the Application Support Programs

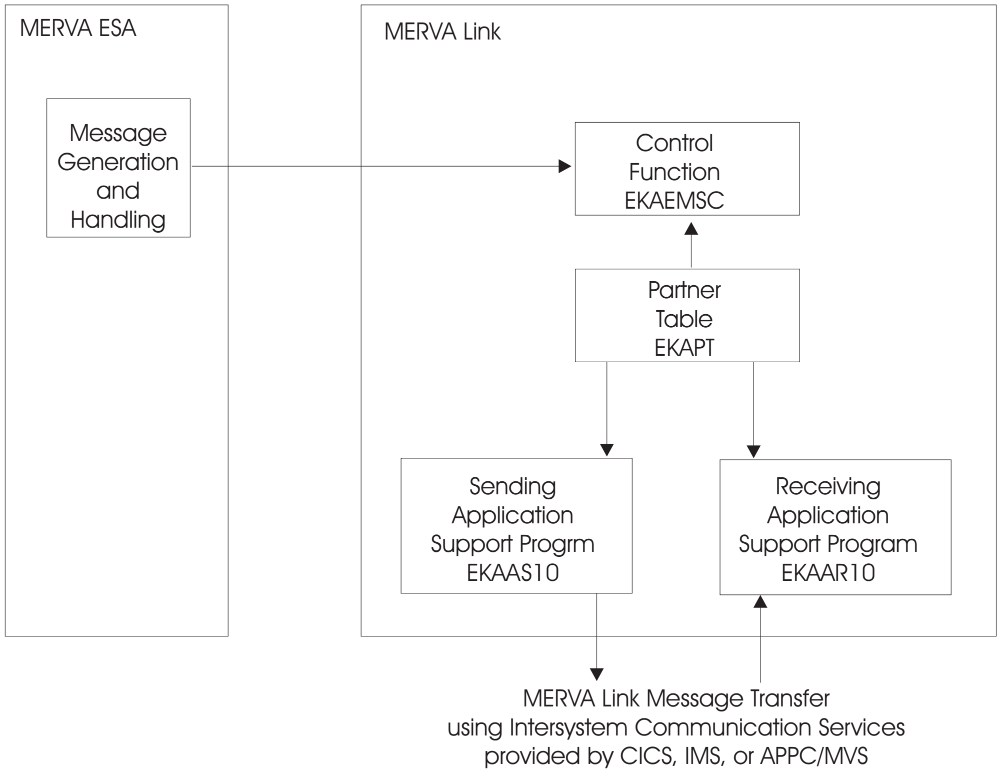

MERVA ESA
Message Generation and Handling
MERVA Link
Control Function EKAEMSC
Partner Table EKAPT
Sending Application Support Progrm EKAAS10
Receiving Application Support Program EKAAR10
MERVA Link Message Transfer using Intersystem Communication Services provided by CICS, IMS, or APPC/MVS

Figure 4. The MERVA Link Structure

(ASPs) for sending (EKAAS10) and receiving (EKAAR10). They run as CICS or IMS transaction. The Partner Table (EKAPT) defines:

- From which MERVA ESA queue to get messages for sending
- Where to send them
- Where to route acknowledged messages and received messages.

The connections to other MERVA systems use VTAM services provided by CICS (LU 6.2), APPC/MVS (LU 6.2), or IMS.

The MERVA System Control Facility (program EKAEMSC) is a MERVA ESA end-user function program that allows for supervising the MERVA Link connections using the Partner Table EKAPT. In addition, an operator may execute all other MERVA ESA operator commands and MERVA ESA test commands in this function.

The MERVA ESA system control facility allows to control the operation on remote systems which are connected to the local system via MERVA Link. Detailed information about MERVA Link can be found in *MERVA for ESA Advanced MERVA Link*.

Chapter 3. MERVA ESA Service Aids

This chapter describes service aids available for all components of MERVA ESA. It also contains information about additional service aids for individual components.

MERVA ESA Journaling

All major events are recorded in the journal, as described in *MERVA for ESA Concepts and Components*. Journaling is particularly useful when analyzing performance problems, because a time stamp is included in the header of each journal record.

MERVA ESA Internal Traces

There are the following types of internal traces:

ROUTING trace
: This trace controls the flow of messages. It can be switched on separately for each individual routing table. The trace output is written to the journal data set. Further information on this trace can be found in *MERVA for ESA Concepts and Components*, in the chapter on the routing trace entries.

QUEUE trace
: This trace controls access to the queues. It can be switched on separately for each individual queue. The trace output is written to the journal data set. Further information on this trace can be found in *MERVA for ESA Concepts and Components*, in the chapter on the queue trace.

Processing trace
: All the major MERVA ESA modules write status entries into the trace table. These can be used to debug problems within MERVA ESA, the SWIFT Link, and the Telex Link. Further information on this trace can be found in "MERVA ESA Processing Trace".

MERVA Link: Internal Module Traces and Protocol-Data-Unit (PDU) Traces
: Internal module traces can be found in dumps of MERVA Link after an error. The MERVA Link conversation or PDU traces control the data exchanged between two VTAM LU 6.2 processes on a sequential data set. The MERVA Link traces are described in *MERVA for ESA Advanced MERVA Link* in the chapter about problem determination aids.

Debugging traces
: MERVA ESA provides debugging traces for MFS and TOF services, and the processing of the nucleus server shell and the request queue handler. Further information on this trace can be found in "Debugging Traces" on page 50.

MERVA ESA Processing Trace

Product-Sensitive programming interface

Main programs store the entry point address of the MERVA ESA trace program DSLTRAP in the field COMTRAPA in the MERVA ESA communication area DSLCOM.

If you specify tracing in DSLPRM, the address of the trace table is in the field COMTRATA.

If you specify external tracing, the trace table is also written to the MERVA ESA journal. Each record in the journal starts with the MERVA ESA trace table header followed by a trace table entry with a unique identification of the program that owns the trace table.

Layout of a Trace Entry

```
offset  0        1        2         4       8                   32
        │ 1st ID │ 2nd ID │ session │ time  │ Trace Data         │
        │        │        │ number  │ stamp │  up to 24 bytes    │
```

The DSLTRA macro instruction contains definitions for the 1st ID used by the MERVA ESA programs. All other information in the trace table entry is used as needed by the programs.

The trace IDs are always shown in hexadecimal representation as this is the form they appear in a dump.

The first trace table entry is never overlayed in a wrap-around of the trace table. The first trace table entry shows the name of the main program in the data part. Trace ID X'31' indicates to DSLTRAP that the actual contents of the Trace Table must be written to the MERVA ESA journal (when the external trace is on) when the main program terminates.

The timestamp field is a 32-bit binary number representing a time interval in microseconds. The time interval is started for each new trace block. The start value, an 8-byte TOD clock value, is stored in the trace table header.

Note: The session number must not be changed or overwritten by any MERVA ESA service program.

Trace IDs

The following Trace IDs are defined for MERVA ESA:

01 – TRAIDNUC	Program DSLNUC
02 – TRAIDNCS	Program DSLNCS
03 – TRAIDNTS	Program DSLNTS
04 – TRAIDJRN	Program DSLJRNP
05 – TRAIDQMG	Program DSLQMGT
06 – TRAIDNMO	Program DSLNMOP
07 – TRAIDTIM	Program DSLTIMP
08 – TRAIDNIC	Program DSLNICT
0A – TRAIDMFS	Programs DSLMMFS, DSLMxxx
0B – TRAIDMPX	Programs DSLMPxxx
0F – TRAIDTSV	Program DSLTOFSV
14 – TRAIDEUD	Program DSLEUD
19 – TRAIDHCP	Program DSLHCP
1A – TRAIDCXT	Program DSLCXT
1B – TRAIDSDI	Program DSLSDI
1C – TRAIDSDO	Program DSLSDO
1D – TRAIDSDY	Program DSLSDY
1E – TRAIDAPI	Program DSLAPI
1F – TRAIDES1	Program DSLCES1

20 – TRAIDES2	Program DSLCES2
21 – TRAIDSE1	Program DSLCSE1
31 – TRAIDTRM	Reserved for Termination
33 – TRAIDGPA	Program DWSDGPA
34 – TRAIDLSK	Program DWSDLSK
38 – TRAIDAUT	Program DWSAUTP
39 – TRAIDEAU	Program DWSEAUT
4C – TRAIDNIN	Program DWSNINT
4D – TRAIDEVC	Programs DWSNAEVC/DWSNAEVV
4E – TRAIDLTC	Program DWSNLTCX
4F – TRAIDAPC	Program DWSNAPCX
40 – TRAIDNFI	Program DWSNFIN
41 – TRAIDAI	Program DWSNAIST
42 – TRAIDTXX	Programs DWSTxxx
43 – TRAIDNLN	Programs DWSNLNKx
44 – TRAIDVTM	Program DWSVTMLC
46 – TRAIDMLI	Program EKAAI100 - MERVA LINK INIT / TERM
47 – TRAIDMLS	Program EKAAS100 - MERVA LINK SEND MSG START
48 – TRAIDMLR	Program EKAAR100 - MERVA LINK RECV MSG START
4A – TRAIDMLX	Program EKAUXS - MERVA LINK MFS USER EXIT
50 – TRAIDKQS	Program DSLKQS
51 – TRAIDKQR	Program DSLKQR

Trace Entry Layout for MERVA ESA Programs

DSLNUC Nucleus Program (ID1=01):

```
ID2 Program                                   DATA

00  DSLNUC   Trace Header                     'DSLNUC TRACE HEADER'
00  DSLNUC   Start Record                     'DSLNUC START'
00  DSLNUC   Stop Record                      'DSLNUC STOP'
00  DSLNUC   Program Call (NPTT)              'DSLNUC CALLS  pgm-name'
00  DSLNUC   Program Ret  (NPTT)              'DSLNUC RETURN pgm-name x'
```

where x is the return code.

DSLNCS Command Server (ID1=02):

```
ID2 Program                                   DATA

00  DSLNCS   Entry                            Command Length & 20 chars
01  DSLNCS   Return                           Response Length & 20 chars
```

DSLNTS Task Server (ID1=03):

```
ID2 Program                                   DATA

00  DSLNTS   Initialization                   'DSLNTS INITIALIZATION'
01  DSLNTS   Termination                      'DSLNTS TERMINATION'
02  DSLNTS   Program found                    'DSLNTS', Pgm, INTRA/INTER, A(ICB)
03  DSLNTS   Program not found                'DSLNTS', Pgm, INTRA/INTER, A(ICB)
```

DSLJRNP Journal Program (ID1=04): The MERVA ESA Journal Program DSLJRNP does not provide trace entries in order to avoid recursive calls to the journal program.

DSLQMGT Queue Management (ID1=05):

```
ID2 Program                                 DATA

00  DSLQMG   Entry                          'DSLQMGT ENTRY', type, name
01  DSLQMG   Return                         'DSLQMGT RETURN', ret code
02  DSLRTNSC Entry                          'DSLRTNSC ENTRY'
03  DSLRTNSC Return                         'DSLRTNSC RETURN', ret code
```

DSLNMOP Console Interface (ID1=06):

```
ID2 Program                                 DATA

00  DSLNMOP  ECB posted                     'ECB POSTED'
04  DSLNMOP  Start console                  'START CONSOLE'
08  DSLNMOP  Stop console                   'STOP CONSOLE'
0C  DSLNMOP  INIT                           'INITIALIZATION CALL'
0C  DSLNMOP  TERM                           'TERMINATION CALL'
0C  DSLNMOP  PUT                            'PUT, NO JRN, NO CONS'
0C  DSLNMOP  PUTJ                           'PUT, WITH JRN, NO CONS'
0C  DSLNMOP  PUTC                           'PUT, NO JRN, WITH CONS'
0C  DSLNMOP  PUTJC                          'PUT, WITH JRN, WITH CONS'
0C  DSLNMOP  Unknown type                   type, 'IS AN UNKNOWN TYPE'
```

DSLTIMP Timer Services (ID1=07):

```
ID2 Program                                 DATA

00  DSLTIMP  INIT                           'INITIALIZATION'
00  DSLTIMP  TERM                           'TERMINATION'
00  DSLTIMP  SET                            'SET UP', name
00  DSLTIMP  SET                            'SET UP, NAME PTR IS ZERO'
00  DSLTIMP  CANCEL                         'CANCEL', name
00  DSLTIMP  CANCEL                         'CANCEL, NAME PTR IS ZERO'
00  DSLTIMP  TIMER                          'TIMER'
00  DSLTIMP  POST                           'ECB POSTED'
00  DSLTIMP  CALC                           'CALCULATE'
00  DSLTIMP  Unknown type                   type, 'IS AN UNKNOWN TYPE'
```

DSLNICT Intertask Communication (ID1=08):

```
ID2 Program                                 DATA

xx  DSLNICT  INIT                           'DSLNICT INITIALIZATION'
xx  DSLNICT  TERM                           'DSLNICT TERMINATION'
xx  DSLNICT  RESP                           'DSLNICT RESPONSE', A(ICB)
xx  DSLNICT  STATUS                         'DSLNICT STATUS'
xx  DSLNICT  ALLOC                          'DSLNICT ALLOCATE'
xx  DSLNICT  FREE                           'DSLNICT FREE', A(ICB)
xx  DSLNICT  REQ                            'DSLNICT REQUEST', A(ICB)
xx  DSLNICT  CHECK                          'DSLNICT CHECK'
xx  DSLNICT  RTV                            'DSLNICT RETRIEVE', a(ICB)
xx  DSLNICT  Unknown type                   'DSLNICT UNKNOWN TYPE'
xx  DSLNICT  Return                         'DSLNICT RETURN',ret code
```

where xx is 00 for INTRA-REGION and 01 for INTER-REGION.

Note: When DSLNICT is called by DSLTRAP to write the trace table to the MERVA ESA journal as a central service, the trace entries of DSLNICT are suppressed by DSLTRAP to avoid problems of recursion.

DSLMMFS Message Format Service (ID1=0A):

```
ID2 Program                                 DATA

00  DSLMMFS  Init MFS                       DSLMMFS REL410  sysparm
01  DSLMMFS  Term MFS                       MFS statistics from PS
02  DSLMMFS  Call MFS service               ->PS, ->TS, ->PL, Plist
03  DSLMMFS  MFS error message              24 bytes of error message
```

```
 04  DSLMMFS  Exit MFS service                  ->PS, ->TS, ->PL, Plist

 09  DSLMUxxx MFS user exit                     Module Header
 0A  DSLMLFP  Mapping Error                     Tag Only Data Mapped

 14  DSLMU054 Message Type Det.                 8 Byte message id
```

For ID2=00, sysparm can be CICSMVS, CICSVSE, or IMSMVS.

For ID2=01, the MFS statistics for termination are:

```
offset  22         24                28
        |number of | maximum size of| current size of|
        |getmains  | getmain storage| getmain storage|
```

For ID2=02 and ID2=04 the DATA field contains:

```
->PS   pointer to the MFS permanent storage
->TS   pointer to the MFS temporary storage
->PL   pointer to the MFS parameter list
Plist  first 12 bytes of the MFS parameter list
```

For ID2=04 the timestamp field contains the execution time of this MFS service call in microseconds.

DSLMPxxx MFS Screen and Print Formatting (ID1=0B):

```
 ID2 Program                                    DATA

 14  DSLMPSSR Exit from Call                    '**DSLMPSSR**'
 19  DSLMPSTP Exit from Call                    '**DSLMPSTP**'
 1E  DSLMPSSY Exit from Call                    '**DSLMPSSY**'
 23  DSLMPSSC Exit from Call                    '**DSLMPSSC**'
 28  DSLMPULD Exit from Call                    '**DSLMPULD**'
 32  DSLMPUTF Entry to Call                     '**DSLMPUTF**'
 33  DSLMPUTF Exit from Call                    '**DSLMPUTF**'
```

For ID2=14 the following working storage is traced in 3 trace entries:

```
 DCL 1 WRK,
       2 WRKID    CHAR(12),           /*IDENTIFICATION **DSLMPSSR** */
       2 *,
         3 SAVLDSA  PTR,               /* ADDRESS LDS               */
         3 SAVEDITA PTR,               /* ADDRESS EDIT BUFFER       */
         3 SAVRKEYA PTR,               /* ADDRESS REKEY BUFFER      */
         3 SAVIOA   PTR,               /* ADDRESS I/O-BUFFER        */
         3 P3270    PTR,               /* CURRENT POSITION IN       */
                                       /* IO BUFFER                 */
         3 SAVSNDA  PTR,               /* ADDRESS SEND STATUS       */
         3 SAVSESSA PTR,               /* ADDRESS SESSION CONTROL   */
         3 SAVTUCBA PTR,               /* ADDRESS TUCB              */
         3 SAVCOMA  PTR,               /* ADDRESS COMMON AREA       */
         3 SAVMFSPA PTR,               /* ADDRESS PERMANENT STORAGE */
         3 SAVTOFA  PTR,               /* ADDRESS TOF               */
       2 *,                            /* WORKING VARIABLES         */
         3 WRKRETC  BIN(8),            /* RETURN CODE               */
         3 *        BIN (8),           /* FUTURE USE                */
         3 WRKREAS  BIN(15),           /* REASON CODE               */
         3 WRKDAOFF  BIN (15),         /* DATA OFFSET: IS USED IF   */
                                       /* ONLY PART OF THE SLOT DATA*/
                                       /* WAS MOVED INTO IO BUFFER  */
         3 WRKCURLO  BIN (15),         /* OFFSET TO CURRENT LDS ITEM*/
                                       /* IS USED WHEN REENTER IS   */
                                       /* NECESSARY                 */
         3 *         BIT (8),
           4 WRKFUNC BIT (1),          /* FUNCTION TO BE PERFORMED  */
                                       /* 0=PUT 1=REENTER           */
```

```
          4 WRKALLM BIT (1),           /* ALL DATA OF THIS SLOT HAS */
                                       /* BEEN MOVED INTO IO BUFFER */
          4 BLOCKFD BIT (1),           /* INDICATES THAT A VALID    */
                                       /* LDS BLOCK WAS FOUND AND   */
                                       /* THE LOOP CAN BE TERMINATED*/
          4 *       BIT (5),           /* FUTURE USE                */
        3 WRKSTAT   BIT(8),            /* PROCESSING OPTIONS        */
          4 WRKLDSC BIT(1),            /* LDS COMPLETE INDICATOR    */
          4 WRKLDSI BIT (1),           /* INVALID LDS               */
          4 WRKIOF  BIT(1),            /* IO BUFFER FULL INDICATOR  */
          4 WRKERR  BIT(1),            /* ERROR DETECTED = '1'B     */
          4 *       BIT(4);            /* FUTURE USE                */
```

For ID2=19 the following working storage is traced in 4 trace entries:

```
 DCL 1 WRK,
       2 WRKID    CHAR(12),           /*IDENTIFICATION **DSLMPSTP** */
       2 *,
         3 SAVLDSA  PTR,               /* ADDRESS LDS               */
         3 SAVEDITA PTR,               /* ADDRESS EDIT BUFFER       */
         3 SAVRKEYA PTR,               /* ADDRESS REKEY BUFFER      */
         3 SAVIOA   PTR,               /* ADDRESS I/O-BUFFER        */
         3 P3270    PTR,               /* CURRENT POSITION IN       */
                                       /* IO BUFFER                 */
         3 SAVIOPTR PTR,               /* SAVE ADDRESS OF IO-BUFFER */
                                       /* WHENEVER A NEW ROW OF DATA*/
                                       /* STARTS                    */

         3 SAVSNDA  PTR,               /* ADDRESS SEND STATUS       */
         3 SAVSESSA PTR,               /* ADDRESS SESSION CONTROL   */
         3 SAVTUCBA PTR,               /* ADDRESS TUCB              */
         3 SAVCOMA  PTR,               /* ADDRESS COMMON AREA       */
         3 SAVMFSPA PTR,               /* ADDRESS PERMANENT STORAGE */
         3 SAVTOFA  PTR,               /* ADDRESS TOF               */
       2 *,                            /* WORKING VARIABLES         */
         3 WRKROWST,                   /* SEND STATUS AT BEGINNING  */
                                       /* OF EACH ROW               */
          4 WRKNXITO BIN (15),          /* NEXT ITEM OFFSET          */
          4 WRKITDO  BIN (15),          /* DATA OFFSET INTO EDIT OR  */
                                       /* REKEY BUFFER              */
          4 WRKNXRW  BIN (15),          /* ROW NUMBER TO BE PROCESSED*/
                                       /* NEXT                      */
          4 WRKDEVO  BIN (15),          /* NEXT FREE OFFSET IN       */
                                       /* PRINTER BUFFER            */
         3 WRKRETC  BIN(8),            /* RETURN CODE               */
         3 *       BIN (8),            /* FUTURE USE                */
         3 WRKREAS  BIN(15),           /* REASON CODE               */
         3 WRKDAOFF  BIN (15),         /* DATA OFFSET: IS USED IF   */
                                       /* ONLY PART OF THE SLOT DATA*/
                                       /* WAS MOVED INTO IO BUFFER  */
         3 WRKCURLO  BIN (15),         /* OFFSET TO CURRENT LDS ITEM*/
                                       /* IS USED WHEN REENTER IS   */
                                       /* NECESSARY                 */
         3 WRKPROW   BIN (15),         /* PREVIOUSLY PROCESSED ROW  */
         3 WRKCCOL   BIN (15),         /* CURRENTLY PROCESSED COLUMN*/
         3 WRKCROW   BIN (15),         /* CURRENTLY PROCESSED ROW   */
         3 *         BIT (16),
           4 WRKFUNC BIT (1),          /* FUNCTION TO BE PERFORMED  */
                                       /* 0=PUT 1=REENTER           */
           4 WRKALLM BIT (1),          /* ALL DATA OF THIS SLOT HAS */
                                       /* BEEN MOVED INTO IO BUFFER */
           4 BLOCKFD BIT (1),          /* INDICATES THAT A VALID    */
                                       /* LDS BLOCK WAS FOUND AND   */
                                       /* THE LOOP CAN BE TERMINATED*/
           4 ROWOVFL BIT (1),          /* ITEM DATA FLOWS BEYOND A  */
                                       /* PRINTER LINE              */
          4 ONLYNL  BIT (1),          /* THE LDS IS COMPLETELY DONE*/
                                      /* ONLY 'NL' ORDERS MUST BE  */
                                      /* INSERTED TO FILL THE PAGE */
          4 FLDUSC  BIT (1),          /* FIELD HAS THE UNDERSCORE  */
                                      /* ATTRIBUTE                 */
          4 ATTINNL BIT (1),          /* THE END ATTRIBUTE FOR AN  */
                                      /* UNDERSCORE FIELD IS IN A  */
                                      /* NEW LINE                  */
          4 *       BIT (9),          /* FUTURE USE                */
        3 WRKSTAT   BIT(8),           /* PROCESSING OPTIONS        */
          4 WRKLDSC BIT(1),           /* LDS COMPLETE INDICATOR    */
          4 WRKLDSI BIT (1),          /* INVALID LDS               */
          4 WRKIOF  BIT(1),           /* IO BUFFER FULL INDICATOR  */
          4 WRKTBF  BIT(1),           /* TERMINAL BUFFER FULL IND. */
          4 WRKERR  BIT(1),           /* ERROR DETECTED = '1'B     */
          4 *       BIT(3);           /* FUTURE USE                */
```

For ID2=1E the following working storage is traced in 4 trace entries:

```
 DCL 1 WRK,
       2 WRKID    CHAR(12),          /*IDENTIFICATION **DSLMPSSY** */
       2 *,
         3 SAVLDSA  PTR,              /* ADDRESS LDS               */
         3 SAVEDITA PTR,              /* ADDRESS EDIT BUFFER       */
         3 SAVRKEYA PTR,              /* ADDRESS REKEY BUFFER      */
         3 SAVIOA   PTR,              /* ADDRESS I/O-BUFFER        */
         3 PIO      PTR,              /* CURRENT POSITION IN       */
                                      /* IO BUFFER                 */
         3 SAVIOPTR PTR,              /* SAVE ADDRESS OF IO-BUFFER */
                                      /* WHENEVER A NEW ROW OF DATA*/
                                      /* STARTS                    */

         3 SAVSNDA  PTR,              /* ADDRESS SEND STATUS       */
         3 SAVSESSA PTR,              /* ADDRESS SESSION CONTROL   */
         3 SAVTUCBA PTR,              /* ADDRESS TUCB              */
         3 SAVCOMA  PTR,              /* ADDRESS COMMON AREA       */
         3 SAVMFSPA PTR,              /* ADDRESS PERMANENT STORAGE */
         3 SAVTOFA  PTR,              /* ADDRESS TOF               */
       2 *,                           /* WORKING VARIABLES         */
         3 WRKROWST,                  /* SEND STATUS AT BEGINNING  */
                                      /* OF EACH ROW               */
          4 WRKNXITO BIN (15),        /* NEXT ITEM OFFSET          */
          4 WRKITDO  BIN (15),        /* DATA OFFSET INTO EDIT OR  */
                                      /* REKEY BUFFER              */
          4 WRKNXRW  BIN (15),        /* ROW NUMBER TO BE PROCESSED*/
                                      /* NEXT                      */
          4 WRKDEVO  BIN (15),        /* NEXT FREE OFFSET IN       */
                                      /* PRINTER BUFFER            */
         3 WRKRETC  BIN(8),           /* RETURN CODE               */
         3 *        BIN (8),          /* FUTURE USE                */
         3 WRKREAS  BIN(15),          /* REASON CODE               */
         3 WRKDAOFF  BIN (15),        /* DATA OFFSET: IS USED IF   */
                                      /* ONLY PART OF THE SLOT DATA*/
                                      /* WAS MOVED INTO IO BUFFER  */

         3 WRKCURLO  BIN (15),        /* OFFSET TO CURRENT LDS ITEM*/
                                      /* IS USED WHEN REENTER IS   */
                                      /* NECESSARY                 */
         3 WRKPROW   BIN (15),        /* PREVIOUSLY PROCESSED ROW  */
         3 WRKCCOL   BIN (15),        /* CURRENTLY PROCESSED COLUMN*/
         3 WRKCROW   BIN (15),        /* CURRENTLY PROCESSED ROW   */
         3 *         BIT (16),
           4 WRKFUNC BIT (1),         /* FUNCTION TO BE PERFORMED  */
                                      /* 0=PUT 1=REENTER           */
           4 WRKALLM BIT (1),         /* ALL DATA OF THIS SLOT HAS */
                                      /* BEEN MOVED INTO IO BUFFER */
           4 BLOCKFD BIT (1),         /* INDICATES THAT A VALID    */
                                      /* LDS BLOCK WAS FOUND AND   */
```

```
                                 /* THE LOOP CAN BE TERMINATED*/
      4 ROWOVFL BIT (1),         /* ITEM DATA FLOWS BEYOND A  */
                                 /* PRINTER LINE              */
      4 ONLYSK  BIT (1),         /* THE LDS IS COMPLETELY DONE*/
                                 /* ONLY SKIP LINES  MUST BE  */
                                 /* INSERTED TO FILL THE PAGE */
      4 NEWLINE BIT (1),         /* PROGRAM PROCESSES A NEW   */
                                 /* LINE                      */
      4 *       BIT (10),        /* FUTURE USE                */
    3 WRKSTAT   BIT(8),          /* PROCESSING OPTIONS        */
      4 WRKLDSC BIT(1),          /* LDS COMPLETE INDICATOR    */
      4 WRKLDSI BIT (1),         /* INVALID LDS               */
      4 WRKIOF  BIT(1),          /* IO BUFFER FULL INDICATOR  */
      4 WRKTBF  BIT(1),          /* TERMINAL BUFFER FULL IND. */
      4 WRKERR  BIT(1),          /* ERROR DETECTED = '1'B     */
      4 WRKLINC BIT(1),          /* LINE COMPLETED            */
      4 *       BIT(2);          /* FUTURE USE                */
```

