MERVA for ESA

Concepts and Components

Version 4 Release 1

SH12-6381-01

MERVA for ESA

IBM

Concepts and Components

Version 4 Release 1

SH12-6381-01

Note!

Before using this information and the product it supports, be sure to read the general information under “Appendix H. Notices” on page 215.

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 introduces the concepts and components of 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 provides a general description of the functions, services, and utilities supplied.

This book is aimed at readers who want to obtain a general understanding of the MERVA ESA concepts of messages, queues, routing, message handling, and network links. You should read this book before starting to install or customize MERVA ESA.

Note: The term *CICS* is used to refer to CICS/ESA®, CICS Transaction Server (CICS TS), and CICS/VSE®. The term *IMS* is used to refer to IMS/VS and IMS/ESA®.

Prerequisites for Using This Book

If you want to use SWIFT Link, you should be familiar with the contents of the S.W.I.F.T. User Handbook, which is published by the Society for Worldwide Interbank Financial Telecommunication, s.c., in La Hulpe, Belgium.

If you are using Telex Link, this book assumes that you are familiar with telex terminology as defined in the documentation provided by your local PTT[1].

1. National Post and Telecommunication Authority (post, telegraph, telephone).

Summary of Changes

This edition reflects the following changes:

FMT/ESA can now use MERVA-MQI Attachment
Financial Message Transfer/ESA (FMT/ESA) can now use MERVA-MQI Attachment as well as MERVA Link ESA to transfer SWIFT messages between two MERVA ESA systems (see "Financial Message Transfer/ESA (FMT/ESA)" on page 143).

Part 1. MERVA ESA Concepts

Chapter 1. Introducing MERVA ESA

MERVA ESA is a *message processing* and *message communication* system that can be used in any business. It enables you to exchange messages within your business using internal networks, or with places outside your business using external networks.

A *message* is a set of information that is given from a sender to a receiver. The message contents and formats are agreed upon by the sender and the receiver. For example, the message may start with information about the sender, the receiver, and the message format. If an external network is used, the contents and the format can also be influenced by this network.

A predefined message format eases readability. For example, in international business relations a *formatted* message can be understood by people speaking different languages. Formatted messages can also be processed by application programs. This allows the processing of a large number of messages without manual intervention.

Message processing can consist of the following steps:
- Creating the message, either by a person or an application program
- Some intermediate processing steps, such as verification or authorization
- Sending to the receiver
- Confirming the receipt of the message
- Some processing after the message is sent
- Receiving the confirmation

Processing of received messages can consist of the following steps:
- Receiving the message
- Confirming the receipt to the sender
- Some intermediate processing steps
- Some manual or automatic processing

MERVA ESA lets you create messages either automatically (using application programs) or manually (at data-entry terminals). Additional processing steps can also be done either by application programs or at data-entry terminals. You can configure MERVA ESA so that it checks messages for formal correctness, or so that it prints messages on terminal or line printers.

To ensure safe message storage, MERVA ESA provides a queuing system that:
- Allows step-by-step message processing
- Routes messages from one processing step to the next
- Automatically invokes application programs if required by the next processing step
- Prevents losing or duplicating messages

MERVA ESA provides several ways for you to distribute messages:
- Use the queuing system to distribute messages within a single MERVA ESA installation.

- Use MERVA Link to distribute messages between several MERVA installations. These can be within your own business (for example, among branch offices), or among several businesses that have agreed to share a private network.
- Exchange SWIFT messages among applications running in different MERVA ESA installations without using the SWIFT network. This is done by using the MERVA-to-MERVA Financial Message Transfer/ESA (FMT/ESA) together with MERVA Link or MERVA-MQI Attachment.
- Use SWIFT Link to connect to the SWIFT network using any number of communication lines. Use MERVA Extended Connectivity for communication lines that use X.25.
- Use Telex Link when exchanging messages with your branch offices or with your business partners via the public telex network.
- Use MERVA-MQI Attachment to exchange SWIFT, telex, or user-defined messages between MERVA ESA and other MERVA or non-MERVA installations where MQSeries® is available.
- Add other network connections.

MERVA ESA helps you with message presentation as follows:

- All SWIFT message formats are available for the lines to the SWIFT network and for display on screen terminals and printers.
- A free-format telex message is available for the public telex network. Information can be added to a SWIFT message for sending it via the public telex network. These messages can be formatted for the telex lines and for display on screen terminals and printers.
- The format definitions supplied with MERVA ESA can be modified, for example, to suit specific application programs.
- New messages can be defined, for example, for messages exchanged within a business or with business partners on “private” networks.

MERVA ESA also provides for:

- Defining what each user is allowed to do in MERVA ESA
- Journaling MERVA ESA activities
- Customizing MERVA ESA according to your environment

Chapter 2. Messages

This chapter explains the MERVA ESA message concept.

A *message* contains information that is given from a sender to a receiver. MERVA ESA assumes that messages have a defined structure. The structure allows the messages to be described to MERVA ESA and processed with the functions available in MERVA ESA. For this purpose, the structure is defined as a sequence of *fields*.

Different structures can be defined for different business purposes, and each structure is given an identifier called the *message type*.

A simple structure can contain fields for the:
- Sender of the message
- Receiver of the message
- Message type
- Fields of the message text

The MERVA ESA message concept is derived from the complex structure of SWIFT messages. All kinds of messages similar to SWIFT messages can be described and processed by MERVA ESA functions. Telex messages are an example of more simple messages. Messages used within your business or with your business partners can be as complex as required.

The Message Structure

The purpose of a message determines its structure. The structure of a message is the fields the message contains. The MERVA ESA message structure is shown in Figure 1 on page 6.

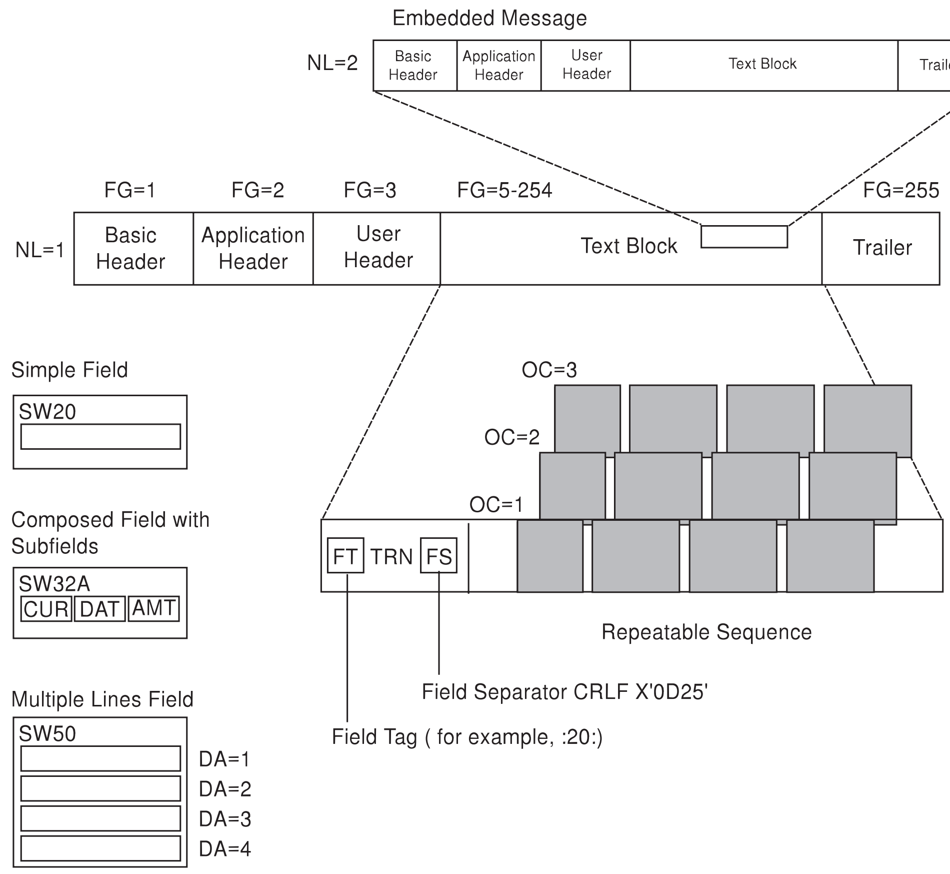


Figure 1. The MERVA ESA Message Structure

MERVA ESA supports the following messages:

- For SWIFT messages, S.W.I.F.T. has defined different structures identified by message types and a given sequence of fields.
- For telex messages, MERVA ESA has defined the structure of outgoing telex messages. However, incoming telex messages may not adhere to this structure and are treated as one field consisting of several lines.
- Messages used within your business or with business partners are defined by you and your business partners.

Message fields have the following attributes in MERVA ESA:

- They are separated from each other by *field separators*. In MERVA ESA, the field separators must be defined in the message description, for example, SWIFT and telex messages use the Carriage Return/Line Feed characters (hexadecimal 0D25) as field separators.
- They can have an identifier to recognize them in the sequence of fields in a message. This identifier is the *field tag*. If a field tag is used, the length and contents of the field can be variable. If field tags are not used, the field must have a fixed length or be ended by a field separator, and the field must always be in the same place in the message.

  A field tag can include an option to indicate variable field contents, for example, the SWIFT field 32 (Amount).
- Individual fields can have the following attributes:
  - A length. The length can be fixed or variable, and has a minimum and maximum value.

  - A data type, which can be numeric, alphabetic, or other.
  - Mandatory or optional fields. Optional fields need a field tag to indicate their presence or absence.
  - Different format options. Based on the format option, a field can appear within the same message type in different formats.
  - A number of data lines. These are called *data areas* in MERVA ESA and are identified by a *data area index*.
- Fields can be divided into subfields with either a fixed or variable structure.
- Fields can be combined into sequences that can be repeated. These sequences are called *repeatable sequences*. Each repetition of the field sequence is called an *occurrence*. MERVA ESA allows for the definition of several repeatable sequences in a message, and the definition of another repeatable sequence within a repeatable sequence. The latter is called a nested repeatable sequence.
- The format specifications of several message types can be combined within one message. For example, SWIFT messages provide for queries about other SWIFT messages. The query is a message type that can include a copy of the message that is the subject of the query.

Message Formats

Messages can be of either of the following formats:

Internal The internal formats are:

Tokenized form (TOF)

To handle the complexity of SWIFT messages, MERVA ESA has introduced an internal format in a specialized message buffer called *tokenized form* (TOF). In this buffer, the message fields are stored individually and accessed using MERVA ESA services. A compressed version of the TOF format is used to store messages in the MERVA ESA queues (queue format).

The MERVA ESA message buffer and tokenized field buffer are shown in Figure 2 on page 8.

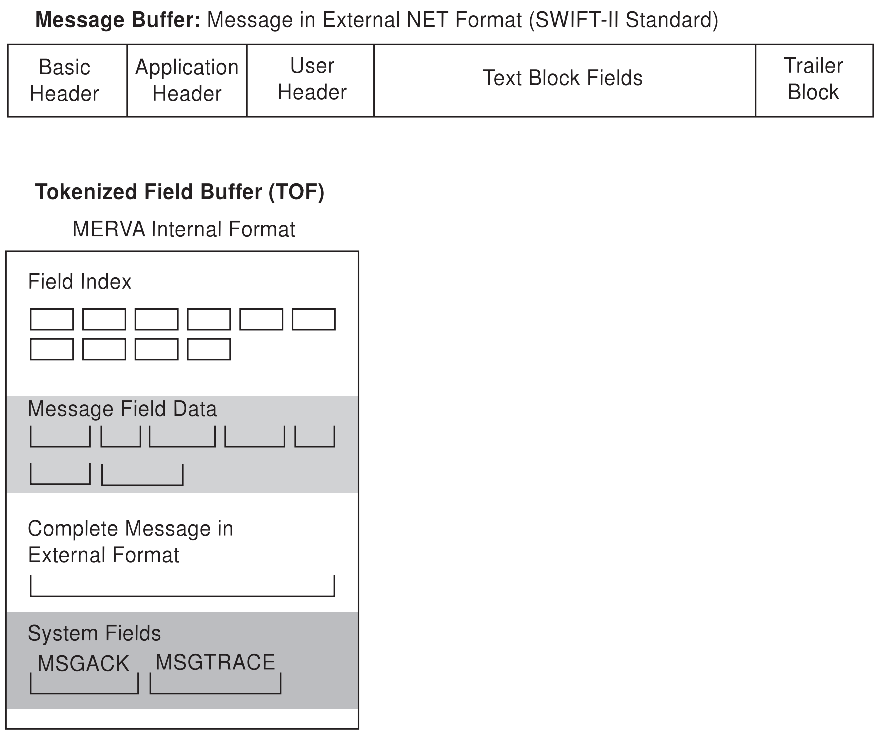


Figure 2. The MERVA ESA Message Format

External line format (ELF)
A message can be stored in external line format in a MERVA ESA queue. The message is not tokenized, but is stored in the external line format as data of a specific field. This format is more compact than the usual tokenized format and the processing of these messages can be faster. SWIFT Link supports this external line format.

Some restrictions apply when the external line format is used, for example, the messages cannot be checked for contents or formatting errors.

External
An external message format is needed for the presentation of messages on:

- Display stations
- Hardcopy printers
- System printers (line printers)
- Lines to external networks
- Sequential data sets
- Application program buffers

The transformation from the internal to the required external formats and the reverse is performed by MERVA ESA.

Defining Messages

Defining messages in MERVA ESA consists of:

- Defining the fields of a message, and their attributes in the MERVA ESA field definition table (FDT)
- Defining the message formats:
  - The sequence of the fields for the TOF format
  - The presentation of the fields for the external message formats (there can be several different formats for display stations, printers, and external networks for the same message type)

  The internal and external formats are defined in MERVA ESA message control blocks (MCBs)

Chapter 3. Functions and Queues

The MERVA ESA message processing concept consists of:

- Message-processing steps called *functions*
- Intermediate storing of messages between processing steps in *queues*
- Message flow between steps called *routing*

Functions, queues, and routing reflect the organization of your business. They are explained in more detail in the following.