For ID2=23 the following working storage is traced in 4 trace entries:

```
DCL 1 WRK,
      2 WRKID    CHAR(12),          /*IDENTIFICATION **DSLMPSSC** */
      2 *,
        3 SAVLDSA  PTR,              /* ADDRESS LDS               */
        3 SAVEDITA PTR,              /* ADDRESS EDIT BUFFER       */
        3 SAVRKEYA PTR,              /* ADDRESS REKEY BUFFER      */
        3 SAVIOA   PTR,              /* ADDRESS I/O-BUFFER        */
        3 PSCS    PTR,               /* CURRENT POSITION IN       */
                                     /* IO BUFFER                 */
        3 SAVIOPTR PTR,              /* SAVE ADDRESS OF IO-BUFFER */
                                     /* WHENEVER A NEW ROW OF DATA*/
                                     /* STARTS                    */
        3 SAVSNDA  PTR,              /* ADDRESS SEND STATUS       */
        3 SAVSESSA PTR,              /* ADDRESS SESSION CONTROL   */
        3 SAVTUCBA PTR,              /* ADDRESS TUCB              */
        3 SAVCOMA  PTR,              /* ADDRESS COMMON AREA       */
        3 SAVMFSPA PTR,              /* ADDRESS PERMANENT STORAGE */
        3 SAVTOFA  PTR,              /* ADDRESS TOF               */

      2 *,                           /* WORKING VARIABLES         */
        3 WRKROWST,                  /* SEND STATUS AT BEGINNING  */
                                     /* OF EACH ROW               */
         4 WRKNXITO BIN (15),        /* NEXT ITEM OFFSET          */
         4 WRKITDO  BIN (15),        /* DATA OFFSET INTO EDIT OR  */
                                     /* REKEY BUFFER              */
         4 WRKNXRW  BIN (15),        /* ROW NUMBER TO BE PROCESSED*/
                                     /* NEXT                      */
         4 WRKDEVO  BIN (15),        /* NEXT FREE OFFSET IN       */
                                     /* PRINTER BUFFER            */
        3 WRKRETC  BIN(8),           /* RETURN CODE               */
        3 *        BIN (8),          /* FUTURE USE                */
        3 WRKREAS  BIN(15),          /* REASON CODE               */
        3 WRKDAOFF  BIN (15),        /* DATA OFFSET: IS USED IF   */
                                     /* ONLY PART OF THE SLOT DATA*/
                                     /* WAS MOVED INTO IO BUFFER  */
        3 WRKCURLO  BIN (15),        /* OFFSET TO CURRENT LDS ITEM*/
                                     /* IS USED WHEN REENTER IS   */
                                     /* NECESSARY                 */
        3 WRKPROW   BIN (15),        /* PREVIOUSLY PROCESSED ROW  */
        3 WRKCCOL   BIN (15),        /* CURRENTLY PROCESSED COLUMN*/
        3 WRKCROW   BIN (15),        /* CURRENTLY PROCESSED ROW   */
        3 *         BIT (16),
          4 WRKFUNC BIT (1),         /* FUNCTION TO BE PERFORMED  */
                                     /* 0=PUT 1=REENTER           */
          4 WRKALLM BIT (1),         /* ALL DATA OF THIS SLOT HAS */
                                     /* BEEN MOVED INTO IO BUFFER */
          4 BLOCKFD BIT (1),         /* INDICATES THAT A VALID    */
                                     /* LDS BLOCK WAS FOUND AND   */
```

```
                                        /* THE LOOP CAN BE TERMINATED*/
          4 ROWOVFL BIT (1),            /* ITEM DATA FLOWS BEYOND A  */
                                        /* PRINTER LINE              */
          4 ROWCHNG BIT (1),            /* THE PREVIOUS ITEM WAS NOT */
                                        /* IN THE SAME ROW AS THE    */
                                        /* CURRENT ITEM.             */
          4 VTABSET BIT (1),            /* VERTICAL TAB FORMATS ARE  */
                                        /* SET.                      */
          4 HTABSET BIT (1),            /* HORIZONTAL TAB FORMATS    */
                                        /* ARE SET.                  */
          4 SKIPEND BIT (1),            /* ONLY LAST VTAB MUST BE SET*/
                                        /* TO SKIP TO PAGE END       */
          4 *       BIT (8),            /* FUTURE USE                */
        3 WRKSTAT   BIT(8),             /* PROCESSING OPTIONS        */
          4 WRKLDSC BIT(1),             /* LDS COMPLETE INDICATOR    */
          4 WRKLDSI BIT (1),            /* INVALID LDS               */
          4 WRKIOF  BIT(1),             /* IO BUFFER FULL INDICATOR  */
          4 WRKTBF  BIT(1),             /* TERMINAL BUFFER FULL IND. */
          4 WRKERR  BIT(1),             /* ERROR DETECTED = '1'B     */
          4 *       BIT(3);             /* FUTURE USE                */
```

For ID2=28 the following working storage is traced in 2 trace entries:

```
DCL 1 WRK,
      2 WRKID    CHAR(12),             /*IDENTIFICATION **DSLMPULD** */
      2 *,
        3 SAVLDSA  PTR,                 /* ADDRESS LDS               */
        3 SAVIOA   PTR,                 /* ADDRESS I/O-BUFFER        */
        3 P3270    PTR,                 /* CURRENT POSITION IN       */
                                        /* IO BUFFER                 */
        3 SAVTUCBA PTR,                 /* ADDRESS TUCB              */
        3 SAVCOMA  PTR,                 /* ADDRESS COMMON AREA       */
        3 SAVMFSPA PTR,                 /* ADDRESS PERMANENT STORAGE */
      2 *,                              /* WORKING VARIABLES         */
        3 WRKRETC  BIN(8),              /* RETURN CODE               */
        3 *        BIN (8),             /* FUTURE USE                */
        3 WRKREAS  BIN(15),             /* REASON CODE               */
        3 WRKCLIN  BIN (15),            /* LINE NUMBER THAT CONTAINS */
                                        /* THE DATA CURRENTLY        */
                                        /* PROCESSED                 */
        3 WRKCCOL  BIN (15),            /* COLUMN NUMBER THAT        */
                                        /* CONTAINS THE DATA THAT IS */
                                        /* CURRENTLY PROCESSED       */
        3 *         BIT (8),
          4 ITEMFD  BIT (1),            /* ITEM  FOUND IN LDS AND THE*/
                                        /* APPROPRIATE DATA IN       */
                                        /* IO-BUFFER                 */

          4 BLOCKFD BIT (1),            /* INDICATES THAT A VALID    */
                                        /* LDS BLOCK WAS FOUND AND   */
                                        /* THE LOOP CAN BE TERMINATED*/
          4 *       BIT (5),            /* FUTURE USE                */
        3 WRKSTAT   BIT(8),             /* PROCESSING OPTIONS        */
          4 WRKLDSC BIT(1),             /* LDS COMPLETE INDICATOR    */
          4 WRKLDSI BIT (1),            /* INVALID LDS               */
          4 WRKIOF  BIT(1),             /* IO BUFFER FULL INDICATOR  */
          4 WRKERR  BIT(1),             /* ERROR DETECTED = '1'B     */
          4 *       BIT(4);             /* FUTURE USE                */
```

For ID2=32 and ID2=33 the following working storage is traced in 3 trace entries:

```
DCL 1 WRK,
      2 WRKID    CHAR(12),             /*IDENTIFICATION **DSLMPUTF** */
      2 *,
        3 SAVLDSA  PTR,                 /* ADDRESS LDS               */
        3 SAVIOA   PTR,                 /* ADDRESS I/O-BUFFER        */
        3 SAVWDWA  PTR,                 /* ADDRESS OF WINDOW CONTROL */
                                        /* BUFFER                    */
      3 SAVTUCBA PTR,               /* ADDRESS TUCB               */
      3 SAVMFSTA PTR,               /* ADDRESS TEMPORARY STORAGE */
      3 SAVMFSPA PTR,               /* ADDRESS PERMANENT STORAGE */
      3 SAVRKYA  PTR,               /* ADDRESS OF REKEY BUFFER   */
      3 SAVTOFA  PTR,               /* ADDRESS OF TOF             */
      3 SAVLFPA  PTR,               /* ADDRESS OF LINE FORMATTER */
                                    /* CONTROL TABLE              */
    2 *,                            /* WORKING VARIABLES          */
      3 WRKRETC  BIN(8),            /* RETURN CODE                */
      3 WRKWDWID BIN (8),           /* CURRENT WINDOW ID          */
      3 WRKREAS  BIN(15),           /* REASON CODE                */
      3 WRKMSGNO BIN (15),          /* MESSAGE NUMBER             */
      3 WRKWCBOF BIN (15),          /* OFFSET OF WINDOW CONTROL  */
                                    /* BLOCK THAT IS CURRENTLY   */
                                    /* PROCESSED                  */
      3 WRKCROF  BIN (15),          /* OFFSET IN LDS TO THE FIELD*/
                                    /* WHERE THE CURSOR SITS     */
      3 *         BIT (8),
        4 WRKCRFD BIT (1),          /* FIELD FOUND WHERE THE     */
                                    /* CURSOR SITS AND WHETHER IT*/
                                    /* IS A CURSOR SELECT FIELD  */
        4 WRKCMD  BIT (1),          /* INDICATES IF COMMAND LINE */
                                    /* CONTAINS A COMMAND ALREADY*/
        4 BLOCKFD BIT (1),          /* INDICATES THAT A VALID    */
                                    /* LDS BLOCK WAS FOUND AND   */
                                    /* THE LOOP CAN BE TERMINATED*/
        4 *       BIT (5),          /* FUTURE USE                 */
      3 WRKSTAT   BIT(8),           /* PROCESSING OPTIONS         */
        4 WRKLDSI BIT (1),          /* INVALID LDS                */
        4 WRKWDWI BIT(1),           /* INVALID WINDOW CONTROL    */
        4 WRKERR  BIT(1),           /* ERROR DETECTED = '1'B     */
        4 *       BIT(5);           /* FUTURE USE                 */
```

DSLTOFSV TOF Supervisor (ID1=0F):

```
 ID2 Program                                    DATA

 46  DSLTOFSV Entry to Call                     '**DSLTOFSV**',->PL,->CB,FTYP
 47  DSLTOFSV Exit from Call                      FDNAM,RC,RSC,->PL,->CB,FCMO
```

->PL	pointer to the TOFSV parameter list
->CB	pointer to the TOFSV Control Block (DSLTCTLB)
FTYP	function type in TOFSV parameter list
FDNAM	field name found in TOFSV request
RC	return code in TOFSV parameter list
RSC	reason code in TOFSV parameter list
FCMO	function modifier in TOFSV parameter list

DSLEUD End User Driver Program (ID1=14):

```
 ID2  Program                                 DATA

 xx   DSLEUD   Session     start              8X'00', lterm, 'SESSTART'
 xx   DSLEUD   Transaction start              uid, lterm, 'TACSTART'
 xx   DSLEFUN  Entry                          'EFUN ENTRY',EUD control
                                              info
 xx   -        Function program               PGM mark,EFUN control info
               entry
 xx   DSLEFUN  Return from function           'EFUN ',function name,
               program                        Function PGM return info
 xx   DSLEUD   Return from DSLEFUN            'EUD ', EUD control info
 xx   DSLEUD   Transaction end                uid, lterm, 'TACEND'
 xx   DSLEUD   Session     end                uid, lterm, 'SESEND'
```

ID2 is set to 0 at the beginning of a session and then incremented for every screen cycle in a wrap-around manner. The session field is set at the beginning of a session and must not be changed by any service programs.

- For CICS: The task ID from the CICS Exec Interface Buffer (EIB) is used.
- For IMS: The sequence number of the logical terminal name in the MERVA ESA terminal feature definition table (DSLTFDT) is used.

```
uid       is the 8-byte userid.
lterm     is the 8-byte logical terminal name.
tucname   is the 8-byte function name
PGM mark  of function program entry :
          DSLEUSR   'EUSR PL='
          DSLEMSG   'EMSG PL='
          DSLECMD   'ECMD PL='
```

Layout of DSLEUD control information:

```
DS    X  CONTROL indicators
*        Process dialog            EQU   X'02'
*        Force sign-off            EQU   X'04'
*        End of IMS work           EQU   X'08'  ( 'QC' after GU )
*        Sign-on accepted          EQU   X'10'
*        IMS : read next SPA       EQU   X'20'
*        Sign-off accepted         EQU   X'40'

DS    X  STATUS program start
*

DS    X  STATUS SIGN-ON SERVICE
*        Sign-on panel to send     EQU   X'01'
*        Sign-on in process        EQU   X'02'
*        Function within sign-on   EQU   X'04'
*        data
*        Sign-on panel to repeat   EQU   X'08'
*        Sign-on without PW        EQU   X'10'
*        Sign-on with PW change    EQU   X'20'

DS    X  STATUS ABNORMAL END
*        Error during EUD initial. EQU   X'02'
*        Error during EUD termin.  EQU   X'04'
*        Dump to print             EQU   X'80'

DS    X  STATUS SIGN-OFF
*        Sign-off in process       EQU   X'01'
*        Sign-off panel sent       EQU   X'02'
*        SOF without MFS/TOF       EQU   X'10'
*        IMS message after SOF     EQU   X'20'
*        EUD start/end error       EQU   X'40'

DS    X  'EUD-TO-DSLEFUN' IN SPECIAL CASES
*        repeat function selection EQU   X'01'
DS    X  DSLHCP reserved
DS    X  ERROR INDICATOR 1
*        no NUCPARMS               EQU   X'40'
*        no terminal               EQU   X'10'
*        GETMAIN for SPA failed    EQU   X'08'
*        error accessing IMS SPA   EQU   X'04'
*        internal SPA file error   EQU   X'02'

DS    X  ERROR INDICATOR 2
*        no ICB                    EQU   X'80'
*        no MFS service            EQU   X'40'
*        no Operator MSG module    EQU   X'20'
*        no TOF service            EQU   X'10'
*        no QMG service            EQU   X'08'

DS    X  DC-SYSTEM INDICATOR
*        CICS                      EQU   X'08'
*        IMS                       EQU   X'80'
```

Layout of DSLEFUN control information (from the first byte of the interface buffer):

```
DS    A  ADDRESS WORKING STORAGE
DS    A  ADDRESS SPA STORAGE
DS    A  ADDRESS DSLCOM
DS    X  TYPE OF CALL
*        Processing call           EQU   X'00'
*        Initialization call       EQU   X'01'
*        Termination call          EQU   X'02'
DS    X  REASON FOR TERMINATION CALL
*        Forced by function PGM
*        because processing went
*        wrong                     EQU   X'01'
*        Forced by EUD
*        send panel failed         EQU   X'04'
DS    X  COMMAND CODE OF SESSION COMMANDS
*        Sign-off command          EQU   X'04'
*        Return command            EQU   X'08'
DS    X  FREE
```

Layout of function program return information (from the return code and subsequent bytes of the interface buffer):

```
DS    X  RETURN CODE
*        Normal return             EQU   X'00'
*        Sign-off forced           EQU   X'01'
*        forced return to
*        function selection        EQU   X'02'
*        calltype invalid          EQU   X'04'
DS    X  free
DS    X  REASON FOR FORCED SIGN-OFF / RETURN
*        DSLMFS error              EQU   X'01'
*        DSLNUC error              EQU   X'02'
*        DSLQMG error              EQU   X'04'
*        DSLTOF error              EQU   X'08'
*        Function PGM error        EQU   X'10'
*        SHUTDOWN entered          EQU   X'20'
DS    X  free
DS    X  DSLEUD ACTIVITY AFTER RETURN
*        SNAP dump to take         EQU   X'01'
*        Confirmation request      EQU   X'04'
*        reject session command    EQU   X'08'
*        keep command in cmd line  EQU   X'10'
*        protect the message       EQU   X'40'
*        set cursor on commandline EQU   X'80'
DS    X  PANEL TYPE TO SELECT PF KEYTABLE
*        set 'Sel Menu' typ        EQU   X'01'
*        set 'MSG panel' typ       EQU   X'02'
*        set 'LIST panel' typ      EQU   X'04'
*        set 'Read only panel' typ EQU   X'08'
```

DSLHCP Hardcopy Printing Transaction (ID1=19):

```
 ID2 Program                                    DATA

 01  DSLHCP   Trace Header                      DSLHCP queue lterm
 00  DSLHCP   Trace Stop Record                 uid    lterm xxxxx
```

where xxxxx is 'ERREND', 'QUE@END' or 'QUE@CONT'

DSLCXT Checking and Expansion Transaction (ID1=1A):

```
 ID2 Program                                    DATA

 00  DSLCXT   Trace Header                      'DSLCXT TRACE HEADER'
 00  DSLCXT   Start Record                       'DSLCXT' function timestamp
 00  DSLCXT   Stop Record                        'DSLCXT STOP',ret code
 00  DSLCXT   Trace Stop Record                  'DSLCXT TRACE STOP'
```

DSLSDI Sequential Data Set Input (ID1=1B):

```
 ID2 Program                                     DATA

 00  DSLSDI   Trace Header                       'DSLSDI TRACE HEADER'
 00  DSLSDI   Start Record                       'DSLSDI START'
 00  DSLSDI   Stop Record                        'DSLSDI STOP',ret code
 00  DSLSDI   Trace Stop Record                  'DSLSDI TRACE STOP'
```

DSLSDO Sequential Data Set Output (ID1=1C):

```
 ID2 Program                                     DATA

 00  DSLSDO   Trace Header                       'DSLSDO TRACE HEADER'
 00  DSLSDO   Start Record                       'DSLSDO START'
 00  DSLSDO   Stop Record                        'DSLSDO STOP',ret code
 00  DSLSDO   Trace Stop Record                  'DSLSDO TRACE STOP'
```

DSLSDY Batch Printing (ID1=1D):

```
 ID2 Program                                     DATA

 00  DSLSDY   Trace Header                       'DSLSDY TRACE HEADER'
 00  DSLSDY   Start Record                       'DSLSDY START'
 00  DSLSDY   Stop Record                        'DSLSDY STOP',ret code
 00  DSLSDY   Trace Stop Record                  'DSLSDY TRACE STOP'
```

DSLAPI Application Program Interface (ID1=1E):

```
 ID2 Program                                     DATA

 00  DSLAPI   Trace Header                       'DSLAPI TRACE HEADER'
 00  DSLAPI   Start Record                       'DSLAPI START'
 00  DSLAPI   Stop Record                        'DSLAPI STOP'
```

DWSAUTP Authenticator-Key File Service (ID1=38):

```
 ID2 Program                                     DATA

 00  DWSAUTP  Entry                              Bytes 8-1F of Parm. List
 FF  DWSAUTP  Exit                               Bytes 8-1F of Parm. List
```

Bytes 8-1F of Parm. List are:

```
AUTPMSG  DC    A(0)                SWIFT - MESSAGE ADDR (AUT)
*                                  OR PROTOCOL BUFFER (UPDATE)
AUTPTYP  DC    XL1'00'             TYPE OF REQUEST
AUTPINIT EQU   X'00'               INIT
AUTPAUT  EQU   X'04'               PREPARE AUTHENTICATION
AUTPTERM EQU   X'08'               TERMINATION
AUTPSTAT EQU   X'0C'               KEY FILE STATUS
AUTPUPD  EQU   X'10'               UPDATE
AUTPCONT EQU   X'14'               CONTINUE DEL, EXC, OR LIS

AUTPUPDF DC    XL1'00'             UPDATE FUNCTION
AUTPADD  EQU   X'04'               ADD ONE ENTRY
AUTPKEY  EQU   X'04'               OLD ADD FUNCTION CODE
AUTPDEL  EQU   X'08'               DELETE ENTRIES
AUTPCHA  EQU   X'0C'               CHANGE (REPLACE) ONE ENTRY
AUTPLIS  EQU   X'10'               LIST/INQUIRY FOR ENTRIES
AUTPINQ  EQU   X'10'               LIST/INQUIRY FOR ENTRIES
AUTPUNL  EQU   X'14'               UNLOAD PART OF THE KEY FILE
AUTPEXC  EQU   X'18'               EXCHANGE KEYS
AUTPBKID EQU   X'1C'               SHOW BKID
AUTPUMX  EQU   X'1C'               MAXIMUM UPDATE FUNCTION
```

```
AUTPRESC DC    XL1'00'             REASON CODE
*        REASON CODES WITHOUT DIAGNOSTIC MESSAGE IN AUTPEMSG
AUTROK   EQU   00                  O.K. ALL WELL DONE
AUTRCONT EQU   04                  O.K., CONT FOR DEL,EXC,LIS
*        REASON CODES WITH DIAGNOSTIC MESSAGE IN AUTPEMSG
AUTRPTOF EQU   01                  DWSPREM TOF ERROR
AUTRPMFS EQU   02                  DWSPREM MFS ERROR
*
AUTRFLOW EQU   05                  LOWEST FINCOPY VALUE
AUTRFTOF EQU   05                  DWSFCPY TOF ERROR
AUTRFBYP EQU   06                  PAC EMPTY (BYPASS MODE)
AUTRFNOF EQU   07                  NO FINCOPY DEFINITION
AUTRFTXT EQU   08                  NO PAC FIELDS IN TEXT (BLOCK 4)
AUTRFHI  EQU   08                  HIGHEST FINCOPY VALUE
*
AUTRLOW  EQU   24                  LOWEST ALLOWED VALUE.
AUTRIOK  EQU   24                  AUTH INPUT OK
AUTRICDT EQU   25                  INVALID CORRESPONDENT DATE
AUTRISDT EQU   26                  INVALID START DATE
AUTRIEDT EQU   27                  INVALID END DATE
AUTRCERR EQU   28                  UNKNOWN CORRESPONDENT STATUS
AUTRCINV EQU   29                  AUTHENTICATION RECORD NOT VALID
AUTRCEXC EQU   30                  AUTHENTICATION RECORD EXCLUSION
AUTRCSUS EQU   31                  AUTHENTICATION RECORD SUSPENSION
AUTRMANU EQU   32                  UNAUTH. DATA IN RECORD     (MERGE)
AUTRMAN2 EQU   33                  MANUAL KEYS FROM MERVA/2   (MERGE)
AUTRMSEQ EQU   34                  MANUAL KEYS AFTER BKE KEYS (MERGE)
AUTRMANY EQU   35                  TOO MANY KEYS              (MERGE)
AUTRIAT  EQU   36                  INVALID ACTIVE TIME
AUTRIET  EQU   37                  INVALID EXPIRY TIME
AUTEADAT EQU   38                  ACTIBE ¬= PREVIOUS EXPIRE
AUTRDSF  EQU   39                  DATASET FULL
AUTRVSE  EQU   40                  VSAM ERROR DURING ADD
AUTRNRF  EQU   41                  NO RECORDS FOUND
AUTRNK   EQU   42                  RECORD CONTAINS NO KEY FOR AUT
AUTRNRTA EQU   43                  NO RECORD TO AUTHENTICATE WITH
AUTRSKNS EQU   44                  SMALL KEYS NO LONGER SUPPORTED
AUTAEDAT EQU   45                  ACTIVE > EXPIRY
AUTAOPDR EQU   46                  ACTIVE OUTSIDE RANGE OF PREV. DISC.
*        EQU   47                     (not used)
AUTRIAD  EQU   48                  INVALID ACTIVE DATE
AUTRIED  EQU   49                  INVALID EXPIRY DATE
AUTRADI  EQU   50                  AUTH DATA FOR RELOAD OF ADD PENDING
AUTRNOM  EQU   51                  MESSAGE DOES NOT HAVE A MAC TRAILER
AUTRRNX  EQU   52                  RECORD DOES NOT EX.(REP,LST,DEL,AUT)
AUTRRAX  EQU   53                  RECORD ALREADY EXIST    (ADD)
AUTREDA  EQU   54                  NO RECORDS FOR EXCHANGE DATE (EXC)
AUTRNKA  EQU   55                  NO NEW KEYS AVAILABLE OR DATE (EXC)
AUTREDI  EQU   56                  EXCHNAGE/LIST DATE INVALID
AUTRIHLT EQU   57                  INVALID HOME LT
AUTRSMI  EQU   58                  SWIFT MESSAGE INVALID   (AUT)
AUTR13N2 EQU   59                  KEYS 1ST AND 3RD BUT NO 2ND
AUTRSMTL EQU   60                  MESSAGE TOO LONG TO ADD AUT TRAILER

AUTRANO  EQU   62                  AUTHENT OUTPUT ERROR RECORD
AUTRADOK EQU   63                  AUTHENT OUTPUT OK DISCONTINUED
AUTRAOK2 EQU   64                  AUTHENT OUTPUT OK RECORD, NOT CUR.
AUTRAOK  EQU   65                  AUTHENT OUTPUT OK RECORD
AUTRMNA  EQU   66                  MESSAGE NOT TO BE AUTH. (AUT)
AUTRSMTT EQU   67                  MSG TYPE NOT FOUND IN MTT (A
AUTRICL  EQU   68                  INVALID CORR LT
AUTRIUF  EQU   69                  INVALID UPDATE FUNCTION
AUTRCSE  EQU   70                  CALLING SEQUENCE ERROR
*              71                  RESERVED KEYS FOUND IN AGING TABLE
AUTREOF  EQU   72                  END OF FILE
AUTRMNT  EQU   74                  RETRIEVED MSG HAS NO TEXT
AUTRKIN  EQU   75                  OLD SENDING   KEY INVALID
```

```
AUTRISTK EQU   76                 STK DOES NOT MATCH STK KCV
AUTRAOPF EQU   79                 AUTHENT OUTPUT ERROR PARAMETER KEY
AUTRAOPS EQU   80                 AUTHENT OUTPUT OK PARAMETER KEY
*        REASON CODES WHICH REQUIRE DWSAUTP TERMINATION
AUTRTERM EQU   83                 CODES ABOVE THIS CAUSE AUTP TO DUMP
AUTRSRV  EQU   83                 FAILURE IN DSLSRV
AUTRJRN  EQU   84                 FAILURE IN DSLJRNP DURING UPDATE
AUTROPER EQU   85                 OPEN ERROR
AUTRFIV  EQU   86                 FILE INVALID VERSION NUMBER
AUTRFDS  EQU   87                 FORMAT AUT DATASET
AUTRFCPY EQU   89                 FAILED TO LOAD DWSFCPY
AUTRCIT  EQU   89                 FAILED TO LOAD DWSCIT
AUTRPREM EQU   90                 FAILED TO LOAD DWSPREM
*
AUTPHOLT DC    XL8'0'             HOME LOGICAL TERMINAL
AUTPCOLT DC    XL8'0'             CORRESPONDING LT
```

DWSEAUT Authenticator-Key File Online Maintenance (ID1=39):

```
 ID2 Program                                 DATA

 00  DWSEAUT  Entry                          24 bytes of work stor.
 FF  DWSEAUT  Exit                           24 bytes of work stor.
```

The 24 bytes of working storage are as follows:

```
EAUWTRAC DS    0CL24                  DATA PART OF TRACE ENTRY
EAUWTPGM DS    CL8                    PROGRAM NAME 'DWSEAUT '
EAUWTTYP DS    X                      EUD REQUEST TYPE
EAUWTACT DS    X                      EUD ACTION CODE
EAUWTRET DS    H                      RETURN CODE
EAUWTREA DS    H                      REASON CODE
EAUWTMSI DS    H                      MESSAGE ID
EAUWTMGC DS    CL8                    MESSAGE CODE
```

DSLKQS MERVA-MQI Attachment Send Program (ID1=50):

```
 ID2 Program                                 DATA

 00  DSLKQS   Start Record                   'DSLKQS START' time
 00  DSLKQS   Stop Record                    'DSLKQS STOP' time
 00  DSLKQS   Function                       'DSLKQS FUNCTION' tucname
 00  DSLKQS   Send Queue                     'Q' sqname 'OF S' sprocname
 00  DSLKQS   Send Queue                     'QUEUE' squeue
 00  DSLKQS   Send Process                   'SPROC' sproc
 00  DSLKQS   MQI Commit                     'S'p'C'cmtdata
 00  DSLKQS   User Exit                      'S'p'X'extdata
 01  DSLKQS   MQI Request                    'S'p'Q'reqdat1
 02  DSLKQS   MQI Request                    'SBQ'reqdat2
 03  DSLKQS   MQI Request                    'SBQ'reqdat3
```

The variable part of the trace data is represented by the following lowercase symbols:

cmtdata

Consists of the following data:

```
DS    0CL21
DS    CL5                     Number of committed messages
DS    CL8                     Name of the send process
DS    CL8                     Name of the send or control queue
```

extdata

Consists of the following data:

```
DS    0CL21
DS    CL5                     MFS user exit number
DS    CL8                     Name of the send process
DS    CL8                     Name of the send queue
```

p The following values can occur:

A After the event.

B Before the event.

The values are associated to the following events:

- MQI Commit
- MQI Request
- User Exit.

reqdat1

Consists of the following data:

```
DS    0CL21
DS    CL5                     MQI request identifier
DS    CL16                    MQI resource name
```

The MQI request identifiers are provided without the leading characters "MQ". The following MQI request identifiers can occur:

- CLOSE
- CONN
- DISC
- GET
- INQ
- OPEN
- PUT
- PUT1.

The relationship between an MQI request identifier and an MQI resource name is as follows:

CLOSE

Send or control queue name.

CONN

Queue manager name or blanks.

DISC Queue manager name or blanks.

GET Control queue name.

INQ Send queue name.

OPEN Send or control queue name.

PUT Send or control queue name.

PUT1 Remote reply-to queue name.

When the trace entry is written before the MQI request (indicated by SBQ...), the following rules apply:

- If the MQI resource name is longer than 16 characters, a second trace record is written containing the next 16 characters (see symbol reqdat2).
- If the MQI resource name is longer than 32 characters, a third trace record is written containing the last 16 characters (see symbol reqdat3).

When the trace entry is written after the MQI request (indicated by SAQ...), only one trace record is written containing the first 16 characters of the MQI resource name.

reqdat2
Consists of the same data as reqdat1 with the following modification: Characters 17 to 32 of the MQI resource name are provided. If the name consists of less than 32 characters, the remaining area is padded with blanks.

reqdat3
Consists of the same data as reqdat1 with the following modification: Characters 33 to 48 of the MQI resource name are provided. If the name consists of less than 48 characters, the remaining area is padded with blanks.

sproc Consists of the following data:

```
DS    0CL18
DS    CL8                     Name of the send process
DS    CL5                     Number of the send process
DS    CL5                     Total number of send processes
```

sprocname
Name of the send process.

sqname
Name of the send queue.

squeue
Consists of the following data:

```
DS    0CL18
DS    CL8                     Name of the send queue
DS    CL5                     Number of the send queue
DS    CL5                     Total number of send queues
```

time The current time in the format 'HH:MM:SS'.

tucname
MERVA ESA function name obtained from the DSLTUCB.

DSLKQR MERVA-MQI Attachment Receive Program (ID1=51):

```
ID2 Program                                    DATA

00  DSLKQR   Start Record                      'DSLKQR START' time
00  DSLKQR   Stop Record                       'DSLKQR STOP' time
00  DSLKQR   Function                          'DSLKQR FUNCTION' tucname
00  DSLKQR   Receive Queue                     'Q' rqind 'OF R' rprocname
00  DSLKQR   Receive Queue                     'QUEUE' rqueue
00  DSLKQR   Receive Process                   'RPROC' rproc
00  DSLKQR   MQI Commit                        'R'p'C'cmtdata
00  DSLKQR   User Exit                         'R'p'X'extdata
01  DSLKQR   MQI Request                       'R'p'Q'reqdat1
02  DSLKQR   MQI Request                       'RBQ'reqdat2
03  DSLKQR   MQI Request                       'RBQ'reqdat3
11  DSLKQR   Triggered Queue                   'DSLKQR TRIGGERQ' tqnam1
12  DSLKQR   Triggered Queue                   tqnam2
13  DSLKQR   Triggered Queue                   tqnam3
```

The variable part of the trace data is represented by the following lowercase symbols:

cmtdata
Consists of the following data:

```
DS    0CL21
DS    CL5                     Number of committed messages
DS    CL8                     Name of the receive process
DS    CL8                     Index of the receive queue
```

The index of the receive queue indicates the position of the receive queue in the receive queue list.

extdata

Consists of the following data:

```
DS    0CL21
DS    CL5                    MFS user exit number
DS    CL8                    Name of the receive process
DS    CL8                    Index of the receive queue
```

The index of the receive queue indicates the position of the receive queue in the receive queue list.

p The following values can occur:

A After the event.

B Before the event.

The values are associated to the following events:

- MQI Commit
- MQI Request
- User Exit.

reqdat1

Consists of the following data:

```
DS    0CL21
DS    CL5                    MQI request identifier
DS    CL16                   MQI resource name
```

The MQI request identifiers are provided without the leading characters "MQ". The following MQI request identifiers can occur:

- CLOSE
- CONN
- DISC
- GET
- INQ
- OPEN
- SET.

The relationship between an MQI request identifier and an MQI resource name is as follows:

CLOSE
Receive queue name.

CONN
Queue manager name or blanks.

DISC Queue manager name or blanks.

GET Receive queue name.

INQ Receive queue name.

OPEN Receive queue name.

SET Receive queue name.

When the trace entry is written before the MQI request (indicated by RBQ...), the following rules apply:

- If the MQI resource name is longer than 16 characters, a second trace record is written containing the next 16 characters (see symbol reqdat2).

- If the MQI resource name is longer than 32 characters, a third trace record is written containing the last 16 characters (see symbol reqdat3).

When the trace entry is written after the MQI request (indicated by RAQ...), only one trace record is written containing the first 16 characters of the MQI resource name.

reqdat2
Consists of the same data as reqdat1 with the following modification: Characters 17 to 32 of the MQI resource name are provided. If the name consists of less than 32 characters, the remaining area is padded with blanks.

reqdat3
Consists of the same data as reqdat1 with the following modification: Characters 33 to 48 of the MQI resource name are provided. If the name consists of less than 48 characters, the remaining area is padded with blanks.

rproc Consists of the following data:

```
DS    0CL18
DS    CL8                     Name of the receive process
DS    CL5                     Number of the receive process
DS    CL5                     Total number of receive processes
```

rprocname
Name of the receive process.

rqind Index of the receive queue.

The index of the receive queue indicates the position of the receive queue in the receive queue list.

rqueue
Consists of the following data:

```
DS    0CL18
DS    CL8                     Index of the receive queue
DS    CL5                     Number of the receive queue
DS    CL5                     Total number of receive queues
```

The index of the receive queue indicates the position of the receive queue in the receive queue list.

time The current time in the format 'HH:MM:SS'.

tqnam1
Name of the triggered MQI receive queue.

The following rules apply:

- If the MQI receive queue name is longer than 8 characters, a second trace record is written containing the next 24 characters (see symbol tqnam2).
- If the MQI receive queue name is longer than 32 characters, a third trace record is written containing the last 16 characters (see symbol tqnam3).

tqnam2
Characters 9 to 32 of the name of the triggered MQI receive queue. If the name consists of less than 32 characters, the remaining area is padded with blanks.

tqnam3

Characters 33 to 48 of the name of the triggered MQI receive queue. If the name consists of less than 48 characters, the remaining area is padded with blanks.

tucname

MERVA ESA function name obtained from the DSLTUCB.

End of Product-Sensitive programming interface

Debugging Traces

Product-Sensitive programming interface

Debugging Trace for MFS and TOF Services

The debugging trace is controlled by DSLMMFS and DSLTOFSV, respectively. You can activate the trace by setting the bits COMTRAMF or COMTRATF, or both of DSLCOM on. In addition there is a formatted snapdump of the actual TOF buffer available via the DSLTSV TYPE=SHOT request. Any output produced is printed to SYSPRINT using DSLPZRT. A trace record is 120 byte in length; on the right hand side the module name and a time stamp is printed. The time stamp shows the current system time (GMT) in the format <HHMMSS.TTTT.

When using the MERVA ESA end-user driver you can use the screen commands **TSHOT** or **$TOFSHOT** to create a formatted snapdump of the actual TOF as described in "TOF Shot" on page 55. The screen command **$TRACE** allows the activation of selected MERVA ESA debugging trace options within a message processing function. The screen command **$DEBUG** displays a help panel showing the use of the **$TRACE** command.

When using the MERVA ESA application programming interface DSLAPI you can activate the debugging trace by setting the field INTRC to C'TT' before calling the interface module.

When using the MERVA ESA batch utility programs DSLSDI, DSLSDO, or DSLSDY you can activate the debugging trace to trace all TOF supervisor and MFS activities concerning a specific message of a batch of messages. For DSLSDI jobs you specify the number of the message to be traced as the sixth parameter on the EXEC statement in the job. For DSLSDO jobs you specify the number of the message to be traced as the seventh parameter on the EXEC statement in the job. For DSLSDY use the sixth parameter.

For example, to trace the formatting for the system printer of the 25th message in queue DMSY0, code the following JCL statement:

```
//PRINT  EXEC PGM=DSLSDY,REGION=2048K,PARM='DMSY0,E,X,,,025'
```

When using intertask communication via MQSeries, you can specify the characters **MQ** instead of the number of a message. This causes the program DSLNMHQ to trace all events related to MQSeries queues, so that you can debug problems related to intertask communication via MQSeries.

Note: When using the debugging trace, the modules DSLMMFS and DSLTOFSV must be link-edited with special parameters. For VSE, the RMODE must be 24; when running under MVS, RMODE ANY is supported. The modules are not reentrant when running the debugging trace. The modules must not reside in write-protected storage, like LPA, CICS RDSA, or CICS ERDSA. Usually it is necessary to relink the modules DSLMMFS and DSLTOFSV into separate load libraries using the link-edit parameters:

- For MVS: `PARM='REUS'`
- For VSE: `PARM='REUS,RMODE=24'`

MFS Debugging Trace

The following MERVA ESA MFS components issue debugging trace information:

DSLMMFS	Interface program
DSLMCHE	Message checking
DSLMXPND	Message expansion
DSLMNOP	Noprompt formatting
DSLMLFP	Line formatter

DSLMMFS Debugging Trace:

- At entry to DSLMMFS the line
```
****** MFS INTERFACE / ENTRY-PARMLIST: ******
```
is printed followed by the MFS parameter list (52 bytes) in dump format. When the DSLCOM or permanent storage addresses are missing in the MFS parameter list, the trace is suppressed.
- At exit, when the MFS return code is not zero, the lines
```
****** MFS INTERFACE / EXIT-ERROR: ******
PARMLIST PREFIX
```
are printed, followed by the MFS parameter list prefix (12 bytes) in dump format. When the field reference address is available in the parameter list, the field reference (15 bytes) is printed in dump format. The MFS error message in buffer MFSPEMSG is printed.
- When MFS dynamic load for a module or MCB is performed the line
```
LOAD MODULE
```
followed by the name of the module or MCB is printed.
- When MFS deletes a module or MCB the line
```
DELETE MODULE
```
followed by the name of the module or MCB is printed.

DSLMCHE Debugging Trace:

- Each message checking cycle starts with the line
```
******  DSLMCHE  / MESSAGE CHECKING    ******
```
- When a field in the TOF is to be checked, the line
```
DSLMCHE CHECK NEXT FIELD:
```
is printed, followed by the field reference (14 bytes) in dump format.
- When the field DSLLFBUF is found in the message, the line
```
DSLMCHE FIELD DSLLFBUF FOUND IN TOF
```
is printed to indicate that a formatting error occurred.

- When one of the field checking routines detected an error, the line

  ```
  DSLMCHE CHECKING ERROR:
  ```

  is printed, followed by the TOF parameter list (76 bytes) in dump format. TOF supervisor and MFS reason codes contained in the parameter list give further information about the error.
- When all TOF fields are processed, the lines

  ```
  DSLMCHE END OF CHAIN REACHED
  TSV-PARMLIST
  ```

  are printed, followed by the TOF parameter list (76 bytes) from the last ACCESS request.

DSLMXPND Debugging Trace:

- Each message expansion cycle starts with the line

  ```
  ******  DSLMXPND / MESSAGE EXPANSION   ******
  ```
- When the field DSLEXIT is not found in the message the lines

  ```
  DSLMXPND READ DSLEXIT FAILED
  DSLTSV PARMLIST
  ```

  are printed, followed by the TOF parameter list (76 bytes) from the READ request. The missing DSLEXIT field indicates that the TOF does not contain a message.
- When a field in the TOF is to be expanded, the line

  ```
  DSLMXPND EXPAND NEXT FIELD:
  ```

  is printed, followed by the field reference (14 bytes) in dump format.
- When a field expansion error occurred, the line

  ```
  DSLMXPND EXPANSION ERROR:
  ```

  is printed, followed by the TOF parameter list (76 bytes) to allow inspection of TOF supervisor and MFS reason codes. When an error message is available in the buffer MFSPEMSG it is printed also.
- When all TOF fields are processed, the line

  ```
  DSLMXPND END OF CHAIN / TSV-PARMLIST:
  ```

  is printed, followed by the TOF parameter list (76 bytes) from the last ACCESS request.

DSLMLFP Debugging Trace:

- When data is detected for a TAG that has no field defined in the MCB (TAG only field), the line

  ```
  DSLMLFP: TAG ONLY FIELD DATA
  ```

  is printed, followed by the data. In this case the data that is not mapped into the TOF is lost.
- When a TOF error occurred during data mapping, the line

  ```
  DSLMLFP: TOF PLIST IN ERROR
  ```

  followed by the TOF parameter list (76 bytes) and the line

  ```
  DSLMLFP: TOF IN ERROR
  ```

  followed by the TOF are printed in dump format.

DSLMNOP Debugging Trace:

- For each GET cycle (noprompt data input from screen), the lines

  ```
  ****** MFS NOPROMPT PROCESSING (GET)   ******
  NOPROMPT CONTROL TABLE (INPUT)
  ```

  are printed, followed by the noprompt control table and the line

  ```
  EDIT BUFFER (INPUT)
  ```

  followed by the edit buffer containing the input data from screen in dump format.
- For each PUT cycle (noprompt data output to screen), the lines

  ```
  ****** MFS NOPROMPT PROCESSING (PUT)   ******
  NOPROMPT CONTROL TABLE (OUTPUT)
  ```

  are printed, followed by the noprompt control table and the line

  ```
  EDIT BUFFER (OUTPUT)
  ```

  followed by the edit buffer containing the output data to be presented on the screen.
- When a noprompt formatting error occurs during PUT cycle, the lines

  ```
  ERROR IN LINE FORMATTING
  ****** MFS NOPROMPT PROCESSING (PUT)   ******
  ```

  are printed. The MFS parameter list prefix of the concerned line formatter call may be inspected to analyze the reason of the error.

TOF Debugging Trace

For each TOF request information is supplied at the end of the request, that is, before DSLTOFSV returns to the caller. Each trace entry is identified by a line showing:

```
***** TOF REQUEST REPORT *****
```

Information supplied in fields with the prefix TSVP is derived from the TOF parameter list TSVPARMS and with the prefix TSVS is derived from the internal TOF Control Block TSVSCB.

The following information is supplied:

- FUNC/MOD:
  - Function type TSVPFTYP
  - Five fields showing the modifiers TSVPMODS for NI, FG, RS, FN, and DA
  - Function modifier, TSVPFCMO (FCMO)
  - Number of key, TSVSNKEY (NKEY) in TOF KEY AREA, if found
  - TOF return code, TSVPRC
  - Address of RS descriptor, TSVSRSDC (RSDC), last accessed
  - Address of data area record, TSVSDARC (DARC), last accessed
  - Offset of data area record in the TOF, TSVSDFND (DFND), if found.
- TSVPCURR:
  - Current position, TSVPCURR (NI,FG,RS,FN,DA,OM) from TSVPARMS
  - TOF reason code, TSVPRSC
  - Address of option field record, TSVSOFRC (OFRC), last accessed

  - Name of last TOF module processed, TSVSNAME (only the last four characters of the name)
  - Internal return code from last TOF module processed, TSVSRCOD (RCOD)
  - Internal return code from last call to DSLTDFND, TSVSDFRC (DRC), indicating whether RS descriptor, data area record or option field was not found.
- TSVPNEXT:
  - Next position parameters, TSVPNEXT (NI,FG,RS,FN,DA,OM) from TSVPARMS
  - Reason code from last MFS request performed by DSLTOFSV, TSVPMFSR (MFSRSC)
  - Last correct RS level in field reference: occurrence required was found on this RS level (TSVSRSLA), significant for nested RS
  - Last occurrence which was found on the first incomplete RS level (OCLA), significant if occurrence not found
  - Number of RS levels of key found in field descriptor extension list in TOF.
- TSVIPOS:
  - The field reference evaluated, TSVIPOS (NI,FG,RS,FN,DA,OM)
  - The corresponding main field name, TSVISFN (MFN) for subfields
  - RSX: Part of field reference in RS extension, displayed only for fields in nested repeatable sequences.
    - Number of RS nesting levels (NO)
    - Internal RS group number (GP)
    - Occurrence number of each RS nesting level, starting with the outermost repeatable sequence.
- STATUS:

  Information saved in the status fields TSVPFSxx (xx= MF,EM,RX,DX,LS, LG,LF,MN,DR), only shown if at least 1 bit is set.
  - Nonzero return code from MFS, TSVPFSMF (MFS ERROR)
  - Data component is empty, TSVPFSEM (EMPTY)
  - More than maximum number of occurrences found TSVPFSRX (RSMAX)
  - More than maximum number of data areas found TSVPFSDX (DAMAX)
  - Field length error found, TSVPFSLS, TSVPFSLG, TSVPFSLF (too small: LENMIN, too long: LENMAX, not fixed: LENFIX)
  - Field is mandatory, TSVPFSMN (MAND)
  - Field may be deleted, TSVPFSDR (DEL) (not currently used).
- SUBFIELD:

  Specifications for this subfield derived from entry in FDT, only shown for subfields.
  - Routine numbers for: Checking (CHK), Editing (EDT), Default Setting (DEF), and Separation (SEP)
  - Length Specifications: LTH1, LTH2, fixed (FIX) or variable (VAR)
  - Offset (OFFS) of subfield in main field
  - Subfield is mandatory (MAND).
- BLEN =

  I/O buffer (TSVPBUFF), length of data in buffer and data as character string (maximum shown 60 characters).
- INTL =

Corresponding information from output buffer supplied for MFS calls (referenced by TSVSBUFO), shown only if contents is different from TOF I/O buffer. This output buffer is not available, if the TOF internal buffer is used (storage already released).

- DSL3nnn

  Explanation of TOF reason code TSVPRSC supplied as a diagnostic message, displayed only for TSVPRSC > 0.
- MFSPEMSR

  Information derived from MFS permanent storage, displayed only for MFSRSC > 99.
  - Error message reason code (MFSPEMSR) and the actual length of the error message (MFSPEMSA)
  - Error message text.

TOF Shot

A formatted snapshot of the current TOF can be obtained by issuing a DSLTSV TYPE=SHOT request. The contents of the TOF is listed in order of the keys as arranged in the TOF key area. The beginning of the TOF shot is identified by:

- ******* HERE STARTS A NEW TOF ***************
- The total number of key entries in the TOF key area

In the following lines information for each key and its associated data is supplied as follows:

- +++ KEY ENTRY #
  - Number of the key. The keys are listed in ascending order.
  - Nesting level (NI =), field group (FG =), field name (FN =), and nesting identifier of next logical nesting level (NID =).
- +++ FD ENTRY and +++ PERM

  In the following two lines information about the related field descriptor entry in the TOF is given.
  - Offset of field descriptor entry in TOF.
  - Number of exit routines defined for checking (CHK), editing (EDT), default setting (DEF), expansion (EXP) and separation (SEP).
  - Length of field descriptor entry.
  - Field features: PERM, QUEUE, Option (OPT), mandatory (MAND), length specifications fixed (LTHX), variable (LTHY), unlimited (LTHU), minimum (LTH1) and maximum (LTH2) length, maximum number of data areas (DAMX), minimum (RSMI) and maximum (RSMX) number of occurrences and number of options in option list (OPTN).
- ++ RS DESCRIPTOR

  In the following lines the associated chain of RS descriptors and its option field and chain of data areas is shown.
  - RS occurrence number (for each level in case of nested RS) and its offset in the TOF.
  - Information about the option field: offset and option data, if available.
  - Information about the chain of data areas, if available.

    For each data area the occurrence number (RS), data area index (DA), the length of the data, the offset of the data area record and the data are shown in two lines.

Note: Two special types of keys with field name DSLRSBEG and DSLRSEND are introduced to identify the start and end of repeatable sequences. These keys are introduced only for repeatable sequences initialized with a RS extension list. These keys have no field descriptor and no chain of RS descriptors assigned. They show the internal RS group number and number of RS levels (NO OF RS INDEXES = 1, for not nested repeatable sequence).

Debugging Traces for Nucleus Server Components

These traces are available for MVS only. You can trace the following nucleus server components:

- Nucleus server shell
- Request queue handler
- MQSeries queue handler.

As an exception, you can also trace the MQSeries nucleus server program (DSLNMQS).

The debugging traces can be activated:

1. By setting the appropriate bits in the Nucleus Server Table (DSLNSVT). This is the static nucleus server shell and request queue handler debugging trace. The static trace is useful if a nucleus server's behavior during its startup is to be traced. You specify the trace area (the nucleus server) by setting the appropriate trace mask. The specified trace is then active at the time MERVA ESA is started. If the trace is no longer needed, you can deactivate the traces with the dynamic trace.
2. By entering the NTRACE command accompanied by the appropriate command parameters. This is the dynamic nucleus server shell and request queue handler debugging trace. You can activate or deactivate traces or alter a static trace.