Functions

MERVA ESA message-processing takes place in individual steps, for example, preparing a message for sending to the SWIFT network can involve:

- Creating the message by a person or an application program
- Verifying the message
- Authorizing the message
- Sending the message to the SWIFT network
- Processing the acknowledgment received from the SWIFT network

A sample message processing flow is shown in Figure 3.

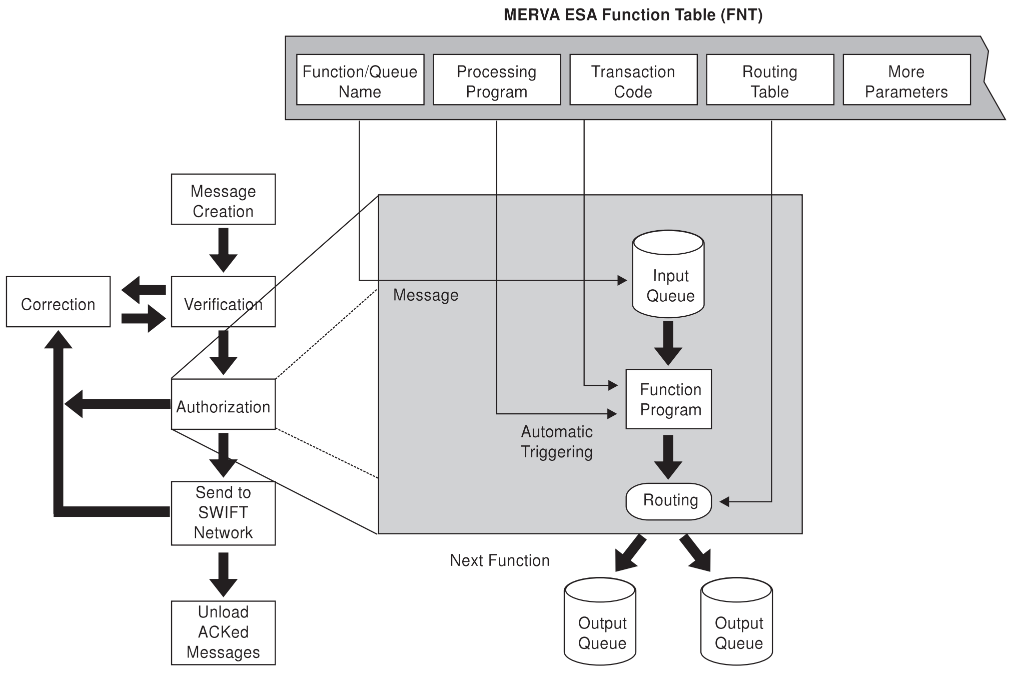


Figure 3. Sample Message Processing Flow

Each step is called a message-processing function in MERVA ESA. There are similar message-processing functions for receiving messages. These functions are performed by persons or application programs, and your business defines the person or program responsible for each message-processing step.

Other MERVA ESA functions that are not directly part of message processing must also be assigned to persons or programs, for example, online file maintenance.

All the functions needed in a MERVA ESA installation are defined in the MERVA ESA function table (FNT).

Queues

A message-processing function is usually associated with a MERVA ESA queue. The queues allow:

- The safe storing of messages between processing steps (this means that the steps need not be done immediately one after the other)
- Asynchronous message processing of steps working at different speeds (for example, several users can create messages while one user verifies or authorizes them)

A single message-processing step could involve:

1. Getting the message from the associated queue
2. Processing the message
3. Routing the message to the queue of the next message-processing step

Messages can be retrieved from the queue either sequentially in the order stored, or directly using a key.

When the next processing step of a message is to be performed by an application program, MERVA ESA can invoke the program automatically.

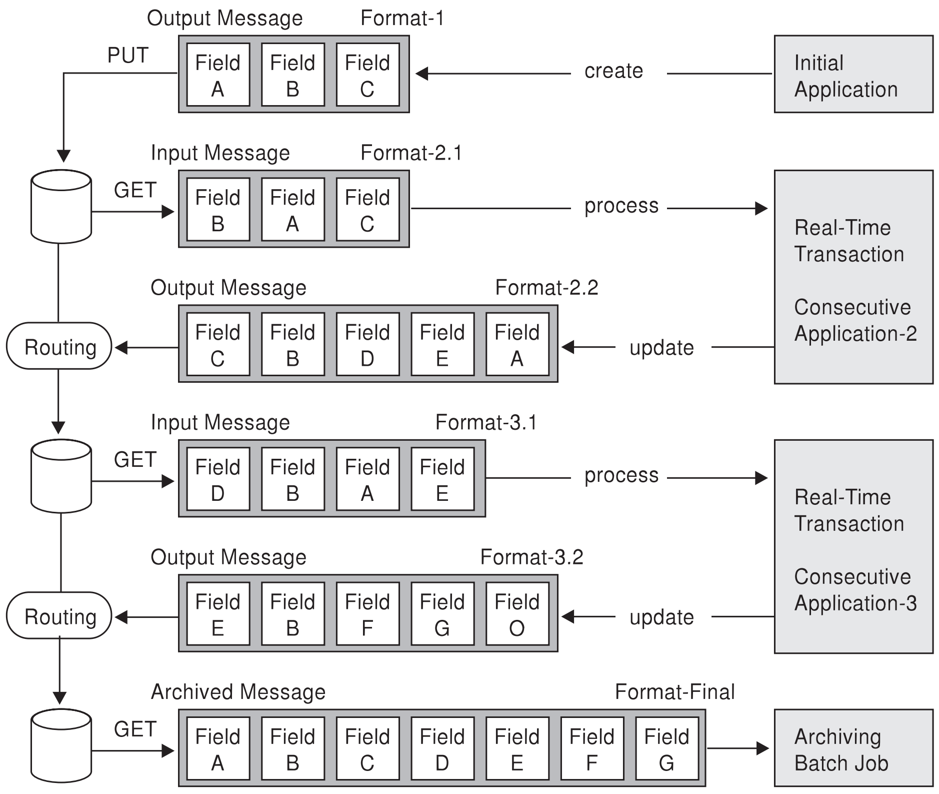


Figure 4. Interapplication Messaging in MERVA ESA

All function queues are contained in the MERVA ESA queue data set, and MERVA ESA ensures that messages are not duplicated or lost.

MERVA ESA programs usually expect the MERVA ESA TOF format when retrieving messages from queues, and provide the same format when storing messages in queues. To save storage space, the TOF format is compressed. This compressed message format is called the *queue format*.

Some MERVA ESA programs store and retrieve information in a format different from the MERVA ESA queue format. This information can not be processed by the other MERVA ESA programs, that is, this information can not be shown on display stations or hardcopy printers.

Routing

Routing determines the flow or path of messages from one message-processing function to the next. That means, after a message has been processed by a function, routing does the following:

- Determines the next message-processing step or function
- Stores the message in the queue for the next function
- If the message was retrieved from a queue, deletes it from this queue

The next step or function can be:

- Fixed.

  Messages processed by a given function are stored in the same queue for the same next processing step.
- The result of a routing criteria evaluation.

  In this case, the messages processed by a function can be stored in different queues for the next function, and they can be stored in up to twelve queues simultaneously (for example, for the next processing function, a print function, and for keeping copies of the messages for other purposes).

  The routing criteria are defined in *routing modules* that allow inspection of the contents of the message. This requires the message being in the MERVA ESA queue format. These routing modules are created during the customization of MERVA ESA.

  Each message-processing function can use a different routing module, or similar message-processing functions can use the same routing module.
- The result of a decision of a person using a display station and a MERVA ESA command to route the message.

  In this case the messages processed by a function can be stored in different queues. The decision made by the person can be controlled by a routing module.

Routing determines the path of a message through the MERVA ESA message-processing functions from the creation by a person or an application program until processing is complete. Finally, the message leaves MERVA ESA when it is deleted, for example, either after having been saved in a database, after printing, or when it is not needed anymore. A message does not leave MERVA ESA when it is sent to an external network, as there is still a copy of the message in a MERVA ESA queue.

Chapter 4. Message Processing

The term *message processing* refers to message creation, intermediate processing steps, and finally message deletion. When processing messages, the following interfaces are used:

- Host-based display stations for end users can be used for message creation, intermediate steps, and message deletion.
- MERVA Message Processing Client workstations can be used for message creation, manipulation, and deletion.
- Hardcopy and system printers. Usually messages are deleted after printing.
- Lines to external networks, such as SWIFT and telex. Sending messages is considered as an intermediate step (as the messages remain in a MERVA ESA queue), receiving messages is message creation.
- Sequential data sets. Input from sequential data sets is message creation, output to sequential data sets optionally deletes the messages.
- Application programs. Input from application programs is message creation, output to application programs can be an intermediate step (if the messages are routed after this step) or deletion (if the messages are deleted after this step).

Each interface requires a specific presentation format of a message. A message, for example, created by a user at a display station, shows the individual fields with explanatory text like field names, heading information, and diagnosis messages. On an external network line, however, the same message can be just one string of data. For each message type, all of its formats are defined in one MERVA ESA message control block (MCB).

End-User Interface

The end-user interface (DSLEUD) processes messages on a display station and performs the following functions:

- Message-processing functions for message creation, intermediate steps, and message deletion. The end-user interface performs the necessary accesses to MERVA ESA queues, checks the message contents, and invokes routing.
- Online file maintenance functions for the:
  - User file
  - General files, such as a private nickname file, the SWIFT currency code file, or the Telex correspondents file (an installation can define and add its own correspondent file, for example, for a national clearing network, under control of the MERVA ESA General File Program)
  - Authenticator-key file

  The end-user interface performs the necessary accesses to these files.
- Operator functions for controlling MERVA ESA and the external network links.

The user communicates with the end-user interface by means of commands. You can add your own functions and commands to the end-user interface.

MERVA Message Processing Client Server

The MERVA Message Processing Client Server supports online users at MERVA Message Processing Client workstations. Workstation users can, depending on permissions defined in MERVA ESA:

- Sign on and sign off to MERVA ESA
- Create messages
- Inspect, authenticate, and correct messages
- Route messages
- Delete messages

Hardcopy Print

The hardcopy printer program (DSLHCP) reads messages from a MERVA ESA queue, transforms them into the printing format, and prints them on a hardcopy (terminal) printer. The messages can be deleted or kept in the MERVA ESA queue after printing.

Sequential Data Set Batch Interfaces

MERVA ESA provides the following batch interfaces for sequential data sets (SDS):

- Input (DSLSDI and DSLSDIR)

  A batch of messages is read from a sequential data set, transformed into MERVA ESA TOF format, and put or routed to the relevant queue(s). The format of the messages in the sequential data set can be:

  – An external network format
  – The MERVA ESA queue format
  – A format defined by your installation

  Batch input can be used to transfer messages from another database or computer system into MERVA ESA for further processing.

- Load (DSLSDLR)

  A batch of messages is read from a sequential data set, transformed into MERVA ESA TOF format, and put back to the relevant queues. The format of the messages in the sequential data set can be:

  – An external network format
  – The MERVA ESA queue format

  Batch load can be used to restore messages into MERVA ESA preserving the queue name, QSN, key values, and the *write back* indicator.

- Output (DSLSDO and DSLSDOR)

  A batch of messages is read from a MERVA ESA queue, transformed into the external format, and written to a sequential data set. The format of the messages in the sequential data set can be:

  – An external network format
  – The MERVA ESA queue format
  – A format defined by your installation

  The messages can be deleted or kept in the MERVA ESA queue after the sequential data set is complete.

Batch output can be used to transfer messages from MERVA ESA to another database or computer system for further processing.

- Unload (DSLSDUR)

  A batch of messages is read from all or specified MERVA ESA queues, transformed into the external format, and written to a sequential data set. The format of the messages in the sequential data set can be:

  - An external network format
  - The MERVA ESA queue format (recommended)

  Batch unload can be used to back up messages to a data set preserving the queue name, QSN, key values, and the *write back* indicator.

- System printer (DSLSDY and DSLSDYR)

  A batch of messages is read from a MERVA ESA queue, transformed into the printing format, and printed on a system printer. The messages can be deleted or kept in the MERVA ESA queue after printing.

User-Written Programs

MERVA ESA functionality can be extended with user-written programs:

- The external interfaces supplied by MERVA ESA can be adapted to specific needs by user exit programs.
- User-written application programs can create messages, process intermediate steps, or further process messages and finally delete them from MERVA ESA queues.
- User-written application programs can work as administrative or maintenance functions that do not process messages.
- Interfaces to other external networks can be added to MERVA ESA.

Chapter 5. Communication Links

MERVA ESA provides the following communication links:

- SWIFT Link for the SWIFT network
- Telex Link
- MERVA Link ESA
- MERVA Link USS
- MERVA-MQI Attachment

SWIFT Link

SWIFT Link provides:

- Connection to the SWIFT network on one or more lines. MERVA ESA supports communication lines to SWIFT using X.25 as defined by S.W.I.F.T.

  For X.25 lines to SWIFT, MERVA ESA uses the product MERVA Extended Connectivity running on a 37xx communication controller. Connection between MERVA ESA SWIFT Link and MERVA Extended Connectivity is implemented by standard SNA sessions. MERVA Extended Connectivity controls the X.25 communication lines to SWIFT; for each communication line up to 10 switched virtual circuits (SVC) are supported. It is possible to connect several MERVA ESA installations to a single MERVA Extended Connectivity running on a 37xx communication controller. One MERVA ESA installation may use up to 30 lines, connections to MERVA Extended Connectivity products running on different communication controllers.
- A set of operator commands for communication with the SWIFT network. With these commands, events like login, logout, select, quit and so on are solicited by a MERVA ESA operator. SWIFT Link performs all the actions required by the protocol of the SWIFT network.

  For login and select, the Secure Login/Select (SLS) functions of the SWIFT User Security Enhancements (USE) are supported.
- To increase the throughput, MERVA ESA allows the customization and use of multiple SWIFT Link servers in parallel. SWIFT Link servers run independently of each other, but use the same MERVA ESA resources like queue data set and journal. Each SWIFT Link server has its own logical terminal table, defining the set of logical terminals processed by the server. The operator commands must be prefixed by a specific SWIFT Link routing command code to identify the requested SWIFT Link server.