Generally, the debugging trace is written to the SYSPRINT data set. The nucleus server shell and request queue handler debugging traces can be specified for:

- Trace areas
- Trace level
- Trace depth

Each trace can be either switched on or off.

Trace Areas

Trace areas can be specified by *nucleus server names*. The nucleus server names are defined in the Nucleus Server Table (DSLNSVT) and can also be obtained by entering the **drqa** command. It is also possible to trace **ALL** nucleus servers.

Note: Only those request areas are traced that are controlled by a nucleus server shell, that is, the DSLNUC maintask module is excluded from the dynamic debugging trace mechanism. However, common services such as the display functions for a specific request (DR command), relations to a specific request (DRR command), request queue and nucleus server administrative statistics (DRQA command), and the nucleus server shell and request queue handler debugging trace (NTRACE command) as well as the request queue handler can be traced.

Trace Levels

Trace levels are grouped by modules that are:

- **SHEL** or **SHEC** (CICS) - nucleus server shell (control module DSLNSHEL or DSLNSHEC)

- **RRP** - request ready event processing (module DSLNRRP) Depending on the request content, any of the following modules are also traced:
  - DSLNSNPT - nucleus program start
  - DSLNPNPT - nucleus program stop
  - DSLNNTR - nucleus task request
  - DSLNNCS - nucleus command service
- **SPP** - service processed event processing (module DSLNSPP)
- **RPP** - request postprocessing (module DSLNRPP)
- **PPP** - program postevent processing (module DSLNPPP)
- **RQH** - request queue handler processing (module DSLNRQHx)
- **MQH** – MQSeries queue handler processing (module DSLNMQHx)
- **MQS** – MQSeries nucleus server processing (module DSLNMQSx)

It is also possible to trace **ALL** levels. Use **NULL** or **0** to ignore level trace setting.

Trace Depths

This defines which depth of the level is to be traced:

- **BASE** - gives a processing overview of the level specified for the trace area.
- **REQUEST** - traces the state and contents of a request accompanied with the queue handler functions performed by the specified trace area.
- **EVENT** - traces the wait and postprocessing, the actual state of ECBs, and the contents of ECB address lists accessed by the specified trace areas and levels.
- **DISP** - traces the display processing. You should activate this trace only if you suspect an error in the display processing of the DRQA, DR, or DRR command.
- **TRACE** - traces the trace processing. You should activate this trace only if you suspect an error in the trace processing.

It is also possible to trace **ALL** depths. Use **NULL** or **0** to ignore depth trace setting.

Note: The display functions for a specific request (DR command), relations to a specific request (DRR command), request queue and nucleus server administrative statistics (DRQA command), and the nucleus server shell and request queue handler debugging trace (NTRACE command) can run as separate nucleus servers or can be integrated into the DSLNUC maintask.

How to Choose the Correct Trace

Generally, setting on a trace generates an overhead that influences MERVA ESA performance, the more areas, levels, and depths you select. First, localize the nucleus server where you suspect a problem. Before you switch on a trace, verify if there is no other trace active which might influence performance and falsify the traces you want. First reset all possibly misleading traces. Note that some nucleus servers have a close relationship, such as the queue management and the journaling. In this case, for example, you should select the trace areas for DSLQMGT *and* DSLJRNP.

To get the first processing overview, set on the trace level for the nucleus server shell control module (SHEL or SHEC) and a basic trace depth. If analysis of this trace gives no hint to the problem, categorize the suspected problem as follows:

- Is it during startup or during normal work?
- Is it in a program defined in the Nucleus Program Table (DSLNPTT)? If yes, is it during its start or stop processing?
- Is it during execution of a central service?

- Is it during execution of a command?
- Could the problem be a wait for an outstanding event?
- Is there a queuing problem?

If the problem occurs during normal work, you can use the display commands.

1. Display the administrative request queue and nucleus server statistics. If you see the number of waiting requests for a nucleus server exceeds 2, then display the request currently being processed.

   If you see the number of waiting requests for the DSLNUC exceeding 2, then you should consider to transfer a service currently running under DSLNUC to a subtask nucleus server. You do this by specifying SERVER=TASK for the service in question.
2. The Request Control Element (DSLNRCE) of a specific request. Beside others, this display informs you about the current request state, the request processing type, the number of the nucleus server which added the request, the number of the nucleus server which obtained the request, and the times spent on the different queue types.

   If the server number which obtained the request does not match the one processing the request in the previous display, you should increase the request queue size: the request number you want to observe is already occupied with another request. If there is no finished time, the request is finished with processing, but is still busy signaling the Request Processed event to the server which has added the request you look at. Display the relations to the request you look at.

   If there is no active time, the request is still active. Either the service described in the request is still being processed or the service has added a subsequent synchronous request. Display the relations to the request you look at.
3. The relations of a specific request. If the request for which you display the relation is current, this request has no child and has no longer a parent, since the parent request is no longer active. You should set on the RRP level accompanied with the BASE and REQUEST depth. If the request for which you display the relations is a parent request and has one or more child requests, you should set on the SPP level accompanied with the BASE and EVENT depth. There are two cases where you see more than one child request:
   - The parent request has created a child request while a previous one is still active.
   - A child request has created another child request and all are still active.

   If the request for which you display the relations is a child request, you see also which request number is the parent request. You should set on the RRP level accompanied with the BASE and EVENT depth. If you see brother relations, you should set on the RPP level accompanied with the BASE and EVENT depth. A brother relation occurs if a parent request creates more than one child request. If the brother relation expires, the brother becomes a child.

If you need to know the exact contents of the Request Control Element (DSLNRCE, you can switch on the REQUEST trace depth together with the SHEL/SHEC and RRP trace level.

Do not set the REQUEST trace depth together with the RQH trace level unless you really suspect an error in the request queuing or if you have an ABEND in module DSLNRQH.

Once programs specified in the Nucleus Program Table (DSLNPTT) are started, they run independently in the appropriate nucleus server until they are stopped by a program stop request. If you suspect a problem in a nucleus server processing such a program, you should switch on the SHEL/SHEC and PPP trace levels together with the BASIC trace depth.

If you suspect a problem during startup, you should activate the static debugging trace. If the trace is set for DSLNUC, the request queue handler and all nucleus server shells will be traced from the beginning. If you know which nucleus server shell has the problem, you should set the trace for the appropriate nucleus server. The following is an example of how to set a debugging trace in the Nucleus Server Table (DSLNSVT) for all nucleus server shells and the request queue handler:

```
⋮

*
*                    BIT    VALUE     Meaning
*
*                     0      128    SHEL   BASE
*                     1       64    RRP    REQ
*                     2       32    SPP    EVT
*                     3       16    RPP    DISP
*                     4        8    PPP    TRC
*                     5        4    RQH
*                     6        2    MQH
*                     7        1    MQS
*
         DSLNSV NAME=DSLNUC,SERVER=MAIN,TRACE=(252,224,0,0)
         DSLNSV NAME=DSLNCMD
*
.
.
.
```

This trace provides a debugging trace for all trace areas and all trace levels with the trace depth of BASE, EVENT, and REQUEST

The following is an example of Nucleus Server Table (DSLNSVT) entries to trace only the nucleus server shell of the queue management and the journal:

```
*
* Journal  program  group
         DSLNSV  NAME=DSLJRNP,
               SERVER=TASK,TRACE=(192,160,0,0)
*
* Queue  management  group
         DSLNSV  NAME=DSLQMGT,
               SERVER=TASK,TRACE=(192,160,0,0)
*
```

This trace provides a debugging trace for these trace areas with trace levels of SHEL and RRP and depth of BASE and EVENT. The trace shows the processing of these two nucleus servers at the time when any event is recognized, especially the processing flow of a Request Ready event.

In the NSVT, you can also switch off the ESTAE error recovery routine of a specific nucleus server. To do this you specify TRACE=(xxx,xxx,0,1).

Don’t forget to switch off all traces if no longer needed. To switch off all traces can enter the command:

```
NTRACE ALL,ALL,ALL,OFF
```

Depending on the trace level and depth specified, the following internal structures may be found in the SYSPRINT output.

The Server Control Block (DSLNSCB)

```
DSLNSCB  DS    0F                  Server Control Block
SCBHDR   DS    0CL12               Control block header
SCBID    DC    CL8'*DSLNSCB*'      Control block ID
SCBGMCW  DS    0F                  Getmain Control Word
SCBSP    DS    XL1                 Subpool ID
SCBLEN   DS    XL3                 Control block length
SCBHLEN  EQU   *-DSLNSCB           Length of SCB header
SCBIDS   DS    0F                  IDs
SCBSRVNO DS    F                    server number
SCBSRVNM DS    CL8                  service name
SCBPTRS  DS    0F                  Pointers
SCBPREVP DS    A                    to previous SCB in chain
SCBNEXTP DS    A                    to next SCB in chain
SCBSMBP  DS    A                    to SMB
SCBRQVTP DS    A                    to RQVT
SCBIRQP  DS    A                    to IRQ
SCBIRQAP DS    A                    to IRQ slot added
SCBIRQOP DS    A                    to IRQ slot obtained
SCBCOMP  DS    A                    to DSLCOM of server
SCBNSVEP DS    A                    to servers NSV main entry
SCBEVTAP DS    A                    to event list area
SCBEVTFP DS    A                    to start of server event list
SCBEVTLP DS    A                    to end of server event list
SCBRSPFP DS    A                    to first SP ECB pointer within
*                                   event list for requests
SCBRSPLP DS    A                    to last SP ECB pointer within
*                                   event list for requests
SCBPGMFP DS    A                    to first program ECB pointer
*                                   within event list
SCBPGMLP DS    A                    to last program ECB pointer
*                                   within event list
SCBRRPFP DS    A                    to first RP ECB pointer within
*                                   event list for requests
SCBRRPLP DS    A                    to last RP ECB pointer within
*                                   event list for requests
SCBOFF   DS    0F                  Offsets
SCBSPOFF DS    F                    to a request's service processed
*                                   ECB pointer
SCBRPOFF DS    F                    to a request's service processed
*                                   ECB
SCBFLAGS DS    F                   Flag bits
SCBTERM  EQU   X'80'                subtask termination in progress
SCBSDWA  EQU   X'40'                indicates whether SDWA is available
SCBRTRY  EQU   X'20'                recovery: retry
SCBRECUR EQU   X'10'                recovery: recursion
SCBWAIT  EQU   X'08'                server currently idle
SCBTSTMP DS    0F                  Timestamps
SCBITIME DS    XL8                  initialization time of server
SCBTTIME DS    XL8                  termination time of server
SCBECBS  DS    0F                  ECBs
SCBSAECB DS    F                    subtask attach ECB
SCBSTECB DS    F                    subtask server termination ECB
SCBTMECB DS    F                    subtask server timer ECB
SCBRRECB DS    F                    servers request ready ECB
SCBSRECB DS    F                    subtask return ECB
SCBSTTCB DS    A                   subtask TCB address
SCBCNTRS DS    0F                  Counters
SCBNWTR  DS    F                    current number of requests in
*                                   waiting queue for this server
SCBMISC  DS    0F                  Miscellaneous
```

```
         DS    XL1                  reserved
         DS    XL1                  server mask bits
SCBSERPR DS    XL1                  SCB number of previous enqueue
SCBCS    DS    XL1                  serialization bits: Compare and Swap
SCBCSTB  EQU   X'80'                 Block trace enqueued
SCBCSTS  EQU   X'40'                 Step trace enqueued
SCBCSIRQ EQU   X'04'                 enqueue on servers IRQ
SCBSRVRC DS    F                    return code from last processed
*                                   service
SCBCSECT DS    CL8                 Last called CSECT
SCBRTRYC DS    F                   enqueue retry count
SCBTRCEP DS    A                   temporary RCE pointer
SCB_END  DS    0F                  End of SCB
```

The Request Control Element (DSLNRCE)

```
DSLNRCE  DS    0D                 Request Control Element
RCEDATA  DS    0F                  RCE data section
RCEURI   DS    F                   Unique request identifier
RCESTATE DS    XL1                 Flags: FSM state
RCEFREE  EQU   X'80'                   element is free
RCEWAIT  EQU   X'40'                   element is waiting
RCEACTIV EQU   X'20'                   element is active
RCEFIN   EQU   X'10'                   element is finished
RCEPROC  DS    XL1                 Flags: Processing indicators
RCEBFALC EQU   X'80'                   dynamic buffer is allocated
RCESYNC  EQU   X'40'                   asynchronous processing
RCEBROTH EQU   X'20'                   request has a brother
RCENODAT EQU   X'10'                   no data buffer although address
*                                      in parm
RCEPRTY  DS    H                   Request priority
RCEMISC  DS    0F
RCESRVAN DS    F                   Number of server which added
*                                  the request
RCESRVON DS    F                   Number of server which should
*                                  obtain the request
RCEUSDAT DS    CL32                User data area
RCEORIG  DS    CL8                 Request originator (UID)
RCEPARNP DS    A                   Address of parent request
RCECHILP DS    A                   Address of oldest child
RCEBROTP DS    A                   Address of next younger brother
RCEWTIM  DS    XL8                 Time request was added to the
*                                  waiting queue
RCEATIM  DS    XL8                 Time request was added to the
*                                  active queue
RCEFTIM  DS    XL8                 Time request was added to the
*                                  finished queue
RCEDTIM  DS    XL8                 Time request was removed from the
*                                  finished queue and added to the
*                                  chain of free elements
RCECEP   DS    A                   pointer to asynchronous processing
*                                  control element
RCESCBAP DS    A                   Pointer to SCB of server which
*                                  created (added) the request
RCESCBOP DS    A                   Pointer to SCB of server which
*                                  processed (obtained) the request
RCESPECB DS    A                   Service processed ECB.
*                                  Posted by the server which
*                                  obtained and processed the
*                                  request
RCERPECP DS    A                   Pointer to request processed ECB.
*                                  If no RP ECB address was given
*                                  when added, it points to RCERPECB
RCERPECB DS    A                   Request processed ECB.
*                                  Posted by the server which
*                                  obtained and processed a child
*                                  request. Awaited by the server
*                                   which processed the parent
*                                   request.
RCENUMAD DS    H                    Number of adds performed by this
*                                   parent request
RCERRIND DS    CL3                  Request ready indicator
RCESPIND DS    CL3                  Service processed indicator
RCERPIND DS    CL3                  Request processed indicator
RCEASYNP DS    0CL97                extension for MQI asynchronous
*                                   processing
RCEMQFMT DS    CL1                  MQ message format
RCERTQNM DS    CL48                 reply-to queue name of requesting
*                                   nucleus instance
RCEMQMNM DS    CL48                 MQM name of requesting
*                                   nucleus instance
RCERTQSZ DS    F                    RTQ size of request originating
*                                   nucleus instance
         DS    F                    reserved
RCE_END  DS    0F                   End of RCE
```

End of Product-Sensitive programming interface

MERVA ESA Dump

Product-Sensitive programming interface

Under certain error conditions, a snapshot dump can be produced by the MERVA ESA nucleus or by the End-User Driver. Under CICS, a formatted region dump can also be produced. Under IMS, a dump of BMP or MPP regions can be produced.

A dump of MERVA ESA can also be produced by a user using the **cancel** or **terminat** commands, with either the ABDUMP or DUMP options; see *MERVA for ESA Operations Guide* for more information. Only authorized users can access this facility.

In MERVA ESA most of the programs contain an eye-catcher generated by the DSLCETO macro instruction showing the following information:

- Program name.
- Release level, for example, REL410 for MERVA ESA V4.1.
- Environment, for example, C410MVS for MERVA ESA under CICS Version 4, Release 1, under MVS.
- PTF number. In the initially supplied MERVA ESA, the PTF number is blank.
- APAR number. In the initially supplied MERVA ESA, the APAR number is blank.
- The assembly date and time.

This information allows for easy identification of the installed level.

Dump Analysis

If an error occurs, first check whether there are any error messages. Error messages can be found:

- On the operating system console, especially from the MERVA ESA batch programs and utilities.
- In the MERVA ESA journal.

- On the user screen terminal. These error messages cannot be found elsewhere, therefore the user should write them on paper in order to keep the complete information of the error message.

The error messages together with reason code and return code can give some helpful information for debugging (see *MERVA for ESA Messages and Codes*).

In some cases, a CICS transaction dump or MVS SNAP dump is taken. The dump code explains which program detected the error or which program failed. Print the dumps for easier reference.

The analysis of a dump is primary the analysis of a CICS, MVS, or VSE dump.

In CICS, the transaction code indicates which program issues the dump.

In IMS, the jobname indicates which region issues the dump.

Additional Hints for Analyzing MERVA ESA Dumps

Dump produced by the DSLNUC maintask: If a dump is produced when the ABEND exit in DSLNUC is entered, the PSW and registers at the moment of the program check can be located in DSLNUC behind the eye-catcher NUCSTAE in the dump. In CICS, PSW and registers at entry to ABEND can be found in the DFHTACB located after trace table.

Usually, general register 1 is pointing to a parameter list, general register 10 is used by most programs as the first base register, general register 13 points to a save area, and general registers 14 and 15 are used in BALR instructions.

Additional information can be found in the parameter list of the program called. Refer to the layout of the parameter lists in the program listings.

Refer to DSLCOM, which can be located via the eye-catcher *DSLCOM* and contains pointers to the MERVA ESA tables and modules. The parameter list of most programs contains the DSLCOM address. Usually the general program register 12 points to the DSLCOM.

If called to produce a dump, DSLSRVP saves the prefix area of its parameter list in the field SRVPREFX, which is located in DSLCOM + X'180' (the prefix area contains the previous request type and the return and reason code). DSLSRVP saves the old save area of the caller in the field SRVDEBUG, which is located in DSLCOM + X'188' (just behind the 8-byte SRVPREFX field).

Further analysis of a dump depends on the particularity of the problem.

Dump produced by a DSLNSHEL / DSLNSHEC subtask: Most of the tips mentioned above also apply for dumps produced in a subtask.

Any ABEND which occurs in a subtask is first handled by the ABEND exit responsible for the subtask. This exit does not perform any recovery actions. It simply produces the dump, gathers data important to help you in diagnosing the error and prints it before returning to the system. The maintask then enters its own ABEND exit and starts terminating MERVA ESA.

Dump produced by intertask communication via APPC/MVS: A return code 56 (X'38') returned from DSLNIC service indicates a problem with the intertask communication via APPC/MVS. This problem usually results from incorrect or insufficient specification of parameters in DSLPRM. The ITCAREQ parameter is used by the requestor part of the intertask communication via APPC/MVS. If an error occurs during intertask communication, the MERVA ESA program issues a dump. If the error is APPC/MVS related and returned by program DSLNICTA, the following steps should be performed to find out the cause of the problem.

The dump analysis must start with the DSLCOM. The DSLCOM can be located via the eye-catcher *DSLCOM*; usually the general program register 12 points to the DSLCOM.

The address of the parameter list for the intertask communication (NICPL) is stored in the MERVA ESA DSLCOM at offset X'70'.

The first field in the NICPL is the ICB pointer. For intertask communication via APPC/MVS, the control block starts with a length field followed by the eye-catcher 'ICB NICA'. In case of APPC/MVS errors the error extract message returned from APPC/MVS is stored in the ICB. The prefix of these messages is ATB8. The messages itself are described in *OS/390 MVS Programming: Writing Transaction Programs for APPC/MVS*.

End of Product-Sensitive programming interface

Operator Messages

For various error conditions and events, operator messages are prepared by MERVA ESA and the network links. They are displayed on the system console and journaled by the MERVA ESA master operator program.

With the MERVA ESA command **dm** (display messages) you can display the operator messages issued by MERVA ESA and the network links.

Status Displays

MERVA ESA and its components offer various commands for status displays. See *MERVA for ESA Operations Guide* for details.

There are also specific operator commands available for diagnostic purposes:

DICB	Display the ICB status.
DRQA	Display the actual request queue and server statistics.
DR	Display information about a specific request.
DRR	Display the relations to a specific request.
NTRACE	Activate and deactivate traces to debug the nucleus server shell and the request queue handler.
XTRACE	Activate and deactivate traces to debug the X.25 connections.

These commands are described in the following sections.

Displaying the ICBs (DICB)

Use the **dicb** command to monitor the status of the MERVA ESA intertask communication servers and control blocks (ICBs) used for the intertask communication via CICS temporary storage queues, via APPC/MVS or via MQSeries. This command is mainly for diagnostic purposes. The ICBs of the other methods of intertask communication can be displayed using MERVA ESA help panels or batch utility programs. These methods are described in “Service Aids for Intertask Communication” on page 80.

Command Format

The format of the **dicb** command is:

dicb	[{ SERVERS }] [{ TSQ [, *number* [, *ALL*]]}] [{ APPC [, *number* [, *ALL*]]}] [{ MQI [, *number* [, *ALL*]]}] [{ *srvname* [, *number* [, *ALL*]]}] [{ *srvname* [, , *STOP*]}]

The first time you enter the **dicb** command, the status of the first 13 ICBs is displayed. If you enter the same command with the same parameter again, the status of the next 13 ICBs is displayed. Only ICBs allocated by a requestor are displayed.

Parameter Descriptions

The parameters for this command have the following meanings:

SERVER
Displays the status of the intertask communication servers. The servers for CICS temporary storage queues, for APPC/MVS, and for MQSeries which are started via the DSLNPTT and the batch APPC/MVS servers executed via program DSLNTSAB are shown. The parameter can be abbreviated to ‘S’.

TSQ
Displays the status of the allocated ICBs used by the intertask communication via CICS temporary storage queues. The parameter can be abbreviated to ‘T’.

APPC
Displays the status of the allocated ICBs used by the intertask communication via APPC/MVS. The parameter can be abbreviated to ‘A’.

MQI
Displays the status of the allocated ICBs used by the intertask communication via MQSeries. The parameter can be abbreviated to ‘M’.

srvname
Information for the specific server is displayed.

number
Is the number of a single ICB which status should be displayed.

ALL
The free and the allocated ICBs are displayed.

STOP
For a batch APPC/MVS server, a STOP request can be issued. The program DSLNTSAB retrieves the pending STOP request within an interval of one minute. The batch program DSLNTSAB is terminated.

Command Examples

The following section shows an example of how to enter the **dicb** command.

Example 1: To display the status of the allocated ICBs for the intertask communication enter the **dicb** command without parameters:

```
dicb
```

Example 2: To display the status of the intertask communication servers enter the **dicb** command with the server parameter:

```
dicb s
```

Examples of the Display from a DICB Command

"Displaying the ICBs (DICB)" on page 64 shows an example of the information that is returned when you enter the **dicb** command without parameters.

```
                        Operator Command Processing

> DICB
  DSL300I Display ICBs
   Server   Type Num Status
   CICSSRV  NTSQ 001 alloc DSLE ASC4 00084              0.023    0.011
   CICSSRV  NTSQ 003 alloc DSLX      00080              5.742    0.105
   CICSSRV  NTSQ 004 alloc DSLH TPR1 00073             17.893    0.041
   APPCSRV1 NTSA 001 alloc MRV$$SDI                   164.064    0.097
   APPCSRV2 NTSA 001 alloc MRV$$SDO                    67.338    0.345
   MQISRV1  NTSM 001 alloc DSLSDI MRC$$SDI 16:57:58     3.230    0.070







  080454   is the time of this display

Command =====>
PF 1=Help      2=Repeat    3=Return     4=DF         5=DU          6=DM Last
PF 7=Page -1   8=Page +1   9=Hardcopy  10=DP        11=DQ filled  12=DL
```

Figure 5. Displaying the Status of ICBs

Message DSL300I is displayed in response to the command, for details refer to *MERVA for ESA Messages and Codes*. The message contains the following information:

Server The name of the server serving the ICBs.

Type Dependent on the method of the intertask communication used, it can be:
- **NTSQ** - the server uses the temporary storage queue (TSQ).
- **NTSA** - the server uses APPC/MVS.
- **NTSM** – the server uses MQSeries.

Num Is the number of the ICB. The total number of ICBs for each nucleus server is determined by the ECB specification in the DSLNPTT definition.

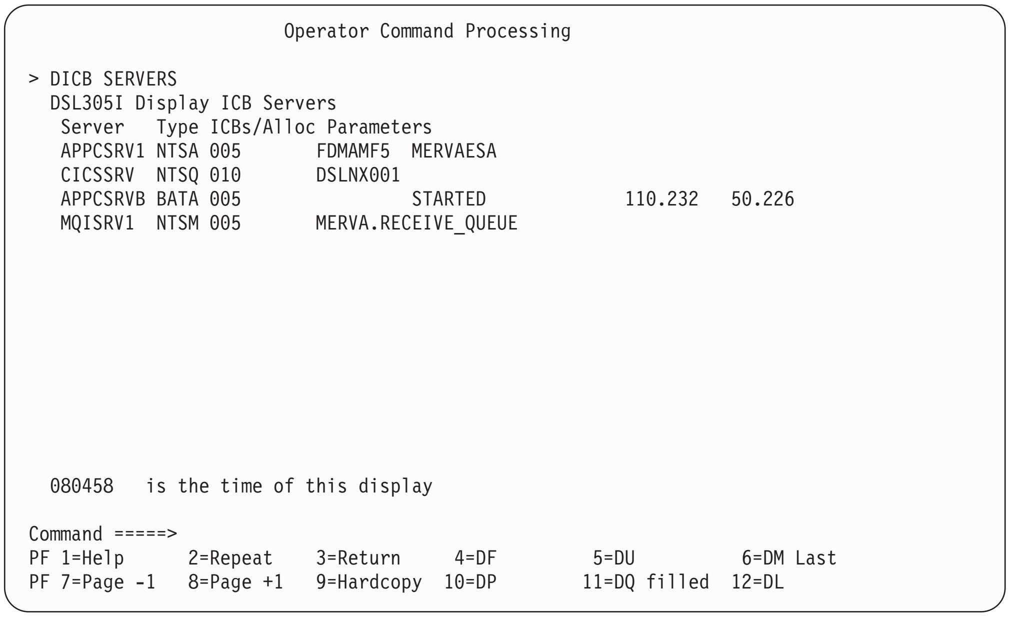
Status The status shows the information available for each ICB. For TSQ, this is usually the CICS transaction code, the terminal ID, and the CICS task number.

For APPC, this is usually the user ID, or the job name. In addition, the elapsed time in seconds is shown since the ICB was allocated and since the last request was processed. This information can be used to identify applications monopolizing MERVA ESA resources. The second timestamp value should be a small value, otherwise, the requestor might have terminated without freeing the ICB resource.

For MQSeries, this is usually the user ID, or the job name.

If other information such as user ID or program name is available in the ICB, this information is also shown.

Figure 6 shows an example of the information that is returned when you enter the **dicb SERVERS** command.



```
                      Operator Command Processing

> DICB SERVERS
  DSL305I Display ICB Servers
   Server   Type ICBs/Alloc Parameters
   APPCSRV1 NTSA 005       FDMAMF5  MERVAESA
   CICSSRV  NTSQ 010       DSLNX001
   APPCSRVB BATA 005                STARTED             110.232   50.226
   MQISRV1  NTSM 005       MERVA.RECEIVE_QUEUE











  080458   is the time of this display

Command =====>
PF 1=Help      2=Repeat    3=Return     4=DF         5=DU          6=DM Last
PF 7=Page -1   8=Page +1   9=Hardcopy  10=DP        11=DQ filled  12=DL
```

Figure 6. Displaying the Status of Intertask Communication Servers

Message DSL305I is displayed in response to the command, for details refer to *MERVA for ESA Messages and Codes*. The message contains the following information:

Server
The name of the server serving the ICBs.

Type
Dependent on the method of the intertask communication used, it can be:

- **NTSQ** - the server uses the temporary storage queue.
- **NTSA** - the server uses APPC/MVS.
- **BATA** - the server runs as a batch program (DSLNTSAB) and uses APPC/MVS.
- **NTSM** - the server uses MQSeries.

ICBs
The total number of ICBs for each server; it is determined by the ECB specification in the DSLNPTT definition or, for the batch program DSLNTSAB, controlled by an EXEC parameter.

Parameters
The parameters used by a server. For NTSQ, it is the temporary storage queue prefix.

For NTSA, it is the NOSCHED LU name and the TP name.

For BATA, the status is shown and the number of seconds since the server was started.

For NTSM, the name of the MQSeries queue where requests are received.

Displaying a Specific Request (DR)

Use the **dr** command to display the status of a specific request if you know the request number from a previous **drr** or **drqa** command or from the output written to the SYSPRINT data set after an **ntrace** command. The data is derived from the Request Control Element (DSLNRCE associated with the request number you have specified with the command.

Command Format

The format of the **dr** command is:

dr	*reqnum*

Parameter Descriptions

The parameter for this command has the following meaning:

reqnum
: The request number represents the static number of a Request Control Element (DSLNRCE within the request queue. You must enter a number of up to 6 digits. The value can be 1 to the maximum request queue size which can be obtained by issuing the **drqa** command. You can omit leading zeros.

Command Examples

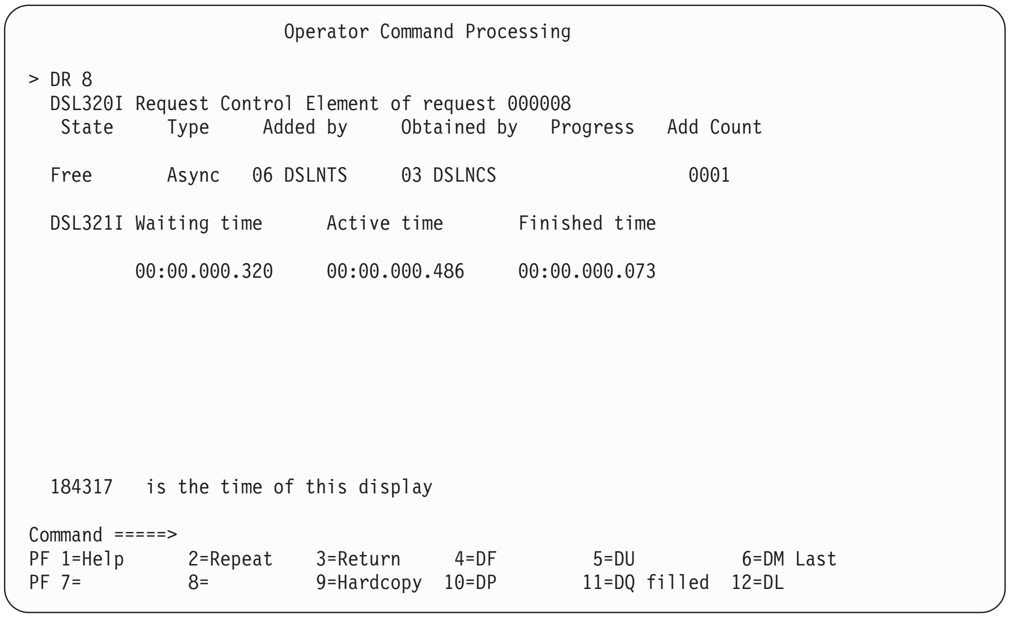
The following section shows an example of how to enter the **dr** command.

Example 1: To display request number 8:

```
dr 8
```

Example of the Display from a DR Command

"Displaying a Specific Request (DR)" on page 68 shows an example of the information that is returned when you enter the **dr 8** command.



```
                        Operator Command Processing

> DR 8
  DSL320I Request Control Element of request 000008
   State     Type     Added by     Obtained by   Progress   Add Count

  Free       Async   06 DSLNTS     03 DSLNCS                 0001

  DSL321I Waiting time      Active time       Finished time

          00:00.000.320     00:00.000.486     00:00.000.073








  184317   is the time of this display

Command =====>
PF 1=Help      2=Repeat    3=Return     4=DF         5=DU          6=DM Last
PF 7=          8=          9=Hardcopy  10=DP        11=DQ filled  12=DL
```

Figure 7. Displaying the Status of a Specific Request

Messages DSL320I and DSL321I are displayed in response to the command. They contain information in the Request Control Element (DSLNRCE, the representation of a queuing request. For details refer to *MERVA for ESA Messages and Codes*. If no request queuing environment is established, message DSL333I is responded.

Message DSL320I heads the first part of the specific request display. The first display part shows the following information:

State The state the request is currently in:

- **Waiting** - The request is in the waiting state. A service requested a service provided by another nucleus server. A new request was created and added to the request queue. This request is now waiting for being processed.
- **Active** - The request is in the active state. A nucleus server responsible to process a specific service has obtained this request from the request queue and is now processing it.
- **Finished** - The request is in the finished state. A nucleus server responsible to process a specific service has finished processing this request. The nucleus server which has added the request is notified.
- **Free** - The request is in the finished state. The nucleus server which has added the request is deleted from the request queue and inserted to the free element chain.

Type The service has added a subsequent service request and has defined one of the following processing types:

1. **Sync** - The nucleus server shell waits until all subsequent requests are finished.

2. **Async** - The nucleus server shell does not wait until any subsequent request is finished. In this case, it is up to the invoking service to wait until any subsequently added request has finished.

If the service request does not involve any nucleus server, this field is left blank.

Added by
This indicates the number and name of the nucleus server which has added this request. The number can range from 0 to 99 and is established for each nucleus server at the time it is initialized during startup. The name reflects the name given in the appropriate nucleus server table entry.

Obtained by
This indicates the number and name of the nucleus server which has obtained this request. The number can range from 0 to 99 and is established for each nucleus server at the time it is initialized during startup. The name reflects the name given in the appropriate nucleus server table entry.

Progress
Several steps are necessary from the time a request is obtained until its processing has finished:

- **RR** - Request Ready event processing. The nucleus server which is responsible to process a request is posted by the nucleus server which added the request to the request queue to signal that a request is ready for being processed. At the time the request was the most eligible, the posted nucleus server obtained it from the request queue and started processing it.
- **SP** - Service Processed event processing. The nucleus server which obtained the request has finished to process the service(s) described in it and indicated this event to its control program. The nucleus server which added the request is posted to signal that this request has been processed.
- **RP** - Request Processed event processing. The nucleus server which added a request may wait until this request is processed. If this event is recognized, the nucleus server deleted the added request and continued processing.

If an error occurred during any of these steps, this situation is indicated by showing **ERR** instead. You should examine the SYSPRINT file for an explaining trace message.

Add Count
A service which is currently executing in the nucleus server which obtained a request may subsequently add one or more requests. This count represents the number of service requests the nucleus server has subsequently created. If the service request does not involve any nucleus server, this field is left blank.

Message DSL322I heads the second part of the specific request display. It shows the time the service request spent in a state. The time is displayed in the format *MM:SS.ttt.mmm*: minutes:seconds.milliseconds.microseconds. The following times are displayed:

Waiting time
The time the request spent on the waiting queue before a nucleus server obtained it for processing.

Active time

The amount of time it took by the nucleus server to process the service described in the request. This includes the time another nucleus server needed to process a subsequently added synchronous request.

Note: There is no time reported for requests still waiting for being processed.

Finished time

The amount of time it took by the nucleus server to process the results of a subsequently added request, and to delete them from the request queue.

Note: There is no time reported for requests still processed by a nucleus server.

If the service request does not involve any nucleus server, these fields are left blank.

Displaying Relations to a Specific Request (DRR)

Use the **drr** command to display the relations to a specific request if you know the request number from a **drqa** command or from the output written to the SYSPRINT data set after an **ntrace** command. The data is derived from the Request Control Element (DSLNRCE associated with the request number you have specified with the command and the Server Control Block (DSLNSCB) of the involved nucleus server.

Command Format

The format of the **drr** command is:

drr	{[FIRST] *reqnum* }
	{ *reqnum* [FIRST]}

The first time you enter the **drr** command without the **FIRST** parameter, the first of a maximum of eight request relations are displayed. If you enter the same command again, the next eight request relations are displayed.

Parameter Descriptions

The parameters for this command have the following meaning:

FIRST

Displays the first eight request relations of the specified request number.

reqnum

The request number represents the static number of a Request Control Element (DSLNRCE within the request queue. You must enter a number of up to six digits. The value can be 1 to the maximum request queue size which can be obtained by issuing the **drqa** command. You can omit leading zeros.

Command Examples

The following section shows an example of how to enter the **drr** command.

Example 1: To display relations of request number 8:

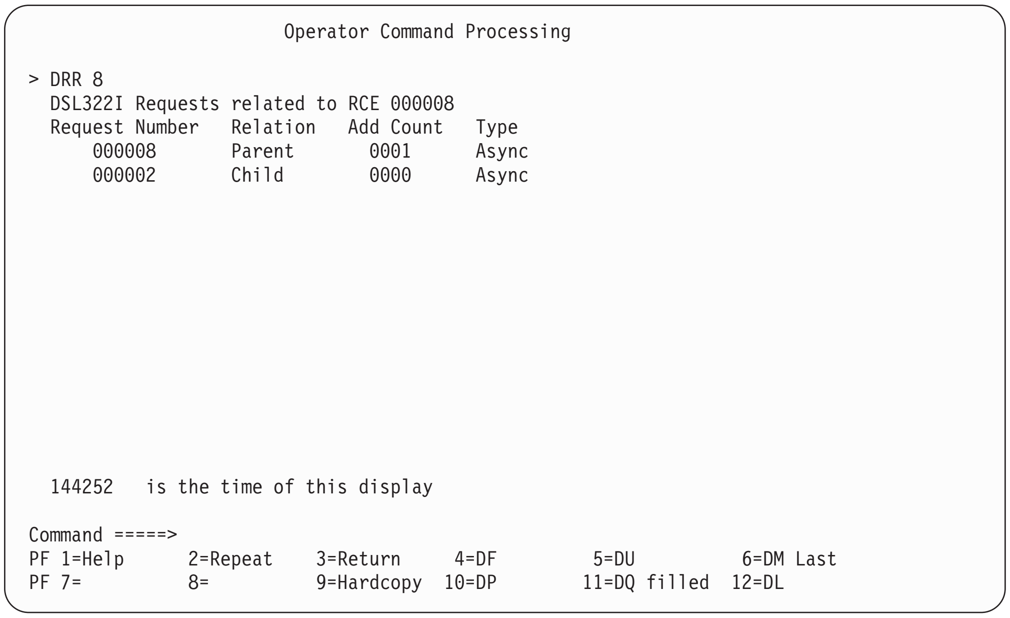
```
drr 8
```

Example 2: To display first 8 relations of request number 8:

```
drr first,8
```

Example of the Display from a DRR Command

"Displaying Relations to a Specific Request (DRR)" on page 71 shows an example of the information that is returned when you enter the **drr 8** command.



```
                          Operator Command Processing

> DRR 8
  DSL322I Requests related to RCE 000008
  Request Number   Relation   Add Count   Type
      000008       Parent       0001      Async
      000002       Child        0000      Async













  144252   is the time of this display

Command =====>
PF 1=Help      2=Repeat    3=Return     4=DF         5=DU          6=DM Last
PF 7=          8=          9=Hardcopy  10=DP        11=DQ filled  12=DL
```

Figure 8. Displaying the Relations of a Specific Request

If no request queuing environment is established, message DSL333I is responded. Message DSL322I is displayed in response to the command. For details refer to *MERVA for ESA Messages and Codes*. It heads the related requests display which is organized in table form sorted by the request number in ascending sequence, and shows information in the Request Control Element (DSLNRCE, the representation of a queuing request, as follows:

Request Number
The numbers of the related requests. This number can range from 1 to 999999 and is controlled by the request queue handler. The numbers in this column are sorted in ascending sequence.

Relation
The relations can be:

- **Current** - This indicates that the specified request has no child (is no parent) or has no parent (is no child). This can happen if the relation chain is broken due to a request control element reuse.
- **Parent** - This indicates that the indicated request is parent to one or more children, the service has added one or more subsequent service requests. If a service request is a parent and is of a synchronous type, it waits until all subsequent child requests are finished before it is allowed to continue processing.
- **Child** - This indicates that the indicated request is a child, another service has added this subsequent request.
- **Brother** - This indicates that the indicated request is a child, but since the parent request is of a synchronous type and has more than one child, all younger children have a brother relation to the oldest child.

Add Count	A service represented by the indicated request number may have created one or more subsequent requests and added them onto the request queue. If the service request does not involve any nucleus server, this field is left blank.
Type	If a service adds a subsequent request, two processing types can be specified: 1. **Sync** - The nucleus server waits until all subsequently added requests are finished. 2. **Async** - The nucleus server does not wait until any subsequently added request is finished. If the service request does not involve any nucleus server, this field is left blank.

Displaying Administration Data for request queue (RQ) and Nucleus Servers (DRQA)

Use the **drqa** command to display the current request queue and nucleus server states.

Command Format

The format of the **drqa** command can be any of the following:

drqa	[{ *srvnum* }] [{ *srvname* }] [{ FIRST [,*srvnum*]}] [{ FIRST [,*srvname*]}] [{ *srvnum* [,FIRST]}] [{ *srvname* [,FIRST]}]

The first time you enter the **drqa** command without the **FIRST** parameter, message DSL323I is issued followed by the request queue counts, and message DSL324I followed by a maximum of four lines with general nucleus server statistics. If you enter the same command again, message DSL324I is issued followed by a maximum of eight lines with the next general nucleus server statistics, if available.

Parameter Descriptions

The parameters for this command have the following meaning:

FIRST
Displays statistics of the first of a maximum of four nucleus servers. The nucleus servers are sorted in ascending sequence by the nucleus server number assigned to it at MERVA startup time. If accompanied with the *srvnum* or *srvname*, the statistics of the next of a maximum of four nucleus servers are displayed.

srvnum
This number is assigned to a nucleus server at MERVA ESA startup time.

srvname
This name is assigned to a nucleus server in the nucleus server table.

Command Examples

The following section shows an example of how to enter the **drqa** command.

Example 1: To display the request queue and nucleus server statistics starting with the lowest nucleus server number:

```
drqa
```

Note that the lowest nucleus server number is 0, which is reserved for the DSLNUC maintask.

Example 2: To display the statistics of the next nucleus servers in ascending sequence:

```
drqa
```

Example 3: To display the statistics of the next nucleus servers in ascending sequence starting with nucleus server number 7:

```
drqa first,7
```

Example 4: To display the statistics of the next nucleus servers in ascending sequence starting with nucleus server name DSLQMGT:

```
drqa first,dslqmgt
```

Example 5: To display the statistics of nucleus server 5:

```
drqa 5
```

Example 6: To display the statistics of nucleus server DSLJRNP:

```
drqa dsljrnp
```

Example of the Display from a DRQA Command

"Displaying Administration Data for request queue (RQ) and Nucleus Servers (DRQA)" on page 73 shows an example of the information that is returned when you enter the **drqa 3** command.

```
                          Operator Command Processing

> DRQA
  DSL323I Current Request Queue Counts
  Tot. Free   Tot. Wait.   Tot. Act.   Tot. Fin.   Queue Size   Servers
   000008      000000       000002      000000       000010      0017

  DSL324I General Nucleus Server Statistics
  Number      Name      Request    # Wait    State
   0000     DSLNUC                  0000     Busy
   0001     DSLJRNP                 0000     Idle
   0002     DSLQMGT                 0000     Idle
   0003     DSLNCS      000008      0000     Busy






  182202   is the time of this display

Command =====>
PF 1=Help      2=Repeat    3=Return     4=DF         5=DU          6=DM Last
PF 7=          8=          9=Hardcopy  10=DP        11=DQ filled  12=DL
```

Figure 9. Displaying Statistics of the Request Queue and the First 4 Nucleus Servers

Messages DSL323I and DSL324I are displayed in response to the command. They contain information derived from internal control blocks of the respective nucleus server. For details refer to *MERVA for ESA Messages and Codes*. If no request queuing environment is established, message DSL333I is responded. Message DSL323I heads the general request queue counts and shows the following information:

Tot. Free	Represents the total number of request queue elements which are currently unused. This is also called to be the free element pool from which the request queue handler fetches an element and assigns it to the request queue of a nucleus server to perform the ADD queuing function.
Tot. Wait.	Represents the total number of request queue elements occupied by requests currently waiting for being processed.
Tot Act.	Represents the total number of request queue elements occupied by requests currently being processed.
Tot. Fin.	Represents the total number of request queue elements occupied by requests which have been finished processing. They are, however, not yet freed by the nucleus servers.
Queue Size	This number represents the total request queue size from which the request queues of all nucleus servers are served.
Servers	This number includes the DSLNUC maintask and the nucleus servers running as separate tasks in parallel. They all share the same free element pool from which the requests are built.

Message DSL324I heads the general nucleus server statistics. The general nucleus server statistics are organized in table form sorted by nucleus server numbers in ascending sequence, and shows the following information:

Number
: This column shows the nucleus server number in ascending sequence, starting with the default of 0000, the one specified, or the one assigned to the nucleus server name specified. The nucleus server number is assigned to a nucleus server at the time MERVA ESA is started.

Name
: This column shows the nucleus server name associated with the nucleus server number. The nucleus server name is specified in the Nucleus Server Table (DSLNSVT).

Request
: The number in this column shows the number of the request which is currently being processed by this nucleus server.

# Wait
: The number in this column shows the number of requests currently waiting on the request queue for being processed by this nucleus server.

State
: A nucleus server can have one of the following states:

1. **Busy** - The nucleus server currently processes a service. This may be a service described in the obtained request with the number indicated, or a previously started program.
2. **Idle** - The nucleus server does not currently process any request.

Activate or Deactivate a Debugging Trace (NTRACE)

Use the **ntrace** command to activate or deactivate a debugging trace.

Command Format

The format of the **ntrace** command is:

ntrace | **ntrc**	{*parm1*,*parm2*,*parm3*,*parm4*}

Parameter Descriptions

The parameters for this command have the following meaning:

parm1
: This parameter specifies the nucleus server to be traced. A specific area can be specified by entering the nucleus server number or the nucleus server name.

  srvnum
  : This maximum 4-digit number is assigned to a nucleus server at MERVA ESA startup time. By issuing the **drqa** command, you can determine the nucleus server number you want to be traced. You can omit leading zeros.

  srvname
  : The nucleus server name consists of up to 8 alphanumeric characters. By issuing the **drqa** command, you can determine the nucleus server name you want to be traced.

  all This specification means that all nucleus servers are affected in the levels specified in *parm2*, the depth specified in *parm3*, and the action specified in *parm4*.

parm2
: This parameter specifies the level to be traced. The levels to be specified can be:

  shel
  : Specifies to trace the nucleus server shell control module for the non-CICS environment in the nucleus server specified in *parm1*.

  shec
  : Specifies to trace the nucleus server shell control module for the CICS environment in the nucleus server specified in *parm1*.

  rrp
  : Specifies to trace a nucleus servers request ready processing in the nucleus server specified in *parm1*.

  spp
  : Specifies to trace a nucleus servers service processed processing in the nucleus server specified in *parm1*.

  rpp
  : Specifies to trace a nucleus servers request postprocessing in the nucleus server specified in *parm1*.

  ppp
  : Specifies to trace a nucleus servers posted program processing in the nucleus server specified in *parm1*.

  rqh
  : Specifies to trace the common request queue handler processing.

  nul | 0
  : Specifies to skip setting of a trace level and leave it unchanged.

  all This specification means that all levels are affected in the nucleus servers specified in *parm1*, the depth specified in *parm3*, and the action specified in *parm4*.

parm3
: This parameter specifies the depth to be traced. The depths to be specified can be:

base | basic
Specifies to trace the general flow and common data of the nucleus server specified in *parm1* in the levels specified in *parm2*.

event | evt
Specifies to trace especially the ECB posting and event list processing of the nucleus server specified in *parm1* in the levels specified in *parm2*.

request | req
Specifies to trace especially the Request Control Element (DSLNRCE contents and the request chaining of the nucleus server specified in *parm1* in the levels specified in *parm2*.

disp
Specifies to trace the common request display processing.

ntrace
Specifies to trace the nucleus server shell debugging trace processing.

nul | 0
Specifies to skip setting of a trace depth and leave it unchanged.

all This specification means that all depths are affected in the nucleus servers specified in *parm1*, the levels specified in *parm2*, and the action specified in *parm4*.

parm4
This parameter specifies the trace action. The action to be specified can be:

on Specifies to switch on the trace in the nucleus servers specified in *parm1*, the levels specified in *parm2*, and the depths specified in *parm3*.

off
Specifies to switch off the trace in the nucleus servers specified in *parm1*, the levels specified in *parm2*, and the depths specified in *parm3*.

Command Examples

The following are examples of how to enter the **ntrace** command. You can use the abbreviation **ntrc** instead.

Example 1: To trace the request ready processing general flow in the nucleus server with name DSLJRNP in a non-CICS environment:

```
ntrace all,all,all,off
ntrace dsljrnp,shel,base,on
ntrace dsljrnp,rrp,base,on
```

Example 2: To trace the posted program processing general flow in the nucleus server with name DSLQMGT in a CICS environment:

```
ntrace all,all,all,off
ntrace dslqmgt,shec,base,on
ntrace dslqmgt,ppp,base,on
```

Example 3: To trace the event processing in the nucleus server with name DSLNCS in any environment:

```
ntrace all,all,all,off
ntrace dslncs,all,base,on
ntrace dslncs,all,evt,on
```

Example 4: To trace the request processing in the nucleus server with name DSLNUSR in any environment:

```
ntrace all,all,all,off
ntrace dslnusr,all,base,on
ntrace dslnusr,all,req,on
```

Example 5: To add the event trace and remove the request trace processing in the nucleus server with name DSLNUSR in any environment:

```
ntrace dslnusr,all,evt,on
ntrace dslnusr,all,req,off
```

Example 6: To trace the request queue handler processing:

```
ntrace all,all,all,off
ntrace all,rqh,all,on
```

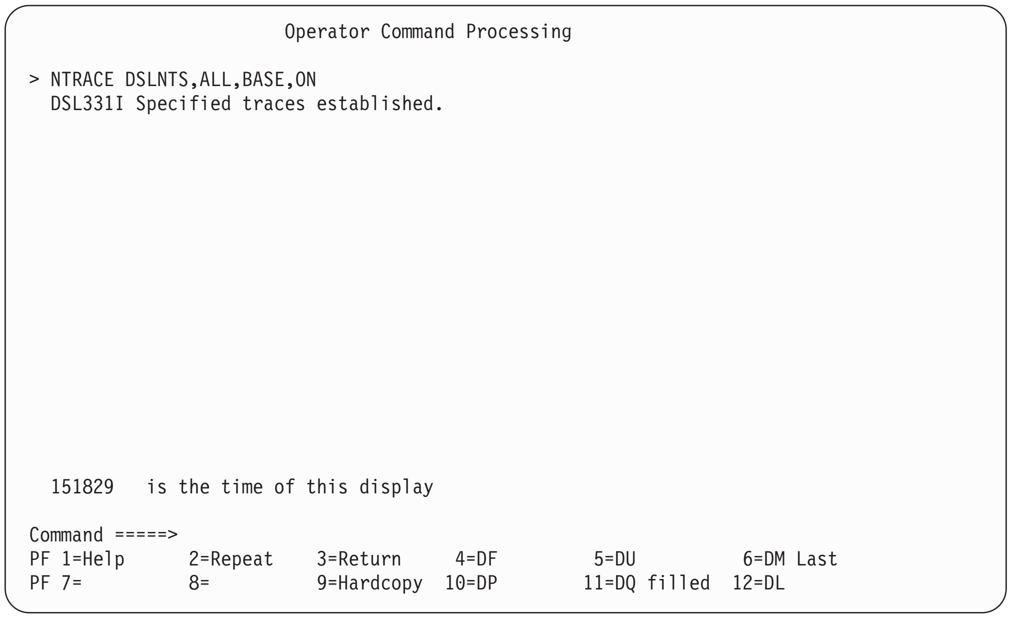
Example 7: To trace all nucleus servers and levels with all depths:

```
ntrace all,all,all,on
```

Note that this trace produces a huge amount of data and slows down the performance significantly.