  This concept of parallel SWIFT Link servers is useful in specific cases only. The installation should use multiple lines and several logical terminals each with a relatively high message volume. Otherwise the required overhead negates the possible performance improvement. The MERVA ESA nucleus should run as a native batch job in its own region to get full control over the available resources. When running the nucleus in a CICS region, the use of parallel SWIFT Link servers is not recommended.
- Field definitions for all SWIFT message fields.
- MCBs for all SWIFT messages:
  - The format for the SWIFT network
  - The formats for display stations, hardcopy printers, system printers, and sequential data sets

- Formal checking for all SWIFT message fields.
- The SWIFT Correspondents File, created from the SWIFT Bank Identifier Code (BIC) Directory update tape, and used for expanding SWIFT addresses. That is, for display stations and printers, the SWIFT bank identifier codes are expanded to the full name of the correspondent.
- The SWIFT currency code file, created from the SWIFT bank identifier code (BIC) directory update tape, and used for checking of currency code fields of SWIFT messages.
- The aAuthenticator-key file for authentication of sent and received messages. This file is maintained with:
  - The bilateral key exchange (BKE) functions of the SWIFT user security enhancements (USE)
  - A utility program
  - A MERVA ESA end user function

Messages sent to the SWIFT network are made available in MERVA ESA queues for further processing, together with the acknowledgments received from SWIFT.

Messages received from the SWIFT network are made available in MERVA ESA queues for further processing, together with the authentication result.

Telex Link

Telex Link lets you use the following Telex interface programs (TXIPs) as interfaces to the public telex network:
- On a PC, Telex Plus/2[2]
- On a fault-tolerant system, Headoffice Telex[2]

Telex Link provides:
- Field definitions for free-format and formatted telex messages
- MCBs for the telex messages:
  - The format for the telex network
  - The formats for display stations, hardcopy printers, system printers, and sequential data sets
- The Telex Correspondents File is created online using the MERVA ESA end-user interface, and is used for expanding telex correspondents addresses. That is, for display stations and printers, short (abbreviated) forms of the correspondents addresses are expanded to their full addresses.
- Interface to a test-key calculation program.

Telex Link via a Workstation

Telex Link via a workstation uses MERVA Link to connect MERVA ESA and MERVA USE & Branch.

Telex Link via a Fault-Tolerant System

Telex Link via a fault-tolerant system provides:
- Communication with the Telex Interface Program. The connection with the telex interface program is a VTAM® connection under control of CICS or IMS. telex interface program communicates with the public telex network.

2. “Telex Plus/2” and “Headoffice Telex” are IBM vendor logo products from *Intercope*, and must be ordered directly from *Intercope*.

- Telex Link performs all actions required by the protocol with the telex interface program. Messages received from the telex interface program are made available in MERVA ESA queues for further processing.
- Messages that are sent to the telex interface program are made available in MERVA ESA queues for further processing when the telex interface program acknowledges the successful transmission via the public telex network.

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.

MERVA Link ESA

MERVA Link ESA allows communication between several MERVA installations and with a MERVA Link USS gateway using Systems Network Architecture (SNA) connections. MERVA Link ESA uses a special Message Integrity Protocol (MIP) to ensure that no message is delivered twice to the receiving application.

MERVA Link ESA provides a partner table to define:

- MERVA ESA installations that communicate with each other
- Rules and communication protocols between two partners, for example, message text encryption and authentication
- Formats of messages exchanged between two partners (either external network formats or the MERVA ESA queue format)

Under normal circumstances, MERVA Link ESA works automatically without operator intervention. The MERVA System Control Facility (MSC) is provided to supervise processing or to restart processing after failures.

MERVA Link USS

MERVA Link USS provides a gateway function to route MERVA Link conversations from an SNA APPC to a TCP/IP network, and vice versa. A MERVA installation that supports only TCP/IP connections can use the services of a MERVA Link USS gateway to communicate with MERVA Link ESA (that supports only SNA APPC connections).

A MERVA Link USS installation is a MERVA Link node of its own. It is not associated with a particular MERVA installation, and cannot deliver messages to a MERVA messaging application. It can, however, provide network routing services to all nodes in a network of interconnected MERVA systems.

Under normal circumstances, MERVA Link USS works automatically without operator intervention. The MERVA Link USS Application Control facility (ACC) is provided in the OS/390 UNIX System Services environment to supervize processing and to assist in problem analysis.

MERVA-MQI Attachment

MERVA-MQI Attachment enables communication between MERVA ESA and MQSeries for MVS/ESA™ or MQSeries for VSE/ESA™.

When sending and receiving messages, MERVA-MQI Attachment uses the facilities of MQSeries to ensure the integrity of the data transfer.

MERVA-MQI Attachment provides a process table to define the characteristics of one or more send processes and receive processes. The message format can be either an external network format or the MERVA ESA queue format.

MERVA-MQI Attachment works automatically without operator intervention. If required, standard MERVA ESA operator commands can be issued to control the processing of MERVA-MQI Attachment.

Chapter 6. Control and Services

MERVA ESA provides services to all its components. These services fall into two categories:

- Central services
- Direct services

These services are explained in the following.

Central Services

Some MERVA ESA resources must be controlled by the MERVA ESA nucleus for two reasons:

Integrity	For example, access to the same message in a queue from two programs or signon with the same user identification from two display stations must be prevented.
Timing	For example, when a quick reaction time is necessary, the resources must be immediately available for a processing program. This means that these resources must not be used by other programs at this time.

The services that access resources controlled by the nucleus are called *central services*. The central services are:

DSLQMGT	Queue management service used to access messages in the MERVA ESA queues
DSLNUSR	User file service used to access records of the MERVA ESA user file
DSLNCS	Operator command service used to execute MERVA ESA operator commands
DSLNMOP	Operator message service used to issue unsolicited messages to the MERVA ESA operators
DSLJRNP	Journal service used to put a record into or to get a record from the MERVA ESA journal
DSLNRTCP	Remote task communication (MVS only) used to communicate with remote tasks
DWSAUTP	Authentication service used to: • Access the records of the authenticator-key file • Authenticate SWIFT messages

Direct Services

Other MERVA ESA resources are available at any time. The services accessing these resources are called *direct services*. Direct services are:

- Message format services (MFS) for the transformation of messages from the MERVA ESA internal tokenized format (TOF) to an external format and reverse, and for formal message checking
- TOF services, for access to the message fields in the MERVA ESA TOF

- Service to retrieve information or diagnostic messages for operators and end users from a table of message texts
- Write-to-operator service for issuing messages to the MERVA ESA operators
- System service that allows programs to request services from the operating system or from CICS (such services are, for example, loading modules, obtaining main storage, and getting various formats of the system date and time)
- General file service that let programs access the MERVA ESA general files

Under MVS™, with queue management using DB2®, batch and API programs can call queue management services directly.

Use of the Services

How to use the MERVA ESA services depends on the relation of the calling program to the MERVA ESA nucleus:

- If the calling program is included in the MERVA ESA nucleus, it can use all services directly. Such programs, for example, are the:
  - Communication part of SWIFT Link
  - Communication part of Telex Link via a fault-tolerant system
  - User-written programs included in the nucleus
- Calling programs that are not included in the MERVA ESA nucleus must request the central services via the MERVA ESA intertask communication. Such programs, for example, are the:
  - End-user driver
  - Hardcopy print program
  - SDS batch programs
  - MERVA Link
  - User-written programs not included in the nucleus

All MERVA ESA services are described in detail in “Part 2. MERVA ESA Components” on page 25.

Part 2. MERVA ESA Components

Chapter 7. Control Facilities

Controlling MERVA ESA has the following aspects:

- Operating MERVA ESA:
  - Starting and stopping MERVA ESA
  - Supervising the processing of MERVA ESA

  The operator functions of the MERVA ESA nucleus DSLNUC provide for this type of control
- Controlling the programs defined in the MERVA ESA nucleus program table (DSLNPTT), including the MERVA ESA intertask communication for accessing the central services
- The timer service of the MERVA ESA nucleus

The MERVA ESA Nucleus (DSLNUC)

DSLNUC is the main program of MERVA ESA. To start, stop, and run MERVA ESA means to start, stop, and run DSLNUC.

The major functions of the MERVA ESA nucleus (DSLNUC) and its components are:

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

The MERVA ESA nucleus can run in different ways, depending on the system environment:

- Under CICS, DSLNUC is started as a CICS transaction using:
  - DSLCMO (master operator program)
  - DSLCAS (automatic start program)
- Under IMS, DSLNUC can be started as an IMS batch message processing program (BMP).
- Under MVS, DSLNUC can be started as a native batch program. In this case the services of CICS and IMS are not available directly. The connection to the transactions running under CICS and IMS is nevertheless possible. Transactions under CICS are started using the External CICS Interface (EXCI), transactions under IMS are started using APPC/MVS.

Refer to the *MERVA for ESA Operations Guide* for details about how to start DSLNUC.

Components of DSLNUC

The main program DSLNUC and the services it provides are table driven. There are three types of services, each type being defined by a separate table:

nucleus program table (DSLNPTT)
DSLNPTT defines the programs that are controlled by external resources and external or internal events. The external resources can be communication lines, or a system operator console. The events can be

created outside of MERVA ESA, for example, an interrupt for a communication line, or can be internal events, for example, when a message is stored in a MERVA ESA queue. The programs currently used in MERVA ESA are:

- The remote task communication program DSLNRTCP (MVS only)
- The operator interface program DSLNMOP for the operating system console
- The task servers for the MERVA ESA intertask communication:
  - DSLNTS is the traditional method using CICS memory copy, MVS SVC, or VSE XPCC
  - DSLNTSQ using CICS temporary storage queues
  - DSLNTSA using APPC/MVS as communication vehicle
  - DSLNTSM using MQSeries queues
- The program DWSDGPA for connecting to the SWIFT network (several SWIFT Link servers can be used running under one MERVA ESA nucleus)
- The program DWSAUTIN for initializing the SWIFT authentication
- The program DWSDLSK for loading pregenerated session keys for the SWIFT Secure Login/Select (SLS) into MERVA ESA queues
- The station program ENLSTP of the Telex Link via a fault-tolerant system
- The synchronization point program DSLISYNP to be used under IMS and CICS
- The message counter program DSLCNTP

Other programs can be coded by a system programmer and added by the system administrator.

nucleus task server request table (DSLNTRT)
DSLNTRT defines the programs that provide central services to other MERVA ESA programs and applications. A central service is executed when its service is required by another program. The central services are described in "Central Services" on page 23.

nucleus command table (DSLNCMT)
DSLNCMT defines the command codes and the execution programs for MERVA ESA nucleus. Whenever an operator or an application program enters a command, the table is scanned to find the service program belonging to that command code.

The main program DSLNUC, the tables described above, and the execution programs defined for each individual service are link-edited together and build the nucleus load module DSLNUC.

An overview of the nucleus structure is shown in Figure 5 on page 29.