Example of the Display from an NTRACE Command

Figure 10 shows an example of the information that is returned when you enter the **ntrace 3** command.



```
                      Operator Command Processing

> NTRACE DSLNTS,ALL,BASE,ON
  DSL331I Specified traces established.













  151829   is the time of this display

Command =====>
PF 1=Help      2=Repeat    3=Return     4=DF         5=DU          6=DM Last
PF 7=          8=          9=Hardcopy  10=DP        11=DQ filled  12=DL
```

Figure 10. Establish a Specific Trace

Message DSL331I is displayed in response to the command if the traces can be successfully established. For details regarding the debugging trace facility refer to "Debugging Traces for Nucleus Server Components" on page 56. If no nucleus server shell and request queuing environment is established, message DSL334I is responded.

Query or Change X.25 Trace Flags (XTRACE)

You use the **xtrace** command to query or change MERVA ESA X.25 trace flags. Currently there are 10 trace flags defined for debugging purposes. Only X.25 protocol lines are supported by this command. You should only use this command when IBM has instructed you to do so.

Command Format

The format of the **xtrace** command is:

xtrace	*line* [,trace flag no. [,ON | OFF]]

Parameter Description

The parameters for this command have the following meanings:

line
: Denotes the number of the line whose trace flags you want to query or change. The line must be an X.25 protocol line. The value entered must be a number from 1 to the maximum number of lines used in the system. This actual number of lines used in the system depends on the customization of the SWIFT Link, which allows a maximum of 30 lines. The number refers to the name of a line definition module, for example, DWSLIN1 for line 1, DWSLIN5 for line 5. If the line is not initialized or is not an X.25 line, the command is rejected.

  If you enter the **xtrace** command only with the *line* parameter, the current setting of this line is displayed.

trace flag no.
: Denotes the number of the trace flag whose setting you want to change. The value entered must be a number from 1 to 10.

ON | OFF
: Indicates whether you want to set the specified trace flag on or off.

Command Examples

This section shows some examples of how to enter the **xtrace** command.

Example 1: Enter the following command to display the status of all trace flags of line no. 3:

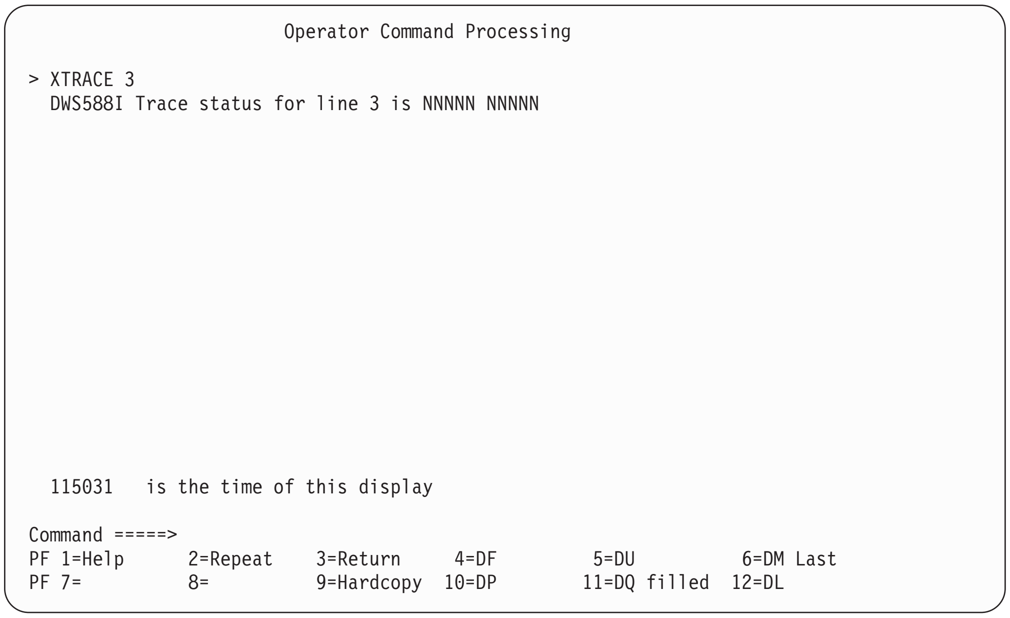
```
xtrace 3
```

Example 2: Enter the following command to change the status of the trace flag no. 5 for line no. 7 to ON:

```
xtrace 7,5,ON
```

Example of the Display from an XTRACE Command

Figure 11 shows an example of a panel displayed in response to an **xtrace***line* command.



```
                        Operator Command Processing

> XTRACE 3
  DWS588I Trace status for line 3 is NNNNN NNNNN




















  115031   is the time of this display

Command =====>
PF 1=Help      2=Repeat    3=Return     4=DF         5=DU          6=DM Last
PF 7=          8=          9=Hardcopy  10=DP        11=DQ filled  12=DL
```

Figure 11. Displaying the Status of MERVA ESA Trace Flags

Service Aids for Intertask Communication

The DICB command can be used to display status information for the intertask communication methods via CICS temporary storage queues, via APPC/MVS, and via MQSeries. Refer to “Displaying the ICBs (DICB)” on page 64 for details.

A MERVA ESA requestor program using the intertask communication method via APPC/MVS issues a dump in case there are problems with APPC/MVS requests. Refer to “Dump produced by intertask communication via APPC/MVS” on page 63 for details on how to analyze the dump.

A common problem which can occur when using the traditional MERVA ESA intraregion and interregion communication methods is that an allocated ICB is not released by the requestor application. Such problems can be analyzed using the methods described below.

To display the status of the control blocks of the inter- and intraregion communication (ICBs) the command **show DSL0NIC** can be used in the CMD or MSC function. The panel DSL0NIC extracts the ICB information from the system and displays the current status. Only the local ICBs can be displayed. The intraregion ICBs are used on a MERVA ESA running under CICS only, for a MERVA ESA under IMS there are no intraregion ICBs. Only the interregion ICBs of MERVA ESA running on an MVS system can be displayed.

Example of the Display from a SHOW DSL0NIC Command

Figure 12 shows an example of a panel displayed in response to a **show DSL0NIC** command.

```
                  MERVA ESA Intertask Communication

Interregion Communication
      Job MRC$CS41   Startup Time 090326   MERVA ID        @MRC
      SVC 214        CVT Offset   088      Subsystem Entry NO

      ICBs:  00005 available   00001 allocated   00004 free
      00001  allocated ICB   MRC$$SDI Requestor is active
      00002  --- free ---    MRC$$SDY Requestor's ASCB not found
      00003
      00004
      00005

------------------------------------------------------------------------
Intraregion Communication
      Job            Startup Time 090326   MERVA ID        @MRC

      ICBs:  00025 available   00003 allocated   00022 free
      00001  Requestor's Trace Table: DSLEUD  6206    TACSTART
      00002  Requestor's Trace Table: DSLCXT  L3CXT   09:04:29

Command =====> SHOW DSL0NIC
PF 1=Help      2=Retrieve  3=End        4=          5=          6=
PF 7=Page -1   8=Page +1   9=Hardcopy  10=         11=         12=
```

Figure 12. Displaying the Status of the Inter- and Intraregion ICBs

The display consists of two parts, the interregion ICBs and, under CICS, the intraregion ICBs. The information on the panel shows:

Job	The jobname of the MERVA ESA batch program or CICS MVS job.
Startup Time	The time MERVA ESA nucleus was started.
MERVA ID	The MERVA ESA identification as defined in DSLPRM.
SVC	The SVC number.
CVT Offset	The offset used in the CVT user extension.
Subsystem Entry	If a subsystem entry is used as interregion communication anchor, YES is shown.
ICBs	Shows the total number of ICBs defined, which is the number of ECBs customized in the DSLNPTT for the intertask communication method. The number of allocated and free ICBs is also shown.

The status of each individual ICB is indicated in an information line.

For interregion communication, the number of the ICB is followed by the indication whether the ICB is allocated or free. The job name of the requestor is shown. When the job is still running it is indicated as active, otherwise the ASCB is not found.

If an ICB is allocated, but the ASCB is not found, the ICB was not freed by the requestor program.

For intraregion communication, the display function may be able to extract additional information about the transaction running. This is done by analyzing the MERVA ESA program trace table of the requestor. The program name, the terminal ID, the function, or the start time of the transaction is shown. When the information is not available, for example, the trace table is not accessible, an error indication is given.

The status of the interregion ICBs of MVS can be displayed using the ICB utility program DSLICBUT.

JCL to Run the DSLICBUT Utility

```
//........ JOB ........
//ICBUT    EXEC PGM=DSLICBUT     ,PARM='S(xxxx)'
//*                              ,PARM='DUMP'
//STEPLIB  DD  DSN=MERVA.SDSLLODB,DISP=SHR
//SYSPRINT DD  SYSOUT=*
```

Figure 13. Running the ICB Report Utility DSLICBUT

The output created by the program DSLICBUT is printed on the SYSPRINT data set.

If you use a subsystem entry for the interregion communication, the subsystem name xxxx must be specified as ‘S(xxxx)’ on the EXEC statement. The subsystem name for MERVA ESA is specified with the MERVA identifier (parameter DSLID in the customization parameter DSLPRM).

Specifying the parameter ‘DUMP’ on the EXEC statement prints the control blocks also in dump format.

Example of a Report from the DSLICBUT Utility

Figure 14 shows an example of a report created by the DSLICBUT program.

```
MERVA ESA FOR MVS  - INTERREGION COMMUNICATION
SUBSYSTEM ENTRY
 --> Address 00BA9338 - Length 0024 <--
BA9338  E2E2C3E3 00BA935C D4C5D9E5 00000000 *SSCT..l*MERV....* 0000
BA9348  00000000 00000000 00BA6200 00000000 *................* 0010
BA9358  00000000                            *....            * 0020

OFFSET 088 : JOB(MRV$CS41) TIME(090326) ID(@MRV) REGS(005) VERSION(V3)
             START GMT(TODAY  064616) JOB(RUNNING,CPU=00112.757)
ICB ALLOCATED  JOB(MRV$$SDI) BUFSIZE(0000017408)
             START GMT(TODAY  083924) JOB(RUNNING,CPU=00002.374)
ICB FREE       JOB(MRV$$SDY) BUFSIZE(0000017408)
             START GMT(TODAY  083924) JOB(TERMINATED           )
             ALLOCATED ICBS(002)      TOTAL BUFFER STORAGE(0000034816)

OFFSET 120 : JOB(XRP$C410) TIME(085324) ID(XRPH) REGS(005) VERSION(V3)
             START GMT(TODAY  065217) JOB(RUNNING,CPU=00050.998)
             ALLOCATED ICBS(000)      TOTAL BUFFER STORAGE(0000000000)
```

Figure 14. Report from DSLICBUT Utility

The information in the report consists of several blocks.

If a subsystem entry is used for MERVA ESA, the subsystem control block is printed in dump format. The control block contains the anchor address of an ICA (intertask communication area).

The CVTUSER extension is examined. If the user extension is used by MERVA ESA, the ICA and all ICBs for each offset in the CVT user extension are printed. When multiple MERVA ESA are running on the same MVS, each MERVA ESA uses its own ICA.

For each ICA, the offset, the job name, the local start time of MERVA ESA, the MERVA ESA identification as defined in the DSLPRM, the number of ECBs customized in the DSLNPTT entry for TYPE=INTER, and the MERVA ESA version area are printed. MERVA ESA V4 uses the same layout as MERVA ESA V3, therefore the version indication is always V3.

The next line indicates the start time of the job in GMT, and if the job is running, the CPU time used in seconds.

For each ICB using this ICA, the requestor's job name and the buffer size used by this requestor is shown. The status of an ICB is either ALLOCATED or FREE. The next line indicates the start time of the job in GMT, and if the job is running, the CPU time used so far.

If an ICB is allocated, the job should be running. If an ICB is allocated, but the job is already terminated, this is an indication of an error situation.

The last line for each ICA shows the total number of ICBs allocated and the total buffer storage used in the ECSA for this ICA.

Service Aids for Other Components

Depending on the operating system and the data communication system used, several additional service aids are available. Some examples are:

- With the Generalized Trace Facility (GTF) you can trace the I/O operations and buffer contents for the line to the SWIFT or telex network under MVS. For the MERVA ESA network links, the I/O trace is likely to be the most useful service aid to determine performance and communication-line problems.
- The System Debugging Aid (SDAID) provides the same trace services under VSE.
- CICS provides various service aids that are useful in the process of determining the cause of possible problems with MERVA ESA and the network links. You can find further information on these in the *CICS/VSE Problem Determination Guide*.
- IMS also provides trace capabilities for tracing internal IMS events. The IMS operator command **/TRACE** is used to invoke this service. You can find further information on these in the *IMS/ESA V4 Operator's Reference*.

Chapter 4. Abnormal Events

In this context, an abnormal event is something that interrupts or affects the operation of your MERVA ESA installation.

Startup Problems

If an error occurs when starting MERVA ESA or the network links, the program issues an error message to the operator. The message contains the function and, where applicable, the return and reason codes of the function request in error.

Errors during startup are likely to be caused by improper installation or customizing. For installation errors check that:

- Tables (for example DSLPRM, DWSPRM, ENLPRM, MERVA Link Partner Table, Message Type Table, Function Table, and Field Definition Table) are installed correctly (no assembly errors, no MNOTEs, and the correct names in the customizing parameters).
- The transaction codes and logical terminal names specified in the Function Table and the MERVA Link Partner Table (if applicable) are also generated with IMS, or specified in the CICS definitions, respectively. This is especially important for the name of the send transaction and the name of the receive transaction in the Telex Link and the MERVA Link.

For DSLNUC errors check that:

- The routing modules are correctly installed.
- All queues specified in the MERVA ESA generation parameters and in the routing modules are also defined in the Function Table.

End-User Errors

The following end-user errors can occur:

- If the message **DSL1004 MERVA is not ready** is issued, a user has tried to sign on before MERVA ESA is started.
- If the message **DSL1010 Not enough main storage available** is displayed, increase the region size.

Performance Problems

To avoid performance problems you should define the highest priority for the MERVA ESA nucleus DSLNUC:

- For CICS define the highest priority in a CICS region or partition.
- For IMS define the highest priority after the IMS control region. It must be higher than the IMS MPP regions.

In the DSLNPTT definition of MERVA ESA, the program DWSDGPA (SWIFT Link) should have a higher scheduling priority than other programs.

Notes for CICS:

1. You must ensure that the ICV parameter of the CICS system initialization table (DFHSIT) has a low value: for example, ICV=100 or ICV=250.

2. You must ensure that the MXT parameter of DFHSIT is high enough (see the MNOTE in the assembly listing of DSLNPTT).

Severe Errors

In the case of the following severe errors, MERVA ESA and the network links terminate with a dump:

- Queue management errors
- Journal errors
- Message Format Service errors (return code > 4), which always cause a signoff or return to function selection, but which do not always produce a dump
- TOF Supervisor errors (return code > 4)
- Timer errors (DWSDGPA, DSLISYNP, and DSLCNTP)
- General Service errors (DSLSRVP)
- Interregion communication errors (DSLEUD only signs off and terminates)
- IMS errors
- Authentication errors (DWSAUTP for the SWIFT Link only)
- End-User Driver errors
- MERVA ESA SPA File error (IMS only).

Some of these errors are described in "What Action to Take".

When a dump is available, the parameter lists appear in the storage of the module in error. When the queue data set or the journal data set is full, a MERVA ESA error message is displayed on the operating-system console. For other queue-management errors, the definition of the function table and routing modules should be checked. If the message counter log data set is unavailable, MERVA ESA terminates with an error message. For the Message Format Service (MFS) and TOF errors, user changes for Message Control Blocks (MCBs, especially line definitions), the Field Definition Table (DSLFDTT), and MFS exit specifications should be examined. When a **TOF too small** or **TOF full** error is indicated, the TOF size specification in the DSLPRM should be checked and corrected.

What Action to Take

The following gives a brief description of the action you can take if certain abnormal events occur.

If a Transaction Is in a Wait State

If MERVA ESA terminates while a transaction is running, the transaction may remain in a wait state because intertask communication no longer functions. The transaction must be canceled using CICS or IMS commands. The processing of the transaction can be resumed using the MERVA ESA operator command **sf** (start function).

If a Batch Program Is in a Wait State

If MERVA ESA terminates while a MERVA ESA program is running, the program may remain in a wait state because intertask communication no longer functions. The batch program must be canceled using operating system commands.

After restarting MERVA ESA, the batch program can be started again. No data is lost because the batch programs determine whether the transfer of data was completed or not. If the transfer was not completed, the batch program automatically carries out restart processing.

If the Queue Data Set Is Full

When a MERVA ESA component detects the **queue data set full** condition, it immediately cancels its network connection (SWIFT Link, Telex Link, MERVA Link). You can do one of the following:

- Ask users to process the messages in the queues.
- Start the hardcopy printer program.
- Start DSLSDY to print and empty a queue.
- Start DSLSDO to move messages to a sequential data set and empty the queues. It should be taken into consideration that DSLSDO needs to put one (small) restart message into the queue before the processing can start. It might be necessary to delete one message before DSLSDO can be started.

If this condition appears frequently and queue management using VSAM is used, the size of the queue data set can be increased using the MERVA ESA utility DSLQDSUT.

During each MERVA ESA startup, queue management automatically determines if a queue-management restart must be performed. When the queue data set can be read physically, queue management maintains the integrity of the data in the queue data set. When queue-data-set duplication is implemented and one of the data sets is corrupted, it can be re-created by copying the uncorrupted data set.

If the Journal Data Sets Are Full

Whenever the journal data set A is full, the MERVA ESA journal program switches automatically to journal data set B. The switch is indicated by the message:

```
DSL040I Switched from journal A to journal B
```

Depending on the size of the journal data set B, MERVA ESA should be terminated when possible, so that you can print and clear the journal data sets before the next MERVA ESA startup. If journal data set B becomes full, MERVA ESA terminates.

MERVA ESA V4.1 provides a command for manually switching the journal data sets and the possibility to reset a journal data set before it is used. This allows for a more flexible operation of the journal data sets and enables the continued operation of MERVA ESA. Provided that the archive operation is set up properly, the journal-full condition can be completely avoided. Refer to the *MERVA for ESA Operations Guide* for details about the journal operation in an 7x24 environment.

Problems with the Network Links

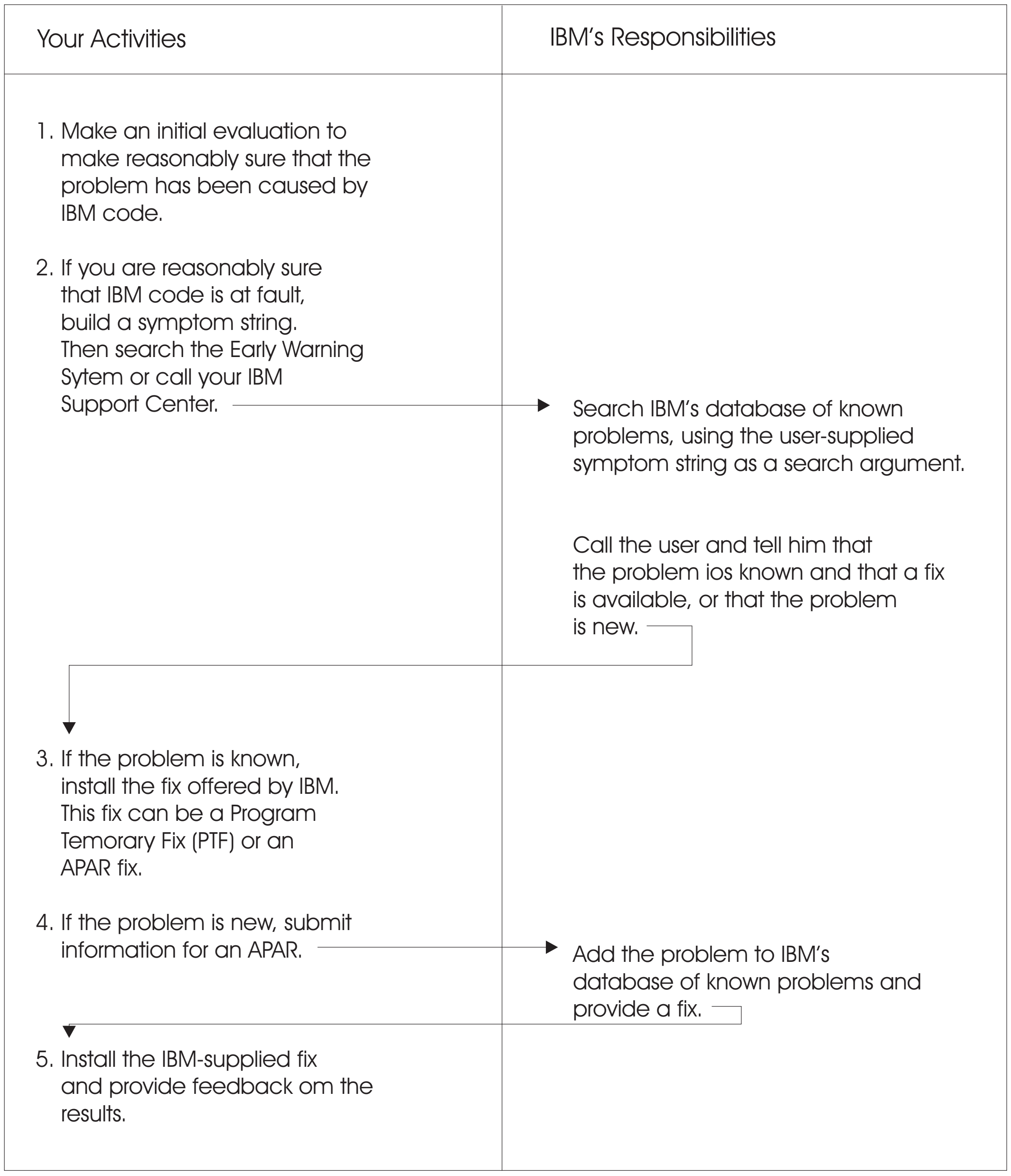
Whenever an abnormal situation arises, the SWIFT Link or Telex Link issues an error message to the MERVA ESA operators. MERVA Link error codes are shown in the MERVA Link application control panel of the MSC function. The MERVA ESA operator should refer to *MERVA for ESA Messages and Codes*, and take the action indicated there.

Problems with the SWIFT Connection

If a problem arises with the connection to the SWIFT network, refer to the *S.W.I.F.T. User Handbook* or contact the SWIFT support center to solve the problem. Before doing this, ensure that there is no problem with the modem or the connection to the telephone network.

Chapter 5. Diagnosing and Reporting Program Failures

This chapter explains the steps to be taken if you have a problem. These steps are summarized in Figure 15, which shows the actions you and IBM should take to solve the problem.



Your Activities

IBM's Responsibilities

1. Make an initial evaluation to make reasonably sure that the problem has been caused by IBM code.
2. If you are reasonably sure that IBM code is at fault, build a symptom string. Then search the Early Warning Sytem or call your IBM Support Center.

Search IBM's database of known problems, using the user-supplied symptom string as a search argument.

Call the user and tell him that the problem ios known and that a fix is available, or that the problem is new.

3. If the problem is known, install the fix offered by IBM. This fix can be a Program Temorary Fix (PTF) or an APAR fix.
4. If the problem is new, submit information for an APAR.

Add the problem to IBM's database of known problems and provide a fix.

5. Install the IBM-supplied fix and provide feedback om the results.

Figure 15. An Overview of IBM's Service Concept

Carrying Out the Initial Evaluation

When MERVA ESA does not function correctly, a numbered diagnostic message is normally issued. The three letters in the message number indicate the component of MERVA ESA that issued the message:

DSL	The Base Functions
DWS	The SWIFT Link
ENL	The Telex Link
EKA	The MERVA Link or the MERVA-to-MERVA Financial Message Transfer/ESA
IMR	The MERVA ESA Traffic Reconciliation.

When carrying out the initial evaluation, the first step is to read the explanation in *MERVA for ESA Messages and Codes* for the messages issued, and to take the recommended action. In addition, if the system runs under an MVS/ESA operating system, the *MVS/ESA Problem Determination Guide* can be useful: in particular, the CICS or IMS trace and diagnostic aids. If the system runs under a VSE/ESA operating system, the *IBM VSE/ESA Guide for Solving Problems* can be useful.

If the same or a related error occurs after you have taken the recommended action, it is possible that the error was caused by an error in IBM-supplied code. Use the following procedure to establish whether this is the case:

1. Go through the checklist shown in Table 1. This will help you to determine whether the problem might have been caused by IBM code. For example, if you have applied an IBM Program Temporary Fix (PTF) to MERVA ESA since the last time you ran the program successfully, and you have applied the fix correctly, IBM code is probably at fault, and you should call IBM.
2. Table 2 on page 92 lists some possible problems in the MERVA ESA environment, and the recommended actions for solving them. It is impossible to provide detailed procedures that will diagnose all possible failures. However, this figure gives you a guideline that helps you to decide if the failure was caused by IBM code, operator error, or generation-parameter errors of MERVA ESA or its network links.
3. If your initial evaluation points to an error in IBM-supplied code, see “Building a Symptom String” on page 92, which describes the information IBM needs to help you to solve the problem.

Problems that occur during a run of MERVA ESA can arise from MERVA ESA code, or from errors in the operating system or the control program environment. When you begin to diagnose a problem, you must consider all these possibilities. Once you are reasonably sure that the problem is in MERVA ESA, you can start to build your symptom string.

Table 1 on page 91 is the checklist to be used during the initial evaluation of the problem.

Table 1. Initial Evaluation Checklist

Item to Be Checked	Possible Cause if YES	Your Action if YES
Release of Operating System		
Did the program run previously under the same release of the operating system?		Continue with checklist.
Did the program run previously under the same release of the data communication system (CICS or IMS)?		Continue with checklist.
Have there been any changes to the operating system since the previous successful runs?	Bad fix	Continue with checklist.
Have there been any changes to the data communication system since the previous successful runs?	Bad fix	Continue with checklist.
System Hardware Configuration		
Was the program run on a system with a configuration different from the one that existed at the time of the previous successful run, for example, different network addresses?	User error	Ensure that the system software correctly reflects the configuration change. Check the job control statements.
Job Control or MERVA ESA Generation Parameters		
Have any of the job control or MERVA ESA (or network links) generation parameters for the program been changed since the previous successful run?	User error	Check the job control statements, MERVA ESA customizing parameters, and the network-links generation parameters.
Program (Module, Subroutine) Code		
Has there been a reassembly of the failing module since the previous successful run?	User error	Ensure that the change was done correctly.
Has the program or any of its modules been recataloged since the previous successful run?	User error	Ensure that the change was done correctly.
Input/Output Media and Devices		
Has there been a change in the location of data on any of the media, for example, has a data set been moved to another disk?	User error	Ensure that the job control is correct for the new storage device.
Has there been a change in any of the input media, for example, types of control units?	User or hardware error	Ensure that the job control and system changes are correct; if I/O error, inform hardware engineer.
Was a different I/O device type assigned, for example, different models of IBM 3390?	User or hardware error	Ensure that the job control and system changes are correct; if I/O error, inform hardware engineer.

Table 2 on page 92 lists the possible error conditions in MERVA ESA and recommends the action you should take.

Table 2. Conditions within MERVA ESA and its Components

Condition	Recommended Action
An error message from the system or MERVA ESA	Look at the message explanation in the appropriate books. There you can find advice on what to do next.
MERVA ESA or a network link terminates unexpectedly	Look at the associated message that gives the reason for the termination. Check the job control. Refer to “Building a Symptom String”.
Loop or wait in MERVA ESA or a network link	Refer to “Building a Symptom String”.
Program check	Refer to “Building a Symptom String”.
Incorrect output	Check whether all the needed input was supplied properly. Make sure the output is incorrect, not just in an unexpected format. Refer to “Building a Symptom String”.

Building a Symptom String

If your initial problem evaluation points to an error in code supplied by IBM, inform IBM of the program failures, describing them in a “symptom string.” A symptom string consists of a series of “keywords.” A keyword can be a word, an abbreviation, or a number, and is used to describe a single aspect of a program failure.

The following types of information can be identified in the symptom string by the use of keywords:

- The component of MERVA ESA in which the problem has arisen (that is, Base Functions, the SWIFT Link, the Telex Link, or the MERVA Link).
- The type of failure that has occurred.
- The version, release level, and modification level of MERVA ESA.
- The maintenance level of MERVA ESA.

  Note: This is not used directly in the symptom string, but during the search process.
- The area of failure: for example, to identify the module that has failed.

The symptom string can be used to search the Early Warning System (EWS) for a similar problem that has already been reported. The EWS is a set of microfiche copies of the known problems within IBM licensed programs, and is available to the users of these licensed programs. If a matching problem is found, it will contain an error description for the problem, and usually a fix or a circumvention.

Under MVS, the symptom keywords can be used more conveniently to search the online Information/Systems database. This database, which can be installed in MVS systems, also contains descriptions of the known problems in IBM licensed programs.

The problem should be reported to the IBM Support Center if:

- You do not have access to the EWS or to the Information/Systems database.
- No matching problem is found.

- No fix was provided for the problem.
- The fix did not resolve the problem.

The representative of the IBM Support Center uses the symptom keywords to search an online database for similar known problems. This database contains more recent information, and it is possible that a matching problem can be found there.

The first keyword of the symptom string identifies MERVA ESA by means of a component identifier. A search of the software-support database with this keyword alone finds all reported problems for the licensed program. Each additional keyword added to the symptom string reduces the number of matches and thereby narrows the search.

In some cases, a search with less than a full string of keywords can find a solution to a problem. Therefore, if circumstances make it particularly difficult to follow the instructions given for selecting a keyword, you can omit it. But, in general, IBM maintenance expects you to identify your problem with a full string of keywords.

Building a full string of keywords for MERVA ESA is a four-step process. The steps identify:

1. The component, identified by the component ID
2. The type of failure and, if known, the module that failed
3. The release level
4. The maintenance level.

Step 1: Component-Identifier Keyword

The component-identification number is the first keyword in the search argument. It identifies the library within the software-support database that contains Authorized Program Analysis Reports (APARs) for MERVA ESA. This keyword should be used whenever you suspect that MERVA ESA is the component in error.

The component identifier for MERVA ESA V4.1 is:

5648–B29

Step 2: Type-of-Failure Keyword

To identify the type of failure that has occurred, select the keyword that seems best to describe the problem from the following list, and then go to the specific instructions for that keyword:

1. Define the types of failure to determine the appropriate failure diagnosis.
2. Follow the procedures provided for step-by-step investigation of each type of failure.

If you are not certain which of several applicable keywords to use, select the one that appears first in the list.

Keyword	Type of Failure
PROGCK	A program check occurs.
ABEND*xxx*	An abnormal end occurs.
MSG*x*	A message was issued because of a MERVA ESA problem, or there is a problem with a MERVA ESA message.

Keyword	Type of Failure
LOOP	MERVA ESA or a network link seems to be doing something repetitively.
WAIT	MERVA ESA or a network link does not seem to be doing anything.
INCORROUT	Output from MERVA ESA or a network link is incorrect or missing.
DOC*n*	There is a problem in the program documentation.
PERFM	Performance of the program is decreased.

PROGCK Use this keyword when a program-check condition occurs.

When a program-check condition occurs, the system shows the type of program exception and the condition code.

Make a note of the condition code and the type of program exception (such as address exception or operation exception).

ABEND*xxx* Use this keyword when a system-control programming (SCP) or user abnormal program end occurs.

Replace the *xxx* part of the ABENDxxx keyword with the ABEND code from either the message or the ABEND dump.

Make a note of the name of the module that has failed.

When an abnormal end occurs, MERVA ESA produces a system dump.

Note: User abnormal ends (Uxxx) can also be initiated by CICS or IMS.

MSG*x* Use this keyword when:

- A message is issued by MERVA ESA and the recommended operator action is to contact IBM.
- A message is not issued under a set of conditions that should have caused it to be issued.
- A message is issued under a set of conditions that should not have caused it to be issued.
- A message contains invalid data or data is missing.

Messages can be issued by programs other than MERVA ESA or its components. If you receive a message from another system (SCP, VSAM) or licensed program used, determine the reason for which the message is produced.

If it appears to be a problem with MERVA ESA, pursue it with the appropriate symptom string; if it does not seem to be an error within MERVA ESA, examine the involved components.

Replace the *x* of the MSG*x* keyword with the complete message identifier. For example, if the message identifier is **DSL042I**, the MSG*x* keyword is MSGDSL042I.

LOOP Use this keyword when MERVA ESA or a network link seems to be doing something repetitively.

Use the diagnostic practices of the system-control program or the data-communication system to produce a dump, examine the dump, and determine the looping address. Determine which module(s) are involved.

WAIT
Use this keyword when MERVA ESA or a network link does not seem to be doing anything.

Use the diagnostic practices of the system control program or the data communication system to produce a dump, examine the dump, and determine the waiting address. Determine which modules are involved.

INCORROUT
Use this keyword when output is missing or incorrect. INCORROUT situations are:

- Output was expected, but not received (missing).
- Output was different from what was expected (incorrect).
- Data returned by the program is missing or incorrect.

DOC*n*
Use this keyword when incorrect or missing information in the MERVA ESA documentation appears to be the cause of a problem.

Replace the *n* of the DOC*n* keyword with the order number of the document (omit the hyphens). For example, if the order number is SH12-6382-00, the DOC keyword is DOCSH12638200.

Find the page in the document on which the error was detected and prepare a description of the problem. You should include this information in the error description for submitting an APAR.

PERFM
Use this keyword when the performance of the program is decreased.

Most performance problems can be related to system tuning, because of an inappropriate selection of program options or parameters.

When you have identified a type-of-failure keyword, read the description of the release-level keyword to continue building the keyword string. This is supplied in Step 3: Release-Level Keyword.

Step 3: Release-Level Keyword

AR*xxx*
Use this keyword to identify the version, release level, and modification level of MERVA ESA (the release level is optional for the search of the EWS). You must include the release-level keyword in an APAR.

If you do not know, or are not sure of, the release level of MERVA ESA, you can find it in the module identification of any MERVA ESA module, whichever is appropriate. Refer to “Step 4: Maintenance-Level Procedure” on page 96 for the procedure that can be used to get a copy of the module to examine its module identification.

The release-level keyword has the format AR*xxx*. For *xxx* use three digits identifying Version, Release, and Modification Level of MERVA ESA, in that order, for example, MERVA ESA: Version 4 Release 1 Modification Level 0 would be represented as: AR410.

To continue building the keyword string, read the following.

Step 4: Maintenance-Level Procedure

Use this procedure to determine the maintenance level of MERVA ESA by identifying APARs that have been applied. This information is not used directly in the symptom string, but rather during the search process.

When you have identified the module or modules in which the failure has occurred, do the following:

In an MVS environment:

- Use the System Modification Program's (SMP) LIST CDS control statement to list the MERVA ESA control data set (CDS) to determine the latest maintenance, update, or replacement applied to that module.

  Or:
- Get a superzap dump of the module, by using the program AMASPZAP, and check the maintenance level given in the module identification when there is any doubt about the integrity of the CDS.

In a VSE environment:

- Use the Maintain System History Program (MSHP) LOOKUP function to determine what maintenance has been applied to the module or modules.

  Or:
- Get a copy of the module using the RSERV or CSERV functions, and check the maintenance level supplied as part of the module identification.

Area-of-Failure Keywords

The procedures here use the problem symptoms to narrow the area of the code that could contain the error: for example, they might be used to identify the module that failed. The area-of-failure keyword associates a problem with a subset of the code. The area of the subset is used as an area-of-failure keyword in the symptom string. One or more of the following types of keywords in Table 3 on page 97 should be used to describe the area of failure.

Table 3. Definitions of Area-of-Failure Keywords

Type of Keyword	Definition
Function Keyword	• Identifies a major part of the MERVA ESA program processing. **DSLNUC** Nucleus **DSLEUD** End-User Driver **DSLMFS** Message Format Service **DSLTSV** TOF Supervisor **MCB** Message Control Block **FDT** Field Definition Table **DSLSDI, DSLSDO, DSLSDY** Sequential data-set processing **DSLHCP** Hardcopy printing **DSLCXT** Expansion transaction **DSLFLUT, DSLQDSUT, DSLQMNT, DSLCNTUT** Utilities **DSLKQR, DSLKQS** MERVA-MQI Attachment transactions. • Identifies a major part of the SWIFT Link program processing. **DWSAUTLD** Authenticator-Key File Load Program **DWSAUTT** Authenticator-Key File Update Transaction **DWSCORUT** Address file program **DWSMCxxx** Checking and separation **SWIFT** Communication with the SWIFT network. • Identifies a major part of the Telex Link program processing. **ADDRESS** Automatic address expansion **TELEX** Preparation of messages for telex transmission **TESTKEY** Interface to the test-key processing program **COMM** Communication between the Telex Link and the Telex Substation. • Identifies a major part of the MERVA Link program processing. **EKAEMSC** System Control Facility **EKASEND** Sending messages **EKARECV** Receiving messages. • Produces, together with the symptom string, a more effective search argument.
Subfunction Keyword	• Describes an operation carried out within a function. • Is useful to identify subsets of the processing done by a large function (combine the subfunction keyword with the function keyword).
Module Keyword	• Names the failing module in control. • Can be identified by using a dump.
Modifier Keyword	• Names a programming statement, command failure, or option that seems to cause the failure. • Can identify the area of failure as effectively as a function, subfunction, or module keyword.

Submitting an Authorized Program Analysis Report (APAR)

You should contact IBM for assistance in problem determination when you are sure that:

- You have checked your own specifications for accuracy.
- You have followed the diagnostic procedures.
- The keyword search has failed to find a solution.

The basic document to be submitted with an APAR is the "APAR Error Description." This document should contain all the descriptive details that will not necessarily appear in any of the materials listed in the following. Depending on the type of problem, the error description should contain, for example:

- The frequency of the problem, for example once a day, once an hour, or every few minutes
- The circumstances in which the problem occurs, for example, under certain stress situations
- Which traps (if any) were applied to produce the submitted listings.

All the above suggestions should be judged individually to ensure their validity for a particular problem being reported.

For all types of problem with MERVA ESA, the following items are required:

- MERVA ESA journal printout. The journal records should be printed in hexadecimal *and* in character representation, preferably with IDCAMS or an equivalent program. A print of a journal file that is evaluated by a user program, or that is printed in character representation only, is not as useful.
- Generalized Trace Facility (GTF) trace for performance problems in MVS.
- The System Debugging Trace Facility (SDAID) for performance problems in VSE.
- Dump listings.
- Listing of the parameters used in generating MERVA ESA.
- Listings of routing modules.
- Console logs.
- Listings of the IMS nucleus generation, for IMS.
- A list of PTFs and APARs applied to MERVA ESA.

If the problem can be reproduced, start the error trace to reproduce the error situation, and attach the printed information to the APAR material.

If it is necessary to submit an APAR, you should follow the instructions of your IBM support representative, who will process the APAR through IBM's database of known problems and arrange for the installation of the fix.

Appendix. Notices

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

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

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

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

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

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

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

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

IBM Deutschland
Informationssysteme GmbH
Department 3982
Pascalstrasse 100

Trademarks

Glossary of Terms and Abbreviations

This glossary defines terms as they are used in this book. If you do not find the terms you are looking for, refer to the *IBM Dictionary of Computing*, New York: McGraw-Hill, and the *S.W.I.F.T. User Handbook*.

A

ACB. Access method control block.

ACC. MERVA Link USS application control command application. It provides a means of operating MERVA Link USS in USS shell and MVS batch environments.

Access method control block (ACB). A control block that links an application program to VSAM or VTAM.

ACD. MERVA Link USS application control daemon.

ACT. MERVA Link USS application control table.

address. See *SWIFT address*.

address expansion. The process by which the full name of a financial institution is obtained using the SWIFT address, telex correspondent's address, or a nickname.

AMPDU. Application message protocol data unit, which is defined in the MERVA Link P1 protocol, and consists of an envelope and its content.

answerback. In telex, the response from the dialed correspondent to the WHO R U signal.

answerback code. A group of up to 6 letters following or contained in the answerback. It is used to check the answerback.

APC. Application control.

API. Application programming interface.

APPC. Advanced Program-to-Program Communication based on SNA LU 6.2 protocols.

APPL. A VTAM definition statement used to define a VTAM application program.

application programming interface (API). An interface that programs can use to exchange data.

application support filter (ASF). In MERVA Link, a user-written program that can control and modify any data exchanged between the Application Support Layer and the Message Transfer Layer.

application support process (ASP). An executing instance of an application support program. Each application support process is associated with an ASP entry in the partner table. An ASP that handles outgoing messages is a *sending ASP*; one that handles incoming messages is a *receiving ASP*.

application support program (ASP). In MERVA Link, a program that exchanges messages and reports with a specific remote partener ASP. These two programs must agree on which conversation protocol they are to use.

ASCII. American Standard Code for Information Interchange. The standard code, using a coded set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

ASF. Application support filter.

ASF. (1) Application support process. (2) Application support program.

ASPDU. Application support protocol data unit, which is defined in the MERVA Link P2 protocol.

authentication. The SWIFT security check used to ensure that a message has not changed during transmission, and that it was sent by an authorized sender.

authenticator key. A set of alphanumeric characters used for the authentication of a message sent via the SWIFT network.

authenticator-key file. The file that stores the keys used during the authentication of a message. The file contains a record for each of your financial institution's correspondents.

B

Back-to-Back (BTB). A MERVA Link function that enables ASPs to exchange messages in the local MERVA Link node without using data communication services.

bank identifier code. A 12-character code used to identify a bank within the SWIFT network. Also called a SWIFT address. The code consists of the following subcodes:
- The bank code (4 characters)
- The ISO country code (2 characters)
- The location code (2 characters)
- The address extension (1 character)

- The branch code (3 characters) for a SWIFT user institution, or the letters "BIC" for institutions that are not SWIFT users.

Basic Security Manager (BSM). A component of VSE/ESA Version 2.4 that is invoked by the System Authorization Facility, and used to ensure signon and transaction security.

BIC. Bank identifier code.

BIC Bankfile. A tape of bank identifier codes supplied by S.W.I.F.T.

BIC Database Plus Tape. A tape of financial institutions and currency codes, supplied by S.W.I.F.T. The information is compiled from various sources and includes national, international, and cross-border identifiers.

BIC Directory Update Tape. A tape of bank identifier codes and currency codes, supplied by S.W.I.F.T., with extended information as published in the printed BIC Directory.

body. The second part of an IM-ASPDU. It contains the actual application data or the message text that the IM-AMPDU transfers.

BSC. Binary synchronous control.

BSM. Basic Security Manager.

BTB. Back-to-back.

buffer. A storage area used by MERVA programs to store a message in its internal format. A buffer has an 8-byte prefix that indicates its length.

C

CBT. SWIFT computer-based terminal.

CCSID. Coded character set identifier.

CDS. Control data set.

central service. In MERVA, a service that uses resources that either require serialization of access, or are only available in the MERVA nucleus.

CF message. Confirmed message. When a sending MERVA Link system is informed of the successful delivery of a message to the receiving application, it routes the delivered application messages as CF messages, that is, messages of class CF, to an ACK wait queue or to a complete message queue.

COA. Confirm on arrival.

COD. Confirm on delivery.

coded character set identifier (CCSID). The name of a coded set of characters and their code point assignments.

commit. In MQSeries, to commit operations is to make the changes on MQSeries queues permanent. After putting one or more messages to a queue, a commit makes them visible to other programs. After getting one or more messages from a queue, a commit permanently deletes them from the queue.

confirm-on-arrival (COA) report. An MQSeries report message type created when a message is placed on that queue. It is created by the queue manager that owns the destination queue.

confirm-on-delivery (COD) report. An MQSeries report message type created when an application retrieves a message from the queue in a way that causes the message to be deleted from the queue. It is created by the queue manager.

control fields. In MERVA Link, fields that are part of a MERVA message on the queue data set and of the message in the TOF. Control fields are written to the TOF at nesting identifier 0. Messages in SWIFT format do not contain control fields.

correspondent. An institution to which your institution sends and from which it receives messages.

correspondent identifier. The 11-character identifier of the receiver of a telex message. Used as a key to retrieve information from the Telex correspondents file.

cross-system coupling facility. See *XCF*.

coupling services. In a sysplex, the functions of XCF that transfer data and status information among the members of a group that reside in one or more of the MVS systems in the sysplex.

couple data set. See *XCF couple data set*.

CTP. MERVA Link command transfer processor.

currency code file. A file containing the currency codes, together with the name, fraction length, country code, and country names.

D

daemon. A long-lived process that runs unattended to perform continuous or periodic systemwide functions.

DASD. Direct access storage device.

data area. An area of a predefined length and format on a panel in which data can be entered or displayed. A field can consist of one or more data areas.

data element. A unit of data that, in a certain context, is considered indivisible. In MERVA Link, a data element consists of a 2-byte data element length field, a 2-byte data-element identifier field, and a field of variable length containing the data element data.

datagram. In TCP/IP, the basic unit of information passed across the Internet environment. This type of message does not require a reply, and is the simplest type of message that MQSeries supports.

data terminal equipment. That part of a data station that serves as a data source, data link, or both, and provides for the data communication control function according to protocols.

DB2. A family of IBM licensed programs for relational database management.

dead-letter queue. A queue to which a queue manager or application sends messages that it cannot deliver. Also called *undelivered-message queue*.

dial-up number. A series of digits required to establish a connection with a remote correspondent via the public telex network.

direct service. In MERVA, a service that uses resources that are always available and that can be used by several requesters at the same time.

display mode. The mode (PROMPT or NOPROMPT) in which SWIFT messages are displayed. See *PROMPT mode* and *NOPROMPT mode.*

distributed queue management (DQM). In MQSeries message queuing, the setup and control of message channels to queue managers on other systems.

DQM. Distributed queue management.

DTE. Data terminal equipment.

E

EBCDIC. Extended Binary Coded Decimal Interchange Code. A coded character set consisting of 8-bit coded characters.

ECB. Event control block.

EDIFACT. Electronic Data Interchange for Administration, Commerce and Transport (a United Nations standard).

ESM. External security manager.

EUD. End-user driver.

exception report. An MQSeries report message type that is created by a message channel agent when a message is sent to another queue manager, but that message cannot be delivered to the specified destination queue.

external line format (ELF) messages. Messages that are not fully tokenized, but are stored in a single field in the TOF. Storing messages in ELF improves performance, because no mapping is needed, and checking is not performed.

external security manager (ESM). A security product that is invoked by the System Authorization Facility. RACF is an example of an ESM.

F

FDT. Field definition table.

field. In MERVA, a portion of a message used to enter or display a particular type of data in a predefined format. A field is located by its position in a message and by its tag. A field is made up of one or more data areas. See also *data area*.

field definition table (FDT). The field definition table describes the characteristics of a field; for example, its length and number of its data areas, and whether it is mandatory. If the characteristics of a field change depending on its use in a particular message, the definition of the field in the FDT can be overridden by the MCB specifications.

field group. One or several fields that are defined as being a group. Because a field can occur more than once in a message, field groups are used to distinguish them. A name can be assigned to the field group during message definition.

field group number. In the TOF, a number is assigned to each field group in a message in ascending order from 1 to 255. A particular field group can be accessed using its field group number.

field tag. A character string used by MERVA to identify a field in a network buffer. For example, for SWIFT field 30, the field tag is **:30:**.

FIN. Financial application.

FIN-Copy. The MERVA component used for SWIFT FIN-Copy support.

finite state machine. The theoretical base describing the rules of a service request’s state and the conditions to state transitions.

FMT/ESA. MERVA-to-MERVA Financial Message Transfer/ESA.

form. A partially-filled message containing data that can be copied for a new message of the same message type.

G

GPA. General purpose application.

H

HFS. Hierarchical file system.

hierarchical file system (HFS). A system for organizing files in a hierarchy, as in a UNIX system. OS/390 UNIX System Services files are organized in an HFS. All files are members of a directory, and each directory is in turn a member of a directory at a higher level in the HFS. The highest level in the hierarchy is the root directory.

I

IAM. Interapplication messaging (a MERVA Link message exchange protocol).

IM-ASPDU. Interapplication messaging application support protocol data unit. It contains an application message and consists of a heading and a body.

incore request queue. Another name for the request queue to emphasize that the request queue is held in memory instead of on a DASD.

InetD. Internet Daemon. It provides TCP/IP communication services in the OS/390 USS environment.

initiation queue. In MQSeries, a local queue on which the queue manager puts trigger messages.

input message. A message that is input into the SWIFT network. An input message has an input header.

INTERCOPE TelexBox. This telex box supports various national conventions for telex procedures and protocols.

interservice communication. In MERVA ESA, a facility that enables communication among services if MERVA ESA is running in a multisystem environment.

intertask communication. A facility that enables application programs to communicate with the MERVA nucleus and so request a central service.

IP. Internet Protocol.

IP message. In-process message. A message that is in the process of being transferred to another application.

ISC. Intersystem communication.

ISN. Input sequence number.

ISN acknowledgment. A collective term for the various kinds of acknowledgments sent by the SWIFT network.

ISO. International Organization for Standardization.

ITC. Intertask communication.

J

JCL. Job control language.

journal. A chronological list of records detailing MERVA actions.

journal key. A key used to identify a record in the journal.

journal service. A MERVA central service that maintains the journal.

K

KB. Kilobyte (1024 bytes).

key. A character or set of characters used to identify an item or group of items. For example, the user ID is the key to identify a user file record.

key-sequenced data set (KSDS). A VSAM data set whose records are loaded in key sequence and controlled by an index.

keyword parameter. A parameter that consists of a keyword, followed by one or more values.

KSDS. Key-sequenced data set.

L

LAK. Login acknowledgment message. This message informs you that you have successfully logged in to the SWIFT network.

large message. A message that is stored in the large message cluster (LMC). The maximum length of a message to be stored in the VSAM QDS is 31900 bytes. Messages up to 2MB can be stored in the LMC. For queue management using DB2 no distinction is made between messages and large messages.

large queue element. A queue element that is larger than the smaller of:

- The limiting value specified during the customization of MERVA
- 32KB

LC message. Last confirmed control message. It contains the message-sequence number of the application or acknowledgment message that was last confirmed; that is, for which the sending MERVA Link system most recently received confirmation of a successful delivery.

LDS. Logical data stream.

LMC. Large message cluster.

LNK. Login negative acknowledgment message. This message indicates that the login to the SWIFT network has failed.

local queue. In MQSeries, a queue that belongs to a local queue manager. A local queue can contain a list of messages waiting to be processed. Contrast with *remote queue*.

local queue manager. In MQSeries, the queue manager to which the program is connected, and that provides message queuing services to that program. Queue managers to which a program is not connected are remote queue managers, even if they are running on the same system as the program.

login. To start the connection to the SWIFT network.