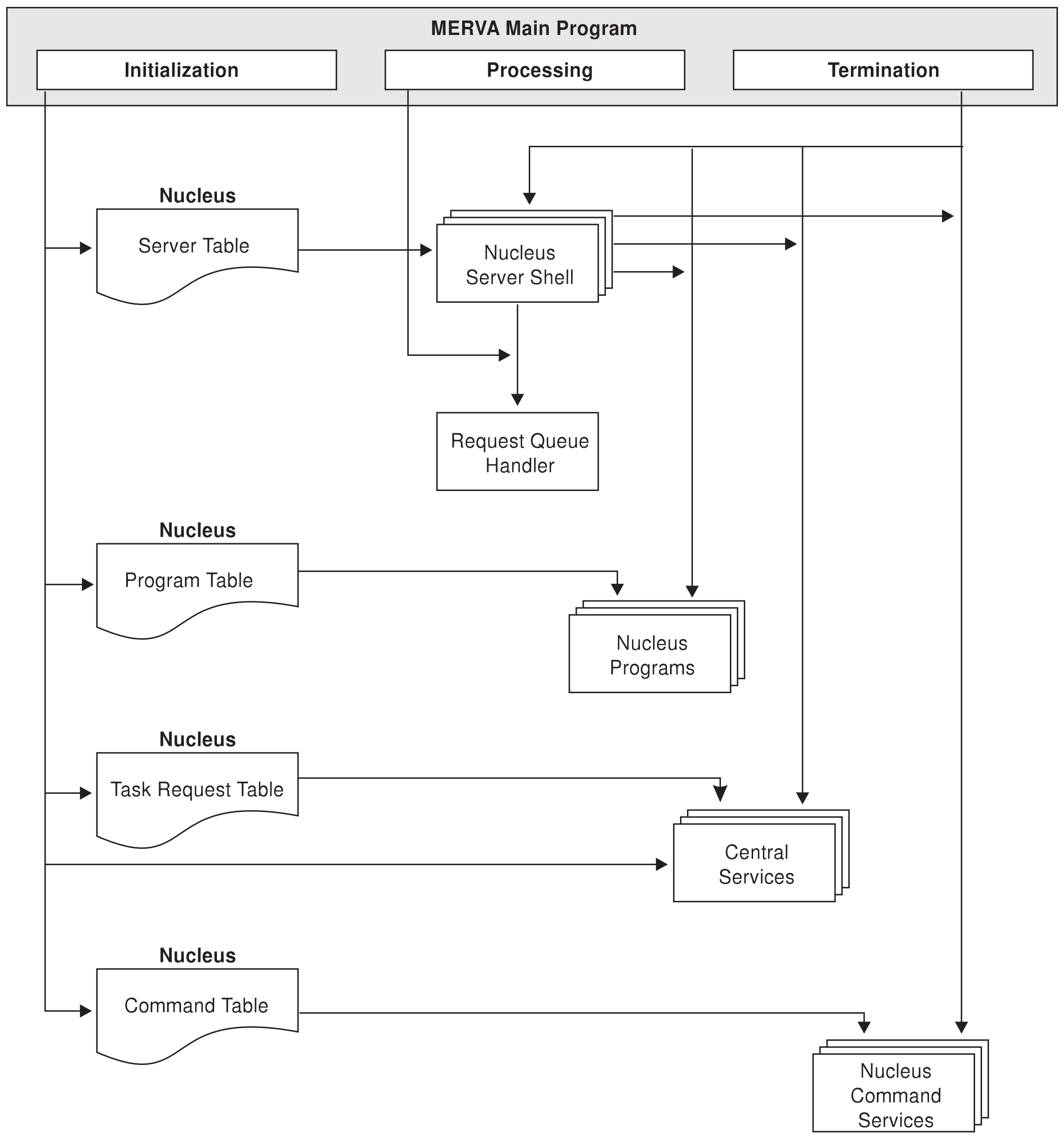


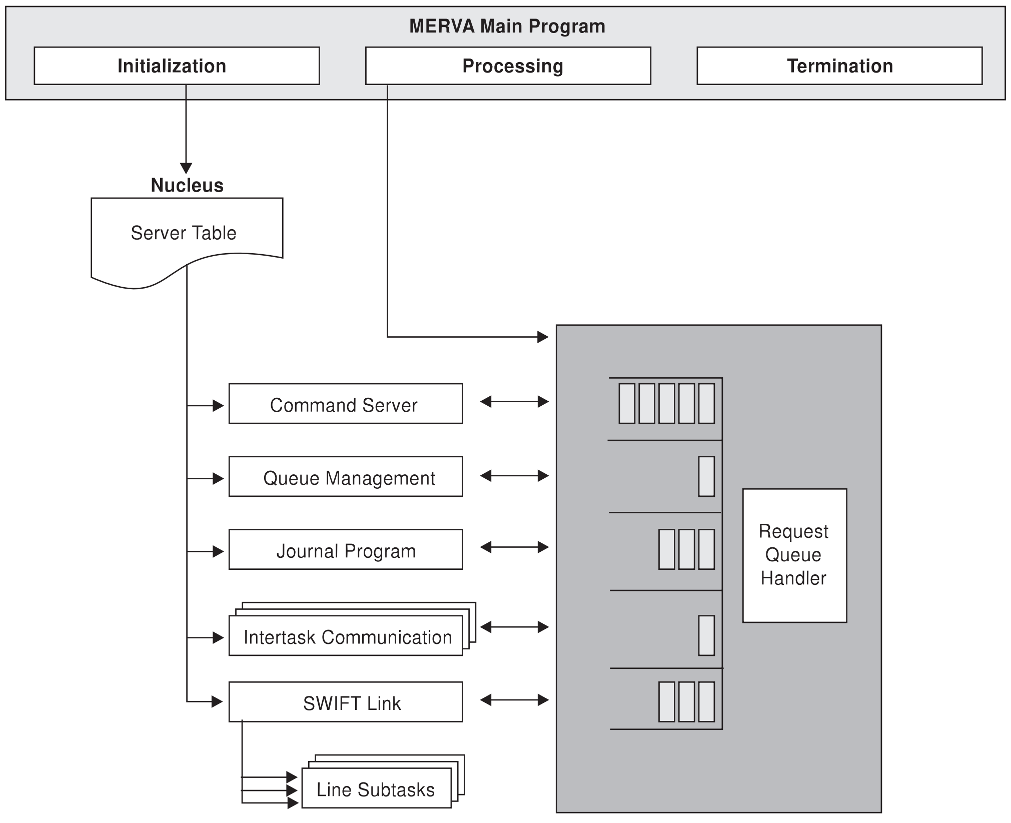
Figure 5. The MERVA ESA Components: Nucleus Main Structure

All nucleus services run in one address space. MERVA ESA allows selected services to be run as a separate task. This is defined in the nucleus server table (DSLNSVT). When there are no definitions in the nucleus server table, or all services are defined as running under direct control of DSLNUC, they are processed directly and synchronously under DSLNUC, thus, there is no parallel processing.

In MERVA ESA, a service can be defined to be processed in parallel. In this case, a service is executed under control of a nucleus server shell program that runs as a separate MVS subtask (DSLNSHEL) or as a CICS task (DSLNSHEC) in the same address space as the main task DSLNUC.

When multiple processors can be used, the service programs will be executed in parallel. When only one processor is available the above concept has the advantage that an overlap of multiple I/O operations is possible with the execution of one service.

The communication between services running as subtasks is performed by the request queue handler. The communication between services running in different MERVA ESA instances is performed by the MERVA ESA MQSeries nucleus server using the interservice communication facility. For more information, refer to "Interservice Communication" on page 45.



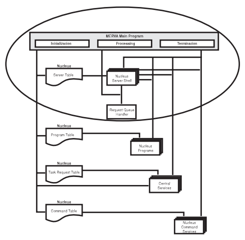


Figure 6. DSLNUC with Parallel Servers

Nucleus Server Shell

The nucleus server shell is the interface between the nucleus and the services performed by the nucleus servers running as separate tasks such as:

- Central services
- Task servers
- Network drivers

It consists of the nucleus server shell main module and the processors for the following events:

- Request Ready
- Service Processed
- Posted Program
- Request Processed

There are two types of nucleus server shells:

- DSLNSHEL, which runs as an MVS subtask invoked by DSLNUC via an ATTACH macro
- DSLNSHEC, which runs as a CICS task invoked by DSLNUC via the CICS START service

The nucleus server shell and the services it invokes is called a nucleus server. The services a nucleus server has to perform must be specified in the nucleus server table (DSLNSVT). Once a nucleus server is attached and initialized, it begins handling the following events:

nucleus server termination
This event is signaled by the nucleus when it is normally or abnormally terminated.

timer expiration
This event is signaled when a timer that was set by a program linked to DSLNUC expires or is canceled.

request ready
A nucleus server starts performing a service when it receives a service request from the request queue handler, signaled as a Request Ready event. This means, DSLNUC or another nucleus server has added a service request to the request queue for this nucleus server. The service request is obtained from the request queue for this nucleus server and analyzed. Depending on the service request contents, the appropriate service is then invoked. This can be:

- A start (initialization) or stop (termination) request for a program
- The execution of a central service such as a queue management or a journal service
- The execution of a command service
- A nucleus task service

Any invoked service in turn can subsequently invoke also one or more other services. If the invoked service decides to process the request asynchronously, control is returned immediately to the nucleus server shell to allow processing of other requests.

On synchronous request processing, the invoked service returns control to the nucleus server shell if all subsequent services have finished. In this case the request postprocessor is invoked.

The request postprocessor notifies the request queue handler that the obtained service request has finished, and the service processed event is signaled to the nucleus server that obtained and processed the service.

posted program
This event is signaled when an ECB is posted for an application program link-edited to DSLNUC. These programs are defined in the nucleus

program table. In the entry for such a program, the number of ECBs associated with the program is also defined. The program is invoked with the address of the posted ECB and processes the application depending on the ECB function.

service processed
This event is signaled by the nucleus server shell after completion of the service request it has obtained. If such an event is detected, the request processed event is signaled to the nucleus server that created the service request that was just processed.

request processed
This signal is awaited by the nucleus server shell of the program that created and added a service request onto the request queue of a nucleus server that was processed asynchronously. The added service request is deleted from the request queue. The request postprocessor is then invoked.

The request postprocessor notifies the request queue handler that the obtained service request has finished, and the service processed event is signaled to the nucleus server that obtained and processed the service.

A service request can be created and added to the request queue by any application program link-edited to DSLNUC or by any central service executed under the DSLNUC maintask or under a nucleus server subtask. All information or references to information is stored in it so that services running under DSLNUC or another nucleus server can access that information. Note that the nucleus server that created a service request must also delete it from the request queue.

General Request Queue Handler (DSLNRQH) Functions

The request queue handler is the common component to queue and schedule service requests created by a nucleus server or the nucleus.

If a service is needed by the nucleus or a nucleus server that is not available there, a service request is created. The service request is set up in the form of a request control element and handed over to the request queue handler by calling the *ADD* queuing function. The request queue handler then selects the request queue for the nucleus server that provides the requested service, fetches an element from the free element pool, inserts the service request, chains it to the queue of *waiting* requests, and signals the request ready event.

Each time the nucleus or a nucleus server detects a request ready event, a new service request is ready for processing. It is handed over by the request queue handler when the *OBTAIN* queuing function is called. The request is then moved from the queue of waiting requests to the queue of *active* requests. The request contents is analyzed and the appropriate service is executed.

When the service request has finished processing, it is signaled to the request queue handler by calling the *NOTIFY* queuing function. The request is then moved from the queue of active requests to the queue of *finished* requests.

The service that created a service request must also delete it again. This is signaled to the request queue handler by calling the *DELETE* queuing function. The request is then moved from the queue of finished requests into the free element pool.

The above sequence is controlled by the request queue handler following the rules of a finite state machine:

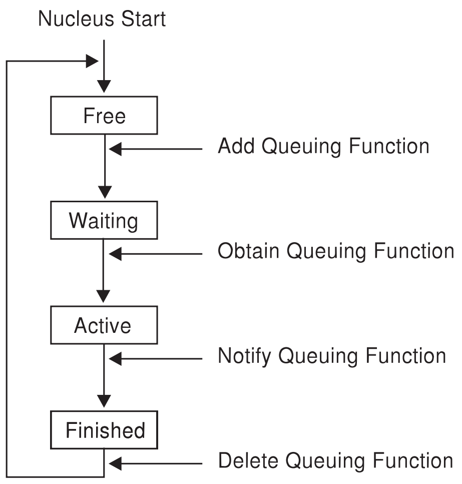


Figure 7. Request Queue Handler (DSLNRQH) Finite State Machine Diagram

MERVA ESA Operator Functions

The MERVA ESA operator functions are:

- Starting MERVA ESA:
  - Under CICS, MERVA ESA is started by the programs DSLCMO or DSLCAS (see "CICS Master Operator Program (DSLCMO)" on page 34 or "CICS Automatic Start Program (DSLCAS)" on page 34).
  - Under IMS, DSLNUC is started as an IMS BMP, either submitting a job to MVS or using an MVS start command for a cataloged procedure.
  - Under MVS, DSLNUC can be started as a native batch program. This works for both CICS and IMS installations.
- Processing of MERVA ESA operator commands. The operators can enter the MERVA ESA operator commands as follows:
  - Using the operating system console. The MERVA ESA operator interface program DSLNMOP must be started for this purpose. Under VSE, the program DSLCMO is used for the communication with the operating system console.
  - Using a display station and the MERVA ESA end-user driver (see "Operator Command Program DSLECMD" on page 99 and "MERVA System Control Facility Program EKAEMSC" on page 101 for more information).

  All operator commands of the base functions of MERVA ESA, SWIFT Link and Telex Link via a fault-tolerant system are defined in the nucleus command table DSLNCMT. User-written operator commands must also be added to this table.
- Processing unsolicited operator messages. The MERVA ESA operator interface program DSLNMOP and the MERVA ESA write-to-operator program DSLWTOP are used (see "The Operator Interface Program (DSLNMOP)" on page 34, and "The Write-to-Operator Program (DSLWTOP)" on page 35).

Starting MERVA ESA and entering MERVA ESA operator commands are described in detail in the *MERVA for ESA Operations Guide*.

CICS Master Operator Program (DSLCMO)

The master operator program (DSLCMO) is a task under CICS in both MVS and VSE and is used for:

- Starting MERVA ESA from a display station
- Starting MERVA ESA from the VSE system console
- Entering MERVA ESA operator commands at the VSE system console

To start the program DSLCMO, the operator enters the transaction code DSL (or another transaction name defined in CICS for the program DSLCMO) at a CICS display station, which can also be the operating system console under VSE.

DSLCMO accepts the attempt to start MERVA ESA (or for VSE, to enter a MERVA ESA operator command) only if the following items match:

- The identifier contained in the MERVA ESA customizing parameters (DSLPRM)
- The operator identification from the CICS signon

DSLCMO issues a CICS start command with the transaction code DSLN for DSLNUC. DSLNUC informs DSLCMO whether the startup was successful or not.

Under VSE, an operator command is given to DSLNMOP for command execution. DSLNMOP in turn gives the command response back to DSLCMO.

CICS Automatic Start Program (DSLCAS)

MERVA ESA can be started automatically during the CICS startup by using the program DSLCAS as described in the *MERVA for ESA Operations Guide*.

DSLCAS issues a CICS start command with the transaction code DSLN for DSLNUC. DSLNUC informs DSLCAS whether the startup was successful or not.

The Operator Interface Program (DSLNMOP)

The MERVA ESA operator interface program (DSLNMOP) operates under the MERVA ESA nucleus DSLNUC and has the following functions:

- Processing MERVA ESA operator commands issued from the operating system console.

  For this purpose, DSLNMOP is defined in the MERVA ESA nucleus program table (DSLNPTT) with the descriptive name CONSOLE. Processing of MERVA ESA operator commands from the system console is only possible if DSLNMOP, that is, CONSOLE, is active in DSLNPTT. The communication with the MERVA ESA operator is done as follows:

  – Under MVS, DSLNMOP issues an MVS WTOR macro with the message:

    ```
    DSL001A Enter a MERVA command
    ```

    The MVS reply command is used to enter the MERVA ESA command. DSLNMOP processes the command, displays the response and another message DSL001A at the operating system console to allow more command input.

  – Under VSE, DSLNMOP gets the command input from DSLCMO. DSLNMOP processes the command and returns the response to DSLCMO.

- Displaying unsolicited operator messages.

  Unsolicited operator messages are messages issued by a MERVA ESA program at the operating system console to inform the MERVA ESA operators about how MERVA ESA is running.

Unsolicited operator messages can be issued by:

– Programs linked to the MERVA ESA nucleus, using DSLNMOP directly.
– Programs not linked to the MERVA ESA nucleus, using DSLNMOP as a central service. The message is issued at the operating system console for the region or partition in which DSLNUC operates.

Unsolicited operator messages are written to:

– Under VSE, the VSE system console using a VSE PUT macro
– Under MVS, the MVS system console using an MVS WTO macro

DSLNMOP makes the unsolicited operator messages available for the MERVA ESA **dm** (display message) command and the MERVA ESA journal. This technique enables an operator to monitor the processing of, for example, the MERVA ESA batch programs DSLSDI, DSLSDO, and DSLSDY from a MERVA ESA display station, and to see their action in the MERVA ESA journal.

The number of unsolicited operator messages saved for the **dm** command can be specified in the MERVA ESA customizing parameters (DSLPRM).

Under MVS, for modification of routing and descriptor codes, DSLNMOP calls the user exit DSLWTOEX before using the WTO macro. Refer to the MERVA for ESA Customization Guide for more information on this user exit.

The Write-to-Operator Program (DSLWTOP)

The MERVA ESA write-to-operator program, called DSLWTOP, issues unsolicited operator messages at the operating system console. DSLWTOP is used by programs operating in a region or partition other than DSLNUC. The message is issued at the operating system console for the region or partition in which the program operates:

- Under VSE, a PUT macro is used to write the message to the console.
- Under MVS, a WTO macro is used to issue the message at the console. DSLWTOP calls the user exit DSLWTOEX for modification of routing and descriptor codes before using the WTO macro. Refer to the *MERVA for ESA Customization Guide* for more information on this user exit.

If the MERVA ESA intertask communication is established for the program calling DSLWTOP, the unsolicited operator messages are made available for the MERVA ESA **dm** (display message) command and the MERVA ESA journal using DSLNMOP as a central service.

MERVA ESA Functions Using a Security Manager

MERVA ESA can use the services of a security manager to control the following functions:

- Signon to MERVA ESA
- Signon to the user file maintenance function within MERVA ESA

The security manager can be an external security manager (ESM) or the basic security manager (BSM) introduced with VSE/ESA Version 2.4. The security manager must comply to the System Authorization Facility (SAF). Under MVS, RACF® is an example for an SAF compliant ESM.

The following scenario illustrates how a security manager can control a signon for MERVA ESA:

- A signon to CICS or IMS is made entering the user ID and password. The user ID and the password are recorded by the security manager.
- Entering the transaction code DSLP starts the MERVA ESA end user driver.
- If the user is authorized, the MERVA ESA function selection panel is displayed. Otherwise, the MERVA ESA signon is rejected and the MERVA ESA signoff panel is displayed with an error message.

  A user is authorized to sign on to MERVA ESA if a MERVA ESA user file record exists for the user ID entered at CICS or IMS signon.
- When the user file maintenance function is selected from the MERVA ESA function selection panel, a password must be entered. It is checked against the security manager that the password is identical to the password entered at CICS or IMS signon.
- When the MERVA ESA user file is empty, the user ID defined in DSLPRM must be used. If the user is authorized, the MERVA ESA user file maintenance panel is displayed. Otherwise, the MERVA ESA signon is rejected and the MERVA ESA signoff panel is displayed with an error message.

  In the MERVA ESA customization the user must be authorized by setting its user ID to a security manager recorded user ID that is entered at CICS or IMS signon.

User-Written Application Programs

MERVA ESA applications are programs that use the MERVA ESA API or MERVA ESA macros to request services from MERVA ESA. These programs can be written in Assembler language, high-level languages such as COBOL, PL/I, or C/370, or in REXX. The MERVA ESA API is described in *MERVA for ESA Application Programming Interface Guide*.

The application programs can run as CICS transactions or IMS MPPs. In this case they can be started in various ways:
- From a CICS or IMS terminal by a user
- By MERVA ESA automatically, triggered by a queue event
- By a MERVA ESA operator

The application programs can also be run as batch programs running in the same MVS or VSE as the MERVA ESA nucleus.

In MERVA ESA it is possible to run MERVA ESA application programs in an MVS image different from the one where MERVA ESA is running, when both systems are connected via APPC/MVS or MQSeries. In this case the intertask communication using one of these services can be used for communication between the MERVA ESA nucleus and the application programs.

An application program coded as a CICS or IMS transaction, as an APPC/MVS transaction program, or as a batch program, accesses the MERVA ESA central services via the MERVA ESA intertask communication.

There is a specific class of MERVA ESA applications that is different from the applications described above and for which certain restrictions apply. This type of applications is described in the next section.

Application Programs Link-Edited to DSLNUC

Application programs that are linked to DSLNUC must be coded in Assembler language. They start to operate only when specific events occur. For example, the

IBM-supplied communication network link SWIFT Link is implemented as this type of application. An appropriate entry in the nucleus program table (DSLNPTT) link-edits an application program to DSLNUC. See the *MERVA for ESA Customization Guide* for the description of the interface for such application programs. User-written programs should only be implemented under the direct control of the nucleus, if one of the following requirements apply:

- The program is running permanently (for example, a SWIFT Link program serving the line to the SWIFT network), and needs information permanently in storage.
- The program needs intervention by operator commands; for example, if it needs the MERVA ESA operator functions.
- The program supplies services that are used as central services.

Programs for which none of the above requirements applies should be written as separate applications running as transaction or batch programs using the MERVA ESA API.

An application program link-edited to DSLNUC has three distinct processing steps:

1. **Initialization**

   A MERVA ESA *start* command is entered for the program. The program obtains main storage, opens data sets, and loads modules for its processing part. Upon return, the program gives the addresses of its event control blocks (ECBs) to DSLNUC. MERVA ESA programs can also be started automatically at MERVA ESA startup.

2. **Processing**

   One or more of the program''s ECBs are posted. The program processes the task associated with the posted ECB.

3. **Termination**.

   A MERVA ESA *stop* command is entered for the program. The program frees all acquired storage, closes its data sets, and deletes all loaded modules so as not to occupy resources needed by other programs. DSLNUC discards the ECB addresses of the program.

An application program controlled by a DSLNPTT entry can run as a subtask. In this case the program must be defined in the nucleus server table as well. All programs that use the same resources or call each other directly must be defined in DSLNSVT to run under the same nucleus server shell.

The sample program DSLBN11A is an example for a nucleus application program. When installed, this sample program can be started and stopped under operator control. It executes MERVA ESA commands repeatedly after a specified time interval. The program source code can be found in the MERVA ESA sample library.

Intertask Communication

Product-Sensitive Programming Interface

The intertask communication facility lets application programs communicate with DSLNUC to request a central service. Intertask communication consists of two parts:

- The requester’s side. This is the application program requesting a service from MERVA ESA. The interface program DSLNICT is used to request a service via intertask communication.
- The server’s side or DSLNUC. This part is processed by one of the task servers of DSLNUC. A task server can run under direct control of DSLNUC or as a subtask under control of a nucleus server shell. It is given control by DSLNUC or the nucleus server shell whenever a requester posts one of the ECBs provided for the intertask communication.

For the communication between the requester’s and server’s sides, several MERVA ESA interface programs are used, depending on the type of intertask communication.

There are several methods for conducting MERVA ESA intertask communication. The value of the ITC parameter in the customization module DSLPRM determines which method is used:

- Requesters running in the same partition or region as DSLNUC use CICS intraregion communication. This is possible only when MERVA ESA is running under CICS and the application programs are CICS transactions.

  The intraregion communication requires a DSLNPTT entry of TYPE=INTRA. This entry specifies the task server DSLNTS as executing program, and defines the server event control blocks (ECBs) for intraregion communication. The intraregion communication control blocks (ICBs) are allocated in an appendix of the DSLNPTT. Actually, the ECB specification determines the number of ICBs allocated in the system, that is the maximum number of parallel communication sessions. For performance reasons, DSLNTS uses a single ECB to handle all allocated ICBs. The data exchange is done directly between the buffers of the requester and the server.

  If necessary, DSLNTS dynamically allocates larger buffers to store the data provided by the requester or server. DSLNICT allocates larger buffers if the data does not fit into the provided buffers and the dynamic buffer option is specified. The maximum size of a buffer is specified in the MAXBUF parameter of the customization module DSLPRM.

  The control blocks are anchored in the CICS CWA. The direct memory transfer method requires that the application programs run in the same key as DSLNUC (CICSKEY) and cannot use the transaction isolation and storage protection feature of CICS.
- Programs running under CICS and requiring CICS storage protection use CICS temporary storage queue communication. The method of communication is through a CICS temporary storage queue and a separate interface program DSLNICQ, which runs with affinity to DSLNUC. The application program itself can run isolated from DSLNUC.
- For requesters running on the same system as DSLNUC but in a partition or region where DSLNUC does not run, an interregion communication takes place. This facility can be used by your application programs that are running in another region or partition:
  - Batch programs
  - CICS transactions running in a CICS region that is not the CICS region where DSLNUC is started
  - Programs running in an IMS MPP region
  - Programs running in an APPC/MVS region

The interregion communication requires a DSLNPTT entry of TYPE=INTER. This entry specifies the task server DSLNTS as executing program, and defines the server event control blocks (ECBs) for interregion communication.

- If requesters and the server run on different systems, two methods for intertask communication are available:
  1. APPC/MVS communication can be used by applications running on any machine within an SNA network connected to an MVS where MERVA ESA is running, provided that the appropriate environment such as load libraries and files are available there.

     When this service is used MERVA ESA registers with APPC/MVS to serve the inbound requests from MERVA ESA clients. MERVA ESA nucleus works as an APPC/MVS server as described in *MVS/ESA SP V5 Writing Servers for APPC/MVS*. A NOSCHED LU must be defined, which is used by the MERVA ESA intertask communication.

     It is important that MERVA ESA runs in an address space that allows to register as a server for the NOSCHED LU. Where this is not possible, for example, when the MERVA ESA nucleus runs in an IMS BMP, a separate batch program performing the MERVA ESA APPC/MVS server functions must be started.

     The batch program DSLNTSAB works as a switch between the APPC/MVS server functions and the traditional interregion communication via SVC.
  2. MQSeries communication can be used by applications running on any machine connected to an MVS where MERVA ESA is running, provided that MQSeries is installed and the appropriate environment such as load libraries and files are available there.

When the allocation request does not work for one of the new intertask communication methods, MERVA ESA tries to communicate via the traditional intra- or interregion communication method. Therefore, it is recommended that you install the MERVA ESA SVC, because this enables interregion communication for MVS.

The following service programs are used to perform the MERVA ESA intertask communication:

DSLNICT	Interface program for the requester
DSLNICTA	Subprogram for APPC/MVS method
DSLNICTM	Subprogram for MQSeries method
DSLNICTQ	Subprogram for CICS TS queue method
DSLNICP	Interregion communication program for VSE (XPCC)
DSLNICPM	Interregion communication program for MVS (SVC)
DSLXSVCX	Interregion and sysplex communication program for MVS
DSLNICQ	Interface to DSLNTSQ for CICS TS queue method
DSLNTS	Task server program (inter- and intraregion)
DSLNTSA	Task server program for APPC/MVS method
DSLNTSAB	APPC/MVS server interface running in a separate address space
DSLNTSM	Task server program for MQSeries method
DSLNTSQ	Task server program for CICS TS queue method

DSLNICT—Intertask Communication Interface

Intertask communication is called with the nucleus intertask communication macro DSLNIC, which invokes the intertask communication interface DSLNICT. DSLNICT decides which type of intertask communication is used. For:

- APPC/MVS, the subprogram DSLNICTA is used (see "DSLNICTA—Intertask Communication Interface for APPC/MVS")
- MQSeries, the subprogram DSLNICTM is used (see "DSLNICTM—Intertask Communication Interface for MQSeries" on page 41)
- CICS TS queue, the subprogram DSLNICTQ is used (see "DSLNICTQ—Intertask Communication Interface for CICS TS" on page 41)

DSLNICT carries out all functions required to exchange information between the requesting task and the MERVA ESA task server. These functions are:

- Allocate an intertask communication block (ICB). This function must be called before a service can be requested. The DSLNIC parameter list contains the information about the ICB. For all subsequent DSLNIC requests, the same parameter list must be used.
- Request a service. This function consists of several steps, executed by DSLNICT:
  – Moving the data to the server's side
  – Waiting for completion of the service request
  – Checking if the request is serviced, or if MERVA ESA has been terminated during the wait
  – Moving data to the requester's storage
- Free an intertask communication block (ICB). This function must be called when the application program terminates. If the application fails to free the ICB, some resources might be blocked. After the free call, no further service requests are allowed.

Data exchange consists of one or two data buffers whose maximum lengths are defined in the customizing parameters (DSLPRM). A service request can require one of the two buffers, or both. DSLNICT allocates larger buffers dynamically if the data does not fit into the provided buffers and the dynamic buffer option is specified.

If the requester and the server operate in different MVS regions or VSE partitions, some of the functions carried out are transferred to the interregion communication program (DSLNICP or DSLNICPM).

DSLNICTA—Intertask Communication Interface for APPC/MVS

If the method of the intertask communication is via APPC/MVS, this subprogram is called by DSLNICT.

DSLNICTA carries out all functions required to exchange information between the requesting task and the MERVA ESA task server. These functions are:

- Allocate an intertask communication block (ICB). This function must be called before a service can be requested. The DSLNIC parameter list contains the information about the ICB. For all subsequent DSLNIC requests, the same parameter list must be used. DSLNICTA uses CPI-C functions to communicate with MERVA ESA. An allocate function is executed to allocate a session to the MERVA ESA task server.
- Request a service. This function consists of several steps, executed by DSLNICTA:

  - Sending the data using the CPI-C send function. The data is sent in three or more chunks. A data buffer containing more than 32KB is segmented.
  - Receiving the resulting data using the CPI-C receive function.
- Free an intertask communication block (ICB). This function must be called when the application program terminates. The CPI-C function DEALLOCATE is executed by DSLNICTA. If the application fails to free the ICB, some resources might be blocked. After the free call, no further service requests are allowed.

DSLNICTM—Intertask Communication Interface for MQSeries

If the method of the intertask communication is via MQSeries, this subprogram is called by DSLNICT.

DSLNICTM carries out all functions required to exchange information between the requesting task and the MERVA ESA task server. DSLNICTM uses MQSeries queues to transport the data from the requester to the server. The names of the queues used in both directions are specified in the customization module DSLPRM. The functions supported by DSLNICTM are:

- Allocate an intertask communication block (ICB). This function must be called before a service can be requested. The DSLNIC parameter list contains the information about the ICB. For all subsequent DSLNIC requests, the same parameter list must be used.
- Request a service. The services of the MQSeries Queue Handler DSLNMQH are used to transfer the parameter list and the data buffer.
- Free an intertask communication block (ICB). At the same time the application is disconnected from MQSeries.

DSLNICTQ—Intertask Communication Interface for CICS TS

If the method of intertask communication is via CICS temporary storage queue, this subprogram is called by DSLNICT.

DSLNICTQ carries out all functions required to exchange information between the requesting task and the MERVA ESA task server. These functions are:

- Allocating an intertask communication block (ICB). This function must be called before a service can be requested. The DSLNIC parameter list contains the information about the ICB. For all subsequent DSLNIC requests, the same parameter list must be used.
- Requesting a service. This function consists of several steps, executed by DSLNICTQ:
  - Storing the data in a CICS temporary storage queue
  - Using EXEC CICS LINK to call the program DSLNICQ, which runs with affinity to the task server program (DSLNUC)
  - On return from DSLNICQ, reading the the result data from the CICS temporary storage queue and moving it to the requester’s storage
- Free an intertask communication block (ICB). This function must be called when the application program terminates. If the application fails to free the ICB, some resources might be blocked. After the free call, no further service requests are allowed.

DSLNICP—Interregion Communication Program for VSE

The interpartition communication program DSLNICP uses the VSE cross-partition communication control (XPCC). DSLNICP is loaded by both the requester and the nucleus. A name defined in the customizing parameters DSLPRM identifies the server's side and the requester's side.

DSLNICP carries out the following major functions in supporting the communication between a requester and a server that operate in two different partitions or regions:

- Communication control
- Data exchange

Communication Control

Communication control is performed using the SEND/RECEIVE facility of XPCC. The function is controlled by the event control blocks in the XPCC control blocks (XPCCB):

- Requester ECBs. The three XPCC ECBs are used for connect, receive, and send.
- Server ECB. The service request is controlled only by the receive ECB.

Data Exchange

This function is carried out via the interregion communication area (DSLICA).

To pass information between a requester and the server, DSLNICP uses the interregion communication control block (ICB). All ICBs required to support the communication between several requesters and the server are allocated in DSLICA.

The buffers needed for the data exchange are also allocated in DSLICA. All storage needed for the communication is obtained using VSE GETVIS macros. For each requester, one set of buffers is needed in the requester's partition, and another set of buffers is needed in the server's partition.

All data coming from the requester is:

1. Copied into the data buffers of the requester's XPCCB
2. Given to XPCC for transferring to the server's buffers of DSLICA
3. Copied into the server's storage before the request is serviced

Data coming from the server is:

1. Copied into the server's data buffers of DSLICA
2. Given to XPCC for transferring to the data buffers of the requester's XPCCB
3. Copied into the requester's storage area

If one of the buffers is too small to contain all data to be copied, a larger buffer is obtained dynamically using the VSE GETVIS macro. The maximum size of a buffer is specified in the MAXBUF parameter in the customization parameters DSLPRM.

DSLNICPM—Interregion Communication Program for MVS

The interregion communication program DSLNICPM is implemented as a type-3 SVC routine. The SVC number is specified in the customizing parameters (DSLPRM) and can be changed without a new MERVA ESA generation. The actual name of the interregion communication program in a MERVA ESA installation is determined by the SVC number.

DSLNICPM uses the interregion communication area (DSLICA) for the communication. Depending on the specifications in the customizing parameters DSLPRM, DSLNICPM saves the address of DSLICA in one of the following ways:

- In a subsystem entry of the subsystem facility of MVS (SSCT)
- In an extension table, the address of which is stored in the CVTUSER field of the MVS common vector table (CVT)

The MVS requirements described in the *MERVA for ESA Installation Guide* provide more information about the use of the MVS subsystem facility or of the CVTUSER field. DSLNICPM carries out the following major functions in supporting the communication between a requester and a server that operate in two different partitions or regions:

- Communication control
- Data exchange

Communication Control

This function is performed using the WAIT/POST facility and controlled by two event control blocks (ECBs):

- Requester ECB. This ECB is contained in the calling parameter list for DSLNICT.
- Server ECB. This ECB is defined by the TYPE=INTER entry of DSLNPTT and is owned by DSLNTS.

Data Exchange

This function is carried out via the interregion communication area (DSLICA).

To pass information between a requester and the server, DSLNICPM uses the interregion communication control block (ICB). All ICBs required to support the communication between several requesters and the server are allocated in DSLICA.

The buffers needed for the data exchange are also allocated in DSLICA. The storage needed for the communication is obtained using MVS GETMAIN macros for the non-fetch protected subpool 241 and the fetch-protected subpool 231. One set of data buffers is needed for each requester.

All data coming from the requester is copied into the data buffers of DSLICA. From there it is copied into the server’s storage before the request is serviced.

If one of DSLNICPM’s buffers is too small to contain all data coming from a requester, DSLNICPM dynamically allocates a larger buffer. The maximum size of a buffer is specified in the MAXBUF parameter in the customization parameters DSLPRM.

Data coming from the server is also copied into DSLICA. From here it is copied into the requester’s storage area.

DSLXSVCX—Interregion and Sysplex Communication Program for MVS

The interregion and sysplex communication program DSLXSVCX is implemented as a type-3 SVC routine.

The SVC number is specified in the customizing parameters (DSLPRM) and can be changed without a new MERVA ESA generation. The actual name of the communication program in a MERVA ESA installation is determined by the SVC number. The program DSLXSVCX is an extension of the program DSLNICPM and provides the services used for interregion and sysplex communication. The calls for interregion communication are compatible to MERVA ESA Version 3.

The extension is used to perform authorized XCF services to:
- Create the MERVA ESA XCF group and join an XCF member to the group (IXCCREAT)
- Leave the MERVA ESA XCF group (IXCLEAVE)
- Quiesce the MERVA ESA group (IXCQUIES)
- Set the current XCF member status (IXCSETUS)

It is also used to issue the MCGRE macro to internally start an address space for a MERVA ESA instance in another system within the sysplex. The parameter ISCXCF in the customizing parameter module (DSLPRM) is used to specify the sysplex environment. The sysplex services are only available for the MERVA ESA interservice communication.

DSLNICQ—Interface to DSLNTSQ

DSLNICQ is the interface program between DSLNICTQ and DSLNTSQ. DSLNICQ runs in the same CICS region and with the same storage key as DSLNUC. DSLNICQ is called by DSLNICTQ via **EXEC CICS LINK**, and the communication information is exchanged via the COMAREA. Because the MERVA plist and data buffer data is exchanged directly between DSLNICTQ and DSLNTSQ via CICS temporary storage queues, an application program that uses DSLNICTQ does not need to run in the same storage key as DSLNUC.

DSLNTS—Task Server

The task server (DSLNTS) is the interface between the central services and all application programs outside the nucleus when using interregion or intraregion communication.

DSLNTS is defined in the DSLNPTT with a DSLNPTT entry of TYPE=INTER for the interregion communication. Under CICS only, DSLNTS is defined with a DSLNPTT entry of TYPE=INTRA for the intraregion communication.

All central services are specified in the nucleus task server request table DSLNTRT. You can add your own central services to the DSLNTRT. Programs contained in the DSLNTRT must follow the interface specifications defined for central service programs, which are described in the *MERVA for ESA Customization Guide*.

DSLNTS initializes the intertask communication during the start of MERVA ESA, and terminates it during MERVA ESA termination.

DSLNTS is invoked for a central service request when one of its event control blocks (ECB) is posted by:
- DSLNICT for intraregion communication (see “DSLNICT—Intertask Communication Interface” on page 40)
- DSLNICP for interregion communication; under VSE, XPCC does the posting (see “DSLNICP—Interregion Communication Program for VSE” on page 42)

DSLNTS locates the requested service in DSLNTRT and invokes it. DSLNTS provides for data to be returned to the requester:
- DSLNTS posts the requester for completion for intraregion communication.

- Under MVS, DSLNTS invokes the MERVA ESA SVC for posting the requester for interregion communication.
- Under VSE, DSLNTS invokes DSLNICP for posting the requester for interregion communication; XPCC does the posting.

DSLNTSA—Task Server for APPC/MVS

The task server DSLNTSA is the interface between the central services and the APPC/MVS interface program (DSLNICTA) outside the nucleus (see "DSLNICTA—Intertask Communication Interface for APPC/MVS" on page 40). DSLNTSA registers as an APPC/MVS server. When the address space is prohibited to use the local LU, an error message is issued. In this case the batch program DSLNTSAB must be used.

DSLNTSA waits for an allocation request for its TPNAME. When a request is received the data is received from APPC/MVS into internal buffers of DSLNTSA. DSLNTSA calls the requested MERVA ESA central service. The resulting data is sent to the requester using the APPC/MVS send function.

DSLNTSAB—Batch Task Server Interface for APPC/MVS

The program DSLNTSAB serves as a switch between APPC/MVS and the MERVA ESA nucleus region in case the registration as an APPC/MVS server is prohibited. This applies, for example, to the MERVA ESA nucleus running in a BMP, at least under certain IMS versions. DSLNTSAB is a batch program that registers as an APPC/MVS server and forwards the requests to the MERVA ESA nucleus using the traditional interregion communication via SVC.

DSLNTSM—Task Server for MQSeries

The task server DSLNTSM is the interface between the central services and the intertask communication program DSLNICTM running on the requester's side (see "DSLNICTM—Intertask Communication Interface for MQSeries" on page 41). The services of the MQSeries queue handler DSLNMQH are used to transfer the parameter list and the data buffer.

DSLNTSQ—Task Server for a CICS TS Queue

The task server DSLNTSQ is the interface between the central services and the interface program DSLNICQ (see "DSLNICQ—Interface to DSLNTSQ" on page 44). DSLNTSQ is invoked by posting its event control block. DSLNTSQ reads the input data from a CICS temporary storage queue.

DSLNTSQ locates the requested service in DSLNTRT and invokes it. DSLNTSQ provides for data to be returned to the requester. The result data is written to the CICS temporary storage queue. Finally, the interface program DSLNICQ is posted.

End of Product-Sensitive Programming Interface

Interservice Communication

Interservice communication enables to spread nucleus services among several systems within a sysplex by defining MERVA ESA instances. This facility is only available for MVS/ESA and requires MQSeries to be installed and available on all systems.

There can be two or more MERVA ESA instances. One must be defined as a *primary*, all others as *secondary* MERVA ESA instances. Only the primary MERVA

ESA instance communicates with the requesters via the intertask communication and provides the operator interface. Before the primary MERVA ESA instance issues the MERVA ready message, the primary and all secondary MERVA ESA instances must have been started and initialized. The services a MERVA ESA instance provides is defined in the nucleus server table.

If a MERVA ESA instance requests a service that is provided by another MERVA ESA instance, the service request is forwarded to the local MERVA ESA MQSeries nucleus server. The local MERVA ESA MQSeries nucleus server communicates with the MERVA ESA MQSeries nucleus server on the remote MERVA ESA instance that provides the requested service. The remote MERVA ESA MQSeries nucleus server forwards the service request to the required nucleus server for being processed. This nucleus server returns the service response to the remote MERVA ESA MQSeries nucleus server to create a response message for being returned to the local MERVA ESA MQSeries nucleus server. The local MERVA ESA MQSeries nucleus server then updates the local service request, parameter and data buffers.

This communication between MERVA ESA instances is called interservice communication. Interservice communication is performed in each MERVA ESA instance by the MERVA ESA MQSeries nucleus server (DSLNMQS) using the MQSeries queue handler (DSLNMQH).

DSLNMQS—MERVA ESA MQSeries Nucleus Server

The MERVA ESA MQSeries nucleus server is an application program link-edited to DSLNUC and must therefore be defined in the nucleus program table. It is started automatically as the first service by DSLNUC: a program start request is added onto the request queue of the MERVA ESA MQSeries nucleus server. The main module of the MERVA ESA MQSeries nucleus server is DSLNMQS. The initialization, precessing, and termination of a MERVA ESA MQSeries nucleus server are described in the following sections.

Initialization

On a program start request, DSLNMQSS is invoked to:

- Analyze the MERVA ESA multisystem environment
- Connect to the local message queue manager
- Build the internal nucleus server map
- Build the program ECB list for the processing phase
- Initialize all message queues defined to the MERVA ESA MQSeries nucleus server
- Allocate the message buffers

The startup extension program DSLNMQSX is invoked to exchange information between the primary and all secondary MERVA ESA instances. At this time, all secondary MERVA ESA instances must have been started. If specified in DSLPRM, the primary MERVA ESA instance invokes program DSLXSVCS to start all secondary MERVA ESA instances automatically. If failure notification is specified in DSLPRM, the primary MERVA ESA instance waits a defined time until all secondary MERVA ESA instances have joined the defined MERVA ESA XCF group.

The following sequence is then continued:

- All MERVA ESA instances send their locally created nucleus server map to the primary MERVA ESA instance.
- The primary MERVA ESA instance consolidates and completes them with further information and builds the global nucleus server map.

- The global nucleus server map is sent to all secondary MERVA ESA instances.
- The global nucleus server map received by a secondary MERVA ESA instance replaces its local nucleus server map. This ensures that each MERVA ESA instance has the same information about all other MERVA ESA instances.
- All secondary instances send a server ready (SRVREADY) message to all MERVA ESA instances, which receive them.
- Invoke program DSLXSVCS, which registers to XCF.

Further processing is continued after the primary MERVA ESA instance has received all nucleus ready messages.

Each MERVA ESA instance needs the following message queue types to be defined:
- The RECEIVE queue. All MQI messages of type DATAGRAM and REQUEST are directed to this queue.
- The REPLY_TO queue. All MQI messages of type REPORT and REPLY are directed to this queue.
- The SEND queue. This is a locally defined remote message queue. There are as many send queues as there are communication partners. For example, if there are two secondary MERVA ESA instances, two send queues for each MERVA ESA instance are required. By defining MQI channels, a send queue is connected to a receive queue.

Processing

The processing of the MERVA ESA MQSeries nucleus server is driven both by internal, request-related events, and by external events contained in ECBs that have been posted to a queue. It performs asynchronous request processing to service multiple requesters in parallel.
- The internal, request-related events are:

  request ready

  If a nucleus program or central service needs another service that is not provided by a nucleus server in the local MERVA ESA instance, the created service request is redirected to the request queue of the MERVA ESA MQSeries nucleus server. Program DSLNMQSQ is invoked to create a service request message that contains buffers for control information, parameters, and data. This message is sent to the MERVA ESA instance where the nucleus server with the requested service runs.

  request processed

  If the requested service in the responding MERVA ESA instance has finished and has signaled this event to the MERVA ESA MQSeries nucleus server, program DSLNMQSR is invoked to create a request response message that contains buffers for control information, parameters, and data. This message is sent to the requesting MERVA ESA instance that runs the originating request.
- For external events, the following event control blocks (ECBs) must be defined for the MERVA ESA MQSeries nucleus server in the nucleus program table:
  - The first ECB is posted by MQSeries if a message has arrived on the receive queue.
  - The second ECB is posted by MQSeries if a message has arrived on the reply-to queue.
  - The third ECB is used for remote failure notification via XCF. It can be enabled by specifying it in the DSLPRM. The system environment change (SEC) ECB is posted by XCF if any status change of a MERVA ESA XCF group member is signaled.

Such external events are handled by program DSLNMQSE.

A message that arrives on the receive queue must be of type REQUEST, and therefore must contain a service request. In this case the MERVA ESA MQSeries nucleus server is at the responding side. The program DSLNMQSC is invoked to create a new service request for the server that provides the service. The new service request is asynchronously processed: if the service request has finished, the request processed (RP) event is signaled.

A message that arrives on the reply-to queue must be of type REPLY, and hence must contain a service response. In this case the MERVA ESA MQSeries nucleus server is at the requesting side. The program DSLNMQSC is invoked to update the originating service request and associated parameter and data buffers. An exception report message generated by MQSeries is also accepted.

If remote failure notification is specified in DSLPRM, all MERVA ESA instances have registered themselves to XCF during startup. Any abnormal status change is signaled to XCF, for example:

– The abnormal termination of a subtask providing a specific service within a MERVA ESA instance
– The abnormal termination of the entire MERVA ESA instance
– The abnormal termination of the system on which a MERVA ESA instance runs

This signal is propagated to all MERVA ESA instances causing the system environment change (SEC) to be recognized. This event causes termination of all MERVA ESA instances.

Termination

Termination is initiated:

- By DSLNUC:
  - Following a program stop request initiated by the operator
  - Following a termination request of the MERVA ESA MQSeries nucleus server
- By XCF, following a system environment change event

On a program stop request, program DSLNMQST is invoked at the primary MERVA ESA instance to send a nucleus termination message (NUCTERM) to all secondary MERVA ESA instances to initiate a program stop.

Program DSLNMQSP is invoked to:

- Deregister from XCF by invoking program DSLXSVCS
- Terminate all message queues
- Disconnect from the local message queue manager
- Free all allocated storage areas

Remote Failure Notification

Remote failure notification offers a way to bypass the message processing via MQSeries in case of a failure. It prevents messages from being put into a queue to be received by another MERVA ESA instance while this instance is no longer able to do so. Each MERVA ESA instance is immediately aware of a failure within the members of the MERVA ESA XCF group.

Each group member registers itself to XCF during startup and deregisters itself during normal and abnormal termination. On abnormal termination of a service subtask, a status change is passed to XCF. These services are provided via the DSLXSVCS program.

Intersystem Service Invokation (DSLXSVCS)

This program calls the SVC (DSLXSVCX) to perform the following authorized services:

- XCF services:
  - QUERY information about a specific or all members of the XCF group
  - JOIN a member to the XCF group
  - Update user state flag
  - LEAVE sets the state of the member from active to not-defined state
  - DELETE sets the state of the member to the not-defined state and from active to not defined
- Internal address space start

XCF Group Exit Routine (DSLXGRP)

This program will be activated when a MERVA ESA instance joins the XCF group. It runs as an system routine (SRB routine) and is scheduled by XCF to the address spaces of all MERVA ESA instances that have joined the MERVA ESA XCF group. The group exit gets control when a group member:

- Joins the group
- Leaves the group
- Changes the state from active to failed
- Sends an update of a user state

When the program gets control, it searches the member in the nucleus server map, copies the user data, and posts the system environment change (SEC) ECB in the nucleus server map entry associated with the MERVA ESA instance that caused the event.

MQSeries Queue Handler

The MQSeries queue handler provides the interface between MQSeries and the various MERVA ESA programs requesting MQI services. The MQSeries queue handler main module is DSLNMQH. Each MQI service is handled by a separate module:

DSLNMQSB	MQI BROWSE message on a local message queue
DSLNMQHC	MQI CONNECT to the local message queue manager (MQM)
DSLNMQHO	MQI OPEN a message queue
DSLNMQHQ	MQI INQUIRE attributes of a local message queue
DSLNMQHI	Message queue initialization
DSLNMQHG	MQI GET message from a local message queue
DSLNMQHS	MQI Set SIGNAL event establishment
DSLNMQHP	MQI PUT message onto a message queue
DSLNMQHT	Message queue termination
DSLNMQHL	MQI CLOSE a message queue

DSLNMQHD
MQI DISCONNECT from the local message queue manager

Any of the above services can be requested via the MQSeries queue handler parameter list. Data buffers are automatically adjusted to the required length. If the maximum message queue length is smaller than the buffer to send, message segmenting is performed. Control is returned if the entire message buffer is sent.

On the receiving side, length of the data buffer is adjusted to the one at the sending side. All message segments belonging together are received into the buffer. Control is returned if the entire message is received.

DSLNRTCP—Remote Task Communication

The remote task communication facility enables one application program to communicate with another application program using the Remote Task Communication program DSLNRTCP as a central service. This facility is only available in MVS.

The remote task communication consists of two parts:

- The receiving task. This task uses the Receiving Remote Task Program (DSLRRTCP) to make itself known to DSLNRTCP, and to retrieve an instruction from DSLNRTCP.
- The instructing task. This task uses DSLNICT to send an instruction to a receiving task.

The Receiving Task

The receiving task communicates with DSLNRTCP using DSLRRTCP.

The receiving task makes four types of calls to DSLRRTCP:

- TYPE=START informs DSLNRTCP that the receiving task has started. DSLNRTCP returns the address of an ECB that will be posted when an instruction for the receiving task is sent.
- TYPE=AVAIL informs DSLNRTCP that the receiving task is now ready to receive instructions. The receiving should now wait on the ECB returned by the TYPE=START.
- TYPE=RETRIEVE retrieves the instruction from DSLNRTCP. After the application has processed the instruction, it must use another TYPE=AVAIL for the next instruction.
- TYPE=STOP informs DSLNRTCP that the receiving task wants to stop the remote task communication.

The Instructing Task

The instructing task communicates with DSLNRTCP using DSLNICT (intertask communication interface).

The instructing task makes one type of call to DSLNRTCP:

- TYPE=INSTRUCT informs DSLNRTCP to send an instruction to the receiving task. If the receiving task is not available to receive the instruction, DSLNRTCP rejects the instruction.

DSLISYNP—Synchronization Point Program

The synchronization point program DSLISYNP is defined in DSLNPTT for MERVA ESA running under IMS or CICS. Under IMS, it requests an IMS synchronization point (SYNC) each time a time interval has passed. The time interval is defined in minutes in the entry of DSLNPTT, and it can be modified by stopping DSLISYNP and starting it with a different time interval while MERVA ESA is running.

In case of an IMS restart, the SYNC request of DSLISYNP prevents IMS from scanning too much of the IMS log for the last activity of the BMP region where DSLNUC was active. Under CICS, a CICS syncpoint is executed each time a time interval has passed. When the nucleus runs as a native batch program, DSLISYNP has no effect.

DSLTIMP—Timer Program

The timer program DSLTIMP provides a timer service only for programs defined in DSLNPTT. Because of this restriction, the timer service is not considered in the categories *direct* or *central* services.

DSLTIMP can handle any number of timer requests and transforms them into a single request to CICS or MVS.

DSLTIMP is exclusively called by DSLNUC or DSLNSHEL/DSLNSHEC for:

- Initialization
- Requesting a timer service from:
  - CICS by means of an EXEC CICS POST command
  - MVS by means of an STIMER macro (for MERVA ESA under IMS)
- Posting the programs of DSLNPTT if necessary
- Termination

DSLTIMP is called by the programs of DSLNPTT for:

- Setting a time interval or an expiration time
- Canceling a time interval or an expiration time

DSLCNTP—Message Counter Program

The message counter program DSLCNTP collects and stores in the message counter log information about the number of incoming and outgoing messages processed by MERVA ESA:

- SWIFT messages transferred via SWIFT Link or FMT/ESA
- Telex messages

The message counter log is a VSAM KSDS similar to the MERVA ESA journal data set.

MERVA ESA provides a command and a batch utility that a system administrator can use to produce a report showing the usage of the MERVA ESA system during the last month and the last year. The report lists the numbers of messages sent or received over all of the following:

- An external network link (SWIFT messages transferred via SWIFT Link, and Telex messages)

- The MERVA Link (SWIFT messages transferred via FMT/ESA)
- MERVA-MQI Attachment (SWIFT messages transferred via FMT/ESA)

The program DSLCNTP is defined in DSLNPTT. It uses the MERVA ESA timer service to update the information in the message counter log data set periodically.

Chapter 8. Message Services

This chapter describes the message services available in MERVA ESA for application programs.

The messages described in the following are the messages that you exchange:

- Within your business
- With your business partners
- With the SWIFT network
- With the telex network
- With other networks

The following components are involved with the message services:

- The message fields defined in the MERVA ESA field definition table (FDT)
- The message types defined in the MERVA ESA message type table (MTT)
- The message formats defined in the MERVA ESA message control blocks (MCBs)
- The attributes of external devices defined in the MERVA ESA terminal feature definition table (TFDT)
- The attributes of message processing functions defined in the MERVA ESA function table (FNT)
- The program function keys defined in the MERVA ESA program function key tables (for example, DSLMPF00)
- The TOF services
- The message format services

These components are described in the following.

Field Definition Table

All fields used in messages are defined in the MERVA ESA Field Definition Table (FDT). This table defines the attributes of the fields as they are most often used in the messages that use these fields.

In the definition of the messages, that is, in the message control blocks (see Message Control Blocks) the fields need only be referenced. If a field has slightly different attributes in a particular message, the MCB can override the field attributes.

The DSLLFLD macro is used to define fields, and the DSLLSUBF macro is used to define subfields. The *MERVA for ESA Macro Reference* explains these macros and their parameters. The *MERVA for ESA Customization Guide* shows examples of how to use these macros.

MERVA ESA supplies a sample field definition table with the name DSLFDTT. It contains all field definitions used by MERVA ESA, for all SWIFT messages, for the telex messages, and for MERVA Link.

Message Type Table

All message types used in messages are defined in the MERVA ESA message type table (MTT). This table defines the attributes of the message types, for example, the name of the message control block used for mapping the message type, if nesting with other message types is allowed, and if, for SWIFT message types, authentication is required.

The DSLMTT macro is used to define the message types. The *MERVA for ESA Macro Reference* explains this macro and its parameters. The *MERVA for ESA Customization Guide* shows examples of how to use this macro.

MERVA ESA supplies a sample message type table with the name DSLMTTT. It contains all message types used by MERVA ESA, for all SWIFT messages, for the telex messages, and for MERVA Link.

Message Control Blocks

The layout of each message type used in MERVA ESA is defined in a message control block (MCB). One MCB can describe several message types. For example, the SWIFT common group messages 199, 299, 399, and so on until 999 are defined in one MCB with the name DWSX99.

An MCB describes the formats of the message type for the following devices:

- The sequence of fields for the MERVA ESA internal format
- The formats for display stations
- The formats for hardcopy printers
- The formats for system printers
- The formats for external network lines and applications

There is only one description of the MERVA ESA internal format of a message, but for the other ones there can be several descriptions using different layouts or languages.

The macros DSLLMCB, DSLLDEV, DSLLGRP, DSLLUNIT, DSLLMFLD, DSLLDFLD, DSLLNFLD, DSLLUEND, DSLLGEN, DSLLCOND, and DSLLEXIT are used to define the various aspects of the message formats. The *MERVA for ESA Macro Reference* explains these macros and their parameters. The *MERVA for ESA Customization Guide* shows examples of how to use these macros.

MERVA ESA supplies sample MCBs for all message types used by MERVA ESA, for all SWIFT messages, for the telex messages, for MERVA Link, and for Financial EDIFACT messages.

In addition, MERVA ESA uses and supplies special MCBs for the following purposes:

- Title information on screen and print pages (top frame)
- Information for the bottom of screen and print pages (bottom frame)
- Signon, function selection, and signoff panels for end users
- Message selection panels in message-processing functions using a display station
- List panels with messages or records for selection
- Panels for online maintenance and operator command processing
- Help information

Terminal Feature Definition Table

The terminal feature definition table (TFDT) defines the attributes of the following devices:

- Display stations for MERVA ESA working under IMS
- Hardcopy printers
- System printer page sizes

The DSLTFD macro is used to define the terminal feature definition table. The *MERVA for ESA Macro Reference* explains this macro and its parameters. The *MERVA for ESA Customization Guide* shows examples of how to use this macro.

MERVA ESA supplies a sample table with the name DSLTFDT.

Function Table

The MERVA ESA function table (FNT) defines the attributes of the functions such as message processing functions. The following attributes are considered by the services that process messages (queue attributes of message-processing functions are shown in "Definition of Queues" on page 70):

- Formal checking of field contents
- Address expansion
- Top and bottom frame MCBs
- Display modes (PROMPT, NOPROMPT)
- Message ID to control which parts of SWIFT and telex messages are shown on a screen or printer
- Program function keys used on display stations
- Formats and languages used for the display
- Protection of fields on screens

The DSLFNT macro is used to define the MERVA ESA function table. The *MERVA for ESA Macro Reference* explains this macro and its parameters. The *MERVA for ESA Customization Guide* shows examples of how to use this macro.

MERVA ESA supplies a sample table with the name DSLFNTT.

Program Function Key Tables

Program function key tables are used to assign commands to the program function keys (PF keys) of a display station. PF keys can be defined depending on the selected function. The PF key tables are referenced:

- In the user file record of a user
- In the selected function
- The PF keys referenced by the PFKEYS commands

If no PF key table name is found, the default table DSLMPF00 supplied by MERVA ESA is used.

The DSLMPFK macro is used to define PF key tables. The *MERVA for ESA Macro Reference* explains this macro and its parameters. The *MERVA for ESA Customization Guide* shows examples of how to use this macro.

TOF Services

When messages are processed with MERVA ESA, they are in the MERVA ESA internal message buffer, also called the *tokenized format* (TOF). In the TOF, the fields of a message are stored individually, and they can be accessed directly using the MERVA ESA TOF Supervisor program DSLTOFSV, as shown in Figure 8.

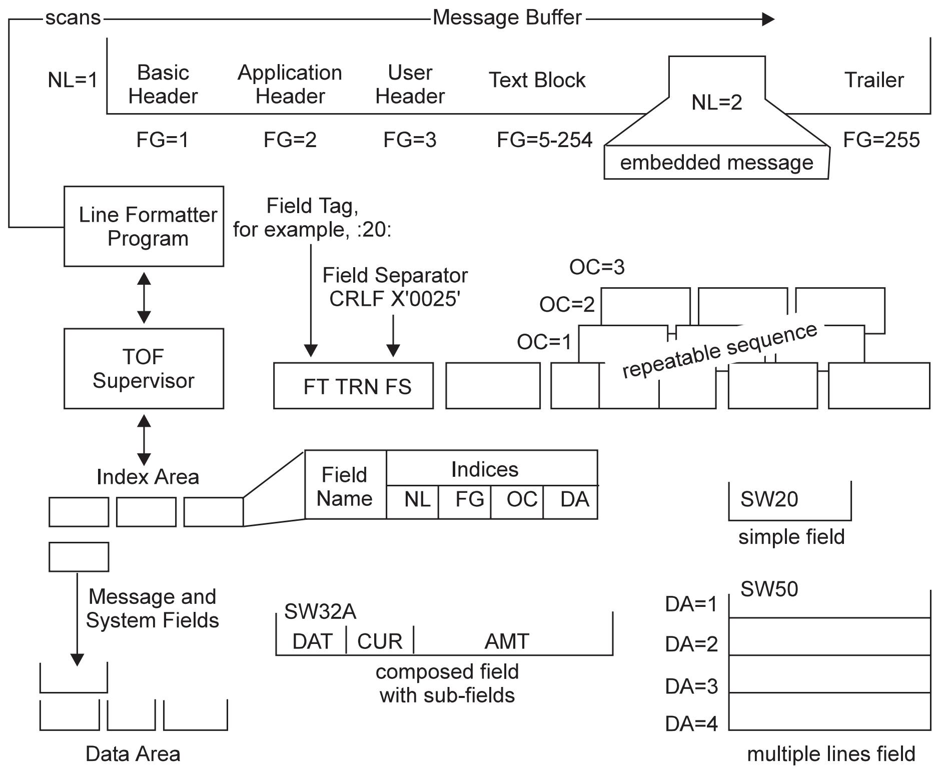


Figure 8. The TOF Supervisor

The TOF Supervisor has the following functions:

- Creating a new TOF
- Initializing fields in the TOF
- Writing data to the TOF
- Adding data areas to the TOF
- Reading data from the TOF
- Deleting data from the TOF
- Accessing fields in the TOF
- Checking fields in the TOF
- Expanding fields in the TOF
- Adding a nesting identifier to the TOF
- Compressing the TOF into a buffer
- Merging the TOF with a TOF from a buffer

The TOF is dynamic, that is, if defined in DSLPRM, the TOF space is increased during processing if necessary. To make use of the TOF dynamic space allocation you must either define the TOF increase value in the DSLPARM, or you set the increase value using a TOF WRITE request.

If the dynamic TOF is enabled, a TOF full condition will result in the allocation of internally requested storage. From now on the TOF is split into an index part (the TOF space supplied by the caller) and a data part (internally requested). These two areas are linked by a pointer in the TOF header of the caller's TOF. The TOF Supervisor will only request more storage if reorganization of the data part does not return enough free space to service the request.

For the dynamic TOF the following new functions are defined:
- Freeing the storage of the TOF data part
- Joining the index part and the data part of the TOF in one buffer

When referencing a field of the TOF in a request to DSLTOFSV, the following qualifiers (called the *field reference*) can be used:
- The name of the field or subfield as defined in the field definition table
- The nesting identifier, for example, zero for internal fields of MERVA ESA, or 1 to *n* for SWIFT message fields depending on the nesting of message types
- The field group if groups are defined in the MCB
- The repeatable sequence index if the field is in a repeatable sequence. This index specifies in which of several repeatable sequences the field is. Several indexes are necessary to address fields in nested repeatable sequences.
- The occurrence index if the field is in a repeatable sequence. This index specifies in which repetition of the sequence the field is.
- The data area index to access one data area of several

In some cases, the field reference can be simplified using request modifiers, for example, with the request modifier VFIRST the TOF is scanned from the very first field until the field name is found.

Request modifiers can also be used to access the option of a field tag.

DSLTOFSV is invoked with the DSLTSV macro directly by the requesting MERVA ESA application program. Refer to the *MERVA for ESA Macro Reference* for details about the DSLTSV macro.

For a detailed description of how to request the TOF services refer to *MERVA for ESA Customization Guide*.

Message Format Services

Product-Sensitive Programming Interface

The MERVA ESA message format service (MFS) performs the following functions:
- Transforms a message from the internal format to the external formats
- Transforms a message from an external format to the internal format
- Performs other message-related services

An overview of the MFS is shown in Figure 9 on page 58.

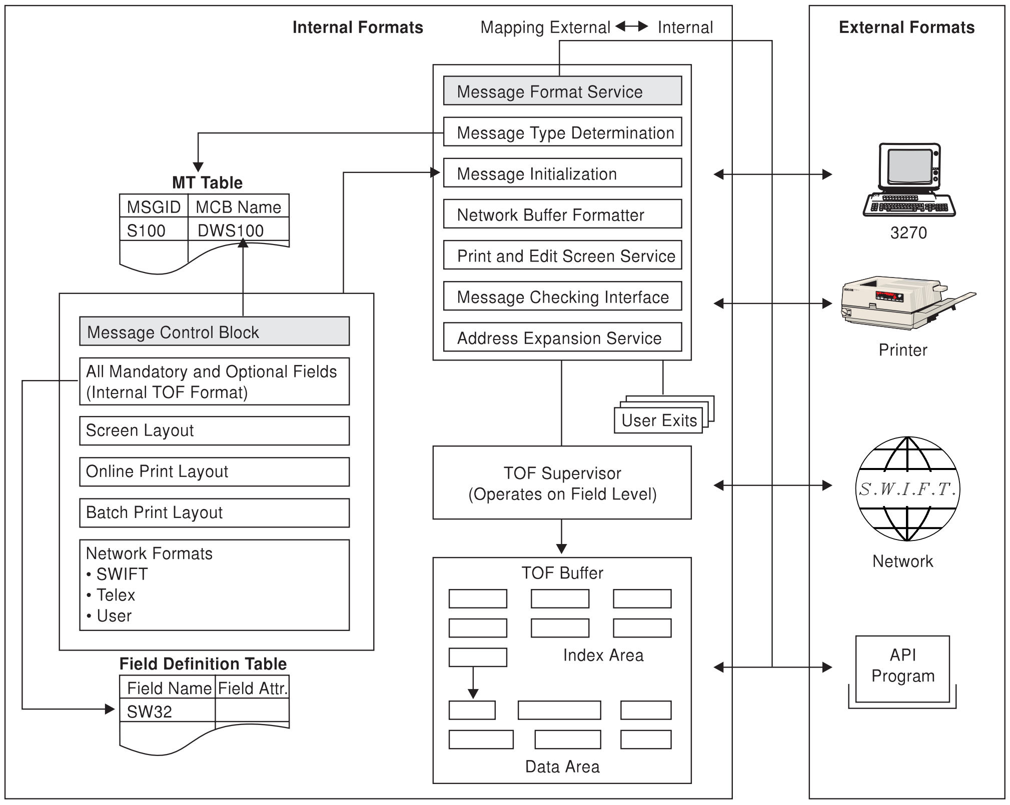


Figure 9. Message Format Service

The MERVA ESA MFS consists of several modules that carry out the message transformation and other services. These are:

DSLMMFS	MFS interface
DSLMPTT	MFS program table for accessing the other modules
DSLMTIN	Message initialization in the TOF
DSLMLFP	Line (network buffer) formatter
DSLMLEF	External line format program
DSLMPxxx	Print and edit (screen) services
DSLMNOP	NOPROMPT service
DSLMCHE	Message checking interface
DSLMXPND	Message expansion interface
MFS exit routines	MFS exit routines for checking, setting defaults, editing, expanding, separating, and general use.

The MFS is invoked directly by the MERVA ESA applications programs using the DSLMFS macro (see the *MERVA for ESA Macro Reference* for details). The DSLMFS macro invokes the interface program DSLMMFS. The execution of the modules is controlled by the program table DSLMPTT. This table contains an entry for each MFS program and the modules used by MFS.

The functions of the modules are described in the following; no distinction is made about which MERVA ESA component really invokes MFS for a given function; when referring to the module or modules invoking MFS, only the terms *caller* or *calling program* are used.

DSLMMFS—MFS Interface

The interface module to all message format services is DSLMMFS.

When a MERVA ESA component requests a service from MFS, it uses the DSLMFS macro with parameters specifying the type of service requested, the addresses of storage areas required for data manipulation, and other related information. The DSLMFS macro prepares the MFS parameter list and passes control to DSLMMFS.

DSLMMFS invokes the appropriate MFS module. When the MFS module is defined as a user exit written in a high-level language, it is invoked via the MERVA ESA exit manager program DSLXMGR. DSLXMGR sets up the required language environment and transfers control to the MFS module. On completion of a function, the MFS module returns control to DSLMMFS, which analyzes and prepares error messages.

The calling program is informed by return and reason codes whether the request was successful.

MFS Initialization

This service establishes the MFS working environment in the MFS permanent storage.

In a conversational IMS MPP, each conversation step must start with an MFS initialization to refresh the addresses of the MFS storage areas used.

Module Control

The requested module is located in the MFS program table DSLMPTT. The table entry indicates if the module is link-edited to DSLMMFS, or if it must be loaded separately.

DSLMMFS keeps track of loaded modules in the MFS load table. This table is either provided by the caller or allocated dynamically by DSLMMFS.

The number of entries in the load table (that is, the number of MCBs and MFS modules held concurrently during a MERVA ESA transaction) is specified by the parameter MCBNUM of the macro DSLPARM in the MERVA ESA customizing parameter module DSLPRM.

Get Device Descriptor: This service locates the description of an external format in an MCB. If the requested layout or language is not found, the first one defined for the requested device is used.

Get Message Type Table Entry: This service locates an entry in the message type table.

Get Program Function Key Table: This service locates a program function key table.

Prepare Error Message: This service evaluates the reason code and possibly generates an error message. The error message is stored in the MFS permanent storage and, depending on the calling option, stacked in the TOF field DSLMSG or written into the TOF field DSLERR.

MFS Termination: This service deletes the MCB and MFS modules named in the MFS load table and releases the MFS load table if it was allocated by DSLMMFS.

DSLMPTT—MFS Program Table

All modules used by DSLMMFS are defined in the MFS program table (DSLMPTT). These are:
- Message format service programs
- MFS exits (including checking, default setting, editing, separation, expansion, and user exits)
- Message control blocks (MCBs)
- Program function key tables (PF key tables)

The programs can be defined for link-editing to DSLMMFS, or for loading. If an MFS program or user exit is not found in DSLMPTT, it cannot be used. If an MCB or PF key table is not found in DSLMPTT, an attempt to load it is done in any case.

A program can be defined as written in one of the following programming languages:
- Assembler
- COBOL
- C/370
- PL/I

The program is called in the requested language environment. The DSLMPT macro is used to define the MFS program table. The *MERVA for ESA Macro Reference* explains this macro and its parameters. The *MERVA for ESA Customization Guide* shows examples of how to use this macro.

MERVA ESA supplies a sample MFS program table DSLMPTT. It contains all entries used by MERVA ESA, SWIFT Link, Telex Link, and MERVA Link.

DSLMTIN—Message Initialization and Formatting

DSLMTIN has three functions:
- Initializing a message in the TOF (INIT)
- Transforming a message from the TOF to the queue format (PUT)
- Transforming a message from the queue format to the TOF (GET)

Message Initialization in the TOF

This service initializes the TOF for a new message or message part. Each part is controlled by an exit field, which contains the message identification of the part used for mapping of the message. The internal format described in the MCB is used for the initialization of the field descriptors in the TOF.

The TOF can be initialized only for one message type at a time, that is, if a message contains nested messages or different exit fields, DSLMTIN must be called separately for each message part.

When a new message is initialized (first call of INIT), the old message is deleted from the TOF. Optionally the permanent fields on TOF nesting identifier 0 can also be deleted. The system field NLEXIT is written to the TOF to contain the name of the exit field that is written to the TOF during message initialization.

The message ID provided by the caller is checked for a correct network identification and message type, and written to the exit field whose name is defined in the MTT-entry. If message initialization is called with OPT=CONT, the old message in the TOF is kept, and a new message part can be added to the existing one (for example, using a separate exit field).

If a field descriptor contains the number of a default setting routine, the TOF supervisor calls this routine to set the defaults. For mandatory fields that have an option list defined and for which only one option is allowed, this option is written into the TOF, and therefore need not be entered manually.

On completion of DSLMTIN, the TOF is ready to accept data for this message type. The MFS user exit DSLMU001 is called to allow additional setting of defaults or special startup processing. For more information on user exits refer to the *MERVA for ESA Customization Guide*.

From TOF to Queue Format: This service transforms a message from the TOF into the queue format, excluding any QUEUE=NO fields. The TOF supervisor function TYPE=COMPRESS is used.

The queue format can be given to the MERVA ESA queue