LR message. Last received control message, which contains the message-sequence number of the application or acknowledgment message that was last received from the partner application.

LSN. Login sequence number.

LT. See *LTERM*.

LTC. Logical terminal control.

LTERM. Logical terminal. Logical terminal names have 4 characters in CICS and up to 8 characters in IMS.

LU. A VTAM logical unit.

M

maintain system history program (MSHP). A program used for automating and controlling various installation, tailoring, and service activities for a VSE system.

MCA. Message channel agent.

MCB. Message control block.

MERVA ESA. The IBM licensed program Message Entry and Routing with Interfaces to Various Applications for ESA.

MERVA Link. A MERVA component that can be used to interconnect several MERVA systems.

message. A string of fields in a predefined form used to provide or request information. See also *SWIFT financial message.*

message body. The part of the message that contains the message text.

message category. A group of messages that are logically related within an application.

message channel. In MQSeries distributed message queuing, a mechanism for moving messages from one queue manager to another. A message channel comprises two message channel agents (a sender and a receiver) and a communication link.

message channel agent (MCA). In MQSeries, a program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue.

message control block (MCB). The definition of a message, screen panel, net format, or printer layout made during customization of MERVA.

Message Format Service (MFS). A MERVA direct service that formats a message according to the medium to be used, and checks it for formal correctness.

message header. The leading part of a message that contains the sender and receiver of the message, the message priority, and the type of message.

Message Integrity Protocol (MIP). In MERVA Link, the protocol that controls the exchange of messages between partner ASPs. This protocol ensures that any loss of a message is detected and reported, and that no message is duplicated despite system failures at any point during the transfer process.

message-processing function. The various parts of MERVA used to handle a step in the message-processing route, together with any necessary equipment.

message queue. See *queue*.

Message Queue Interface (MQI). The programming interface provided by the MQSeries queue managers. It provides a set of calls that let application programs access message queuing services such as sending messages, receiving messages, and manipulating MQSeries objects.

Message Queue Manager (MQM). An IBM licensed program that provides message queuing services. It is part of the MQSeries set of products.

message reference number (MRN). A unique 16-digit number assigned to each message for identification purposes. The message reference number consists of an 8-digit domain identifier that is followed by an 8-digit sequence number.

message sequence number (MSN). A sequence number for messages transferred by MERVA Link.

message type (MT). A number, up to 7 digits long, that identifies a message. SWIFT messages are identified by a 3-digit number; for example SWIFT message type MT S100.

MFS. Message Format Service.

MIP. Message Integrity Protocol.

MPDU. Message protocol data unit, which is defined in P1.

MPP. In IMS, message-processing program.

MQA. MQ Attachment.

MQ Attachment (MQA). A MERVA feature that provides message transfer between MERVA and a user-written MQI application.

MQH. MQSeries queue handler.

MQI. Message queue interface.

MQM. Message queue manager.

MQS. MQSeries nucleus server.

MQSeries. A family of IBM licensed programs that provides message queuing services.

MQSeries nucleus server (MQS). A MERVA component that listens for messages on an MQI queue, receives them, extracts a service request, and passes it via the request queue handler to another MERVA ESA instance for processing.

MQSeries queue handler (MQH). A MERVA component that performs service calls to the Message Queue Manager via the provided Message Queue Interface.

MRN. Message reference number.

MSC. MERVA system control facility.

MSHP. Maintain system history program.

MSN. Message sequence number.

MT. Message type.

MTP. (1) Message transfer program. (2) Message transfer process.

MTS. Message Transfer System.

MTSP. Message Transfer Service Processor.

MTT. Message type table.

multisystem application. (1) An application program that has various functions distributed across MVS systems in a multisystem environment. (2) In XCF, an authorized application that uses XCF coupling services. (3) In MERVA ESA, multiple instances of MERVA ESA that are distributed among different MVS systems in a multisystem environment.

multisystem environment. An environment in which two or more MVS systems reside on one or more processors, and programs on one system can communicate with programs on the other systems. With XCF, the environment in which XCF services are available in a defined sysplex.

multisystem sysplex. A sysplex in which one or more MVS systems can be initialized as part of the sysplex. In a multisystem sysplex, XCF provides coupling services on all systems in the sysplex and requires an XCF couple data set that is shared by all systems. See also *single-system sysplex*.

MVS/ESA. Multiple Virtual Storage/Enterprise Systems Architecture.

N

namelist. An MQSeries for MVS/ESA object that contains a list of queue names.

nested message. A message that is composed of one or more message types.

nested message type. A message type that is contained in another message type. In some cases, only part of a message type (for example, only the mandatory fields) is nested, but this “partial” nested message type is also considered to be nested. For example, SWIFT MT 195 could be used to request information about a SWIFT MT 100 (customer transfer). The SWIFT MT 100 (or at least its mandatory fields) is then nested in SWIFT MT 195.

nesting identifier. An identifier (a number from 2 to 255) that is used to access a nested message type.

network identifier. A single character that is placed before a message type to indicate which network is to be used to send the message; for example, **S** for SWIFT

network service access point (NSAP). The endpoint of a network connection used by the SWIFT transport layer.

NOPROMPT mode. One of two ways to display a message panel. NOPROMPT mode is only intended for experienced SWIFT Link users who are familiar with the structure of SWIFT messages. With NOPROMPT mode, only the SWIFT header, trailer, and pre-filled fields and their tags are displayed. Contrast with *PROMPT mode*.

NSAP. Network service access point.

nucleus server. A MERVA component that processes a service request as selected by the request queue handler. The service a nucleus server provides and the way it provides it is defined in the nucleus server table (DSLNSVT).

O

object. In MQSeries, objects define the properties of queue managers, queues, process definitions, and namelists.

occurrence. See *repeatable sequence*.

option. One or more characters added to a SWIFT field number to distinguish among different layouts for and meanings of the same field. For example, SWIFT field 60 can have an option F to identify a first opening balance, or M for an intermediate opening balance.

origin identifier (origin ID). A 34-byte field of the MERVA user file record. It indicates, in a MERVA and SWIFT Link installation that is shared by several banks, to which of these banks the user belongs. This lets the user work for that bank only.

OSN. Output sequence number.

OSN acknowledgment. A collective term for the various kinds of acknowledgments sent to the SWIFT network.

output message. A message that has been received from the SWIFT network. An output message has an output header.

P

P1. In MERVA Link, a peer-to-peer protocol used by cooperating message transfer processes (MTPs).

P2. In MERVA Link, a peer-to-peer protocol used by cooperating application support processes (ASPs).

P3. In MERVA Link, a peer-to-peer protocol used by cooperating command transfer processors (CTPs).

packet switched public data network (PSPDN). A public data network established and operated by network common carriers or telecommunication administrations for providing packet-switched data transmission.

panel. A formatted display on a display terminal. Each page of a message is displayed on a separate panel.

parallel processing. The simultaneous processing of units of work by several servers. The units of work can be either transactions or subdivisions of larger units of work.

parallel sysplex. A sysplex that uses one or more coupling facilities.

partner table (PT). In MERVA Link, the table that defines how messages are processed. It consists of a header and different entries, such as entries to specify the message-processing parameters of an ASP or MTP.

PCT. Program Control Table (of CICS).

PDE. Possible duplicate emission.

PDU. Protocol data unit.

PF key. Program-function key.

positional parameter. A parameter that must appear in a specified location relative to other parameters.

PREMIUM. The MERVA component used for SWIFT PREMIUM support.

process definition object. An MQSeries object that contains the definition of an MQSeries application. A queue manager uses the definitions contained in a process definition object when it works with trigger messages.

program-function key. A key on a display terminal keyboard to which a function (for example, a command) can be assigned. This lets you execute the function (enter the command) with a single keystroke.

PROMPT mode. One of two ways to display a message panel. PROMPT mode is intended for SWIFT Link users who are unfamiliar with the structure of SWIFT messages. With PROMPT mode, all the fields and tags are displayed for the SWIFT message. Contrast with *NOPROMPT mode*.

protocol data unit (PDU). In MERVA Link a PDU consists of a structured sequence of implicit and explicit data elements:
- Implicit data elements contain other data elements.
- Explicit data elements cannot contain any other data elements.

PSN. Public switched network.

PSPDN. Packet switched public data network.

PSTN. Public switched telephone network.

PT. Partner table.

PTT. A national post and telecommunication authority (post, telegraph, telephone).

Q

QDS. Queue data set.

QSN. Queue sequence number.

queue. (1) In MERVA, a logical subdivision of the MERVA queue data set used to store the messages associated with a MERVA message-processing function. A queue has the same name as the message-processing function with which it is associated. (2) In MQSeries, an object onto which message queuing applications can put messages, and from which they can get messages. A queue is owned and maintained by a queue manager. See also *request queue*.

queue element. A message and its related control information stored in a data record in the MERVA ESA Queue Data Set.

queue management. A MERVA service function that handles the storing of messages in, and the retrieval of messages from, the queues of message-processing functions.

queue manager. (1) An MQSeries system program that provides queueing services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns. See also *local queue manager* and *remote queue manager*. (2) The MQSeries object that defines the attributes of a particular queue manager.

queue sequence number (QSN). A sequence number that is assigned to the messages stored in a logical queue by MERVA ESA queue management in ascending order. The QSN is always unique in a queue. It is reset to zero when the queue data set is formatted, or when a queue management restart is carried out and the queue is empty.

R

RACF. Resource Access Control Facility.

RBA. Relative byte address.

RC message. Recovered message; that is, an IP message that was copied from the control queue of an inoperable or closed ASP via the **recover** command.

ready queue. A MERVA queue used by SWIFT Link to collect SWIFT messages that are ready for sending to the SWIFT network.

remote queue. In MQSeries, a queue that belongs to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with *local queue*.

remote queue manager. In MQSeries, a queue manager is remote to a program if it is not the queue manager to which the program is connected.

repeatable sequence. A field or a group of fields that is contained more than once in a message. For example, if the SWIFT fields 20, 32, and 72 form a sequence, and if this sequence can be repeated up to 10 times in a message, each sequence of the fields 20, 32, and 72 would be an occurrence of the repeatable sequence.

In the TOF, the occurrences of a repeatable sequence are numbered in ascending order from 1 to 32767 and can be referred to using the occurrence number.

A repeatable sequence in a message may itself contain another repeatable sequence. To identify an occurrence within such a nested repeatable sequence, more than one occurrence number is necessary.

reply message. In MQSeries, a type of message used for replies to request messages.

reply-to queue. In MQSeries, the name of a queue to which the program that issued an MQPUT call wants a reply message or report message sent.

report message. In MQSeries, a type of message that gives information about another message. A report message usually indicates that the original message cannot be processed for some reason.

request message. In MQSeries, a type of message used for requesting a reply from another program.

request queue. The queue in which a service request is stored. It resides in main storage and consists of a set of request queue elements that are chained in different queues:

- Requests waiting to be processed
- Requests currently being processed
- Requests for which processing has finished

request queue handler (RQH). A MERVA ESA component that handles the queueing and scheduling of service requests. It controls the request processing of a nucleus server according to rules defined in the finite state machine.

Resource Access Control Facility (RACF). An IBM licensed program that provides for access control by identifying and verifying users to the system, authorizing access to protected resources, logging detected unauthorized attempts to enter the system, and logging detected accesses to protected resources.

retype verification. See *verification*.

routing. In MERVA, the passing of messages from one stage in a predefined processing path to the next stage.

RP. Regional processor.

RQH. Request queue handler.

RRDS. Relative record data set.

S

SAF. System Authorization Facility.

SCS. SNA character string

SCP. System control process.

SDI. Sequential data set input. A batch utility used to import messages from a sequential data set or a tape into MERVA ESA queues.

SDO. Sequential data set output. A batch utility used to export messages from a MERVA ESA queue to a sequential data set or a tape.

SDY. Sequential data set system printer. A batch utility used to print messages from a MERVA ESA queue.

service request. A type of request that is created and passed to the request queue handler whenever a nucleus server requires a service that is not currently available.

sequence number. A number assigned to each message exchanged between two nodes. The number is increased by one for each successive message. It starts from zero each time a new session is established.

sign off. To end a session with MERVA.

sign on. To start a session with MERVA.

single-system sysplex. A sysplex in which only one MVS system can be initialized as part of the sysplex. In a single-system sysplex, XCF provides XCF services on the system, but does not provide signalling services between MVS systems. A single-system sysplex requires an XCF couple data set. See also *multisystem sysplex*.

small queue element. A queue element that is smaller than the smaller of:

- The limiting value specified during the customization of MERVA
- 32KB

SMP/E. System Modification Program Extended.

SN. Session number.

SNA. Systems network architecture.

SNA character string. In SNA, a character string composed of EBCDIC controls, optionally mixed with user data, that is carried within a request or response unit.

SPA. Scratch pad area.

SQL. Structured Query Language.

SR-ASPDU. The status report application support PDU, which is used by MERVA Link for acknowledgment messages.

SSN. Select sequence number.

subfield. A subdivision of a field with a specific meaning. For example, the SWIFT field 32 has the subfields date, currency code, and amount. A field can have several subfield layouts depending on the way the field is used in a particular message.

SVC. (1) Switched Virtual Circuit. (2) Supervisor call instruction.

S.W.I.F.T. (1) Society for Worldwide Interbank Financial Telecommunication s.c. (2) The network provided and managed by the Society for Worldwide Interbank Financial Telecommunication s.c.

SWIFT address. Synonym for *bank identifier code*.

SWIFT Correspondents File. The file containing the bank identifier code (BIC), together with the name, postal address, and zip code of each financial institution in the BIC Directory.

SWIFT financial message. A message in one of the SWIFT categories 1 to 9 that you can send or receive via the SWIFT network. See *SWIFT input message* and *SWIFT output message*.

SWIFT header. The leading part of a message that contains the sender and receiver of the message, the message priority, and the type of message.

SWIFT input message. A SWIFT message with an input header to be sent to the SWIFT network.

SWIFT link. The MERVA ESA component used to link to the SWIFT network.

SWIFT network. Refers to the SWIFT network of the Society for Worldwide Interbank Financial Telecommunication (S.W.I.F.T.).

SWIFT output message. A SWIFT message with an output header coming from the SWIFT network.

SWIFT system message. A SWIFT general purpose application (GPA) message or a financial application (FIN) message in SWIFT category 0.

switched virtual circuit (SVC). An X.25 circuit that is dynamically established when needed. It is the X.25 equivalent of a switched line.

sysplex. One or more MVS systems that communicate and cooperate via special multisystem hardware components and software services.

System Authorization Facility (SAF). An MVS or VSE facility through which MERVA ESA communicates with an external security manager such as RACF (for MVS) or the basic security manager (for VSE).

System Control Process (SCP). A MERVA Link component that handles the transfer of MERVA ESA commands to a partner MERVA ESA system, and the receipt of the command response. It is associated with a system control process entry in the partner table.

System Modification Program Extended (SMP/E). A licensed program used to install software and software changes on MVS systems.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operating sequences for transmitting information units through, and for controlling the configuration and operation of, networks.

T

tag. A field identifier.

TCP/IP. Transmission Control Protocol/Internet Protocol.

Telex Correspondents File. A file that stores data about correspondents. When the user enters the corresponding nickname in a Telex message, the corresponding information in this file is automatically retrieved and entered into the Telex header area.

telex header area. The first part of the telex message. It contains control information for the telex network.

telex interface program (TXIP). A program that runs on a Telex front-end computer and provides a communication facility to connect MERVA ESA with the Telex network.

Telex Link. The MERVA ESA component used to link to the public telex network via a Telex substation.

Telex substation. A unit comprised of the following:

- Telex Interface Program
- A Telex front-end computer
- A Telex box

Terminal User Control Block (TUCB). A control block containing terminal-specific and user-specific information used for processing messages for display devices such as screen and printers.

test key. A key added to a telex message to ensure message integrity and authorized delivery. The test key is an integer value of up to 16 digits, calculated manually or by a test-key processing program using the significant information in the message, such as amounts, currency codes, and the message date.

test-key processing program. A program that automatically calculates and verifies a test key. The Telex Link supports panels for input of test-key-related data and an interface for a test-key processing program.

TFD. Terminal feature definitions table.

TID. Terminal identification. The first 9 characters of a bank identifier code (BIC).

TOF. Originally the abbreviation of *tokenized form*, the TOF is a storage area where messages are stored so that their fields can be accessed directly by their field names and other index information.

TP. Transaction program.

transaction. A specific set of input data that triggers the running of a specific process or job; for example, a message destined for an application program.

transaction code. In IMS and CICS, an alphanumeric code that calls an IMS message processing program or a CICS transaction. Transaction codes have 4 characters in CICS and up to 8 characters in IMS.

Transmission Control Protocol/Internet Protocol (TCP/IP). A set of communication protocols that support peer-to-peer connectivity functions for both local and wide area networks.

transmission queue. In MQSeries, a local queue on which prepared messages destined for a remote queue manager are temporarily stored.

trigger event. In MQSeries, an event (such as a message arriving on a queue) that causes a queue manager to create a trigger message on an initiation queue.

trigger message. In MQSeries, a message that contains information about the program that a trigger monitor is to start.

trigger monitor. In MQSeries, a continuously-running application that serves one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. It uses the information in the trigger message to start a process that serves the queue on which a trigger event occurred.

triggering. In MQSeries, a facility that allows a queue manager to start an application automatically when predetermined conditions are satisfied.

TUCB. Terminal User Control Block.

TXIP. Telex interface program.

U

UMR. Unique message reference.

unique message reference (UMR). An optional feature of MERVA ESA that provides each message with a unique identifier the first time it is placed in a queue. It is composed of a MERVA ESA installation name, a sequence number, and a date and time stamp.

UNIT. A group of related literals or fields of an MCB definition, or both, enclosed by a DSLLUNIT and DSLLUEND macroinstruction.

UNIX System Services (USS). A component of OS/390, formerly called OpenEdition (OE), that creates a UNIX environment that conforms to the XPG4 UNIX 1995 specifications, and provides two open systems interfaces on the OS/390 operating system:

- An application program interface (API)
- An interactive shell interface

UN/EDIFACT. United Nations Standard for Electronic Data Interchange for Administration, Commerce and Transport.

USE. S.W.I.F.T. User Security Enhancements.

user file. A file containing information about all MERVA ESA users; for example, which functions each user is allowed to access. The user file is encrypted and can only be accessed by authorized persons.

user identification and verification. The acts of identifying and verifying a RACF-defined user to the system during logon or batch job processing. RACF identifies the user by the user ID and verifies the user by the password or operator identification card supplied during logon processing or the password supplied on a batch JOB statement.

USS. UNIX System Services.

V

verification. Checking to ensure that the contents of a message are correct. Two kinds of verification are:

- Visual verification: you read the message and confirm that you have done so
- Retype verification: you reenter the data to be verified

Virtual LU. An LU defined in MERVA Extended Connectivity for communication between MERVA and MERVA Extended Connectivity.

Virtual Storage Access Method (VSAM). An access method for direct or sequential processing of fixed and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry sequence), or by relative-record number.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

VSAM. Virtual Storage Access Method.

VTAM. Virtual Telecommunications Access Method (IBM licensed program).

W

Windows NT service. A type of Windows NT application that can run in the background of the Windows NT operating system even when no user is logged on. Typically, such a service has no user interaction and writes its output messages to the Windows NT event log.

X

X.25. An ISO standard for interface to packet switched communications services.

XCF. Abbreviation for *cross-system coupling facility*, which is a special logical partition that provides high-speed caching, list processing, and locking functions in a sysplex. XCF provides the MVS coupling services that allow authorized programs on MVS systems in a multisystem environment to communicate with (send data to and receive data from) authorized programs on other MVS systems.

XCF couple data sets. A data set that is created through the XCF couple data set format utility and, depending on its designated type, is shared by some or all of the MVS systems in a sysplex. It is accessed only by XCF and contains XCF-related data about the sysplex, systems, applications, groups, and members.

XCF group. The set of related members defined to SCF by a multisystem application in which members of the group can communicate with (send data to and receive data from) other members of the same group. All MERVA systems working together in a sysplex must pertain to the same XCF group.

XCF member. A specific function of a multisystem application that is defined to XCF and assigned to a group by the multisystem application. A member resides on one system in a sysplex and can use XCF services to communicate with other members of the same group.

Bibliography

MERVA ESA Publications

- *MERVA for ESA Version 4: Application Programming Interface Guide*, SH12-6374
- *MERVA for ESA Version 4: Advanced MERVA Link*, SH12-6390
- *MERVA for ESA Version 4: Concepts and Components*, SH12-6381
- *MERVA for ESA Version 4: Customization Guide*, SH12-6380
- *MERVA for ESA Version 4: Diagnosis Guide*, SH12-6382
- *MERVA for ESA Version 4: Installation Guide*, SH12-6378
- *MERVA for ESA Version 4: Licensed Program Specifications*, GH12-6373
- *MERVA for ESA Version 4: Macro Reference*, SH12-6377
- *MERVA for ESA Version 4: Messages and Codes*, SH12-6379
- *MERVA for ESA Version 4: Operations Guide*, SH12-6375
- *MERVA for ESA Version 4: System Programming Guide*, SH12-6366
- *MERVA for ESA Version 4: User's Guide*, SH12-6376

MERVA ESA Components Publications

- *MERVA Automatic Message Import/Export Facility: User's Guide*, SH12-6389
- *MERVA Connection/NT*, SH12-6339
- *MERVA Connection/400*, SH12-6340
- *MERVA Directory Services*, SH12-6367
- *MERVA Extended Connectivity: Installation and User's Guide*, SH12-6157
- *MERVA Message Processing Client for Windows NT: User's Guide*, SH12-6341
- *MERVA-MQI Attachment User's Guide*, SH12-6714
- *MERVA Traffic Reconciliation*, SH12-6392
- *MERVA USE: Administration Guide*, SH12-6338
- *MERVA USE & Branch for Windows NT: User's Guide*, SH12-6334
- *MERVA USE & Branch for Windows NT: Installation and Customization Guide*, SH12-6335
- *MERVA USE & Branch for Windows NT: Application Programming Guide*, SH12-6336
- *MERVA USE & Branch for Windows NT: Diagnosis Guide*, SH12-6337
- *MERVA USE & Branch for Windows NT: Migration Guide*, SH12-6393
- *MERVA USE & Branch for Windows NT: Installation and Customization Guide*, SH12-6335
- *MERVA Workstation Based Functions*, SH12-6383

Other IBM Publications

- *CICS/ESA V4.1 Problem Determination Guide*, SC33-1176
- *CICS/VSE Problem Determination Guide*, SC33-0716
- *IMS/ESA V5 Operator's Reference*, SC26-8030
- *OS/390 MVS Programming: Writing Servers for APPC/MVS*, GC28-1774
- *OS/390 MVS Programming: Writing Transaction Programs for APPC/MVS*, GC28-1775
- *VSE/ESA Guide for Solving Problems*, SC33-6510

S.W.I.F.T. Publications

The following are published by the Society for Worldwide Interbank Financial Telecommunication, s.c., in La Hulpe, Belgium:

- *S.W.I.F.T. User Handbook*
- *S.W.I.F.T. Dictionary*
- *S.W.I.F.T. FIN Security Guide*
- *S.W.I.F.T. Card Readers User Guide*

Index

MERVA Requirement Request

Use the form overleaf to send us requirement requests for the MERVA product. Fill in the blank lines with the information that we need to evaluate and implement your request. Provide also information about your hardware and software environments and about the MERVA release levels used in your environment.

Provide a detailed description of your requirement. If you are requesting a new function, describe in full what you want that function to do. If you are requesting that a function be changed, briefly describe how the function works currently, followed by how you are requesting that it should work.

If you are a customer, provide us with the appropriate contacts in your organization to discuss the proposal and possible implementation alternatives.

If you are an IBM employee, include at least the name of one customer who has this requirement. Add the name and telephone number of the appropriate contacts in the customer's organization to discuss the proposal and possible implementation alternatives. If possible, send this requirement online to MERVAREQ at SDFVM1.

For comments on this book, use the form provided at the back of this publication.

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

Send the fax to:

```
To: MERVA Development, Dept. 5640              Fax Number: +49-7031-16-4881
    Attention: Gerhard Stubbe                  Internet address:
                                               mervareq@de.ibm.com
    IBM Deutschland Entwicklung GmbH
    Schoenaicher Str. 220
    D-71032 Boeblingen
    Germany
```

MERVA Requirement Request

```
To: MERVA Development, Dept. 5640                   Fax Number: +49-7031-16-4881
    Attention: Gerhard Strubbe                      Internet address:
                                                    mervareq@de.ibm.com
    IBM Deutschland Entwicklung GmbH
    Schoenaicher Str. 220
    D-71032 Boeblingen       Germany

Page 1 of ______
```

Customer's Name	______________________
Customer's Address	______________________
	______________________
	______________________
Customer's Telephone/Fax	______________________
Contact Person at Customer's Location Telephone/Fax	______________________
	______________________
MERVA Version/Release	______________________
Operating System Sub-System Version/Release	______________________
	______________________
Hardware	______________________
Requirement Description	______________________
	______________________
	______________________
	______________________
	______________________
	______________________
	______________________
Expected Benefits	______________________
	______________________
	______________________

Readers' Comments — We'd Like to Hear from You

MERVA for ESA
Diagnosis Guide
Version 4 Release 1

Publication No. SH12-6382-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

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

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
SH12-6382-01

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Information Development, Dept. 0446
Schoenaicher Strasse 220
71032 Boeblingen
Germany

Fold and Tape | **Please do not staple** | Fold and Tape

SH12-6382-01

Cut or Fold
Along Line

Program Number: 5648-B29

SH12-6382-01

Spine information:

IBM MERVA for ESA Diagnosis Guide Version 4 Release 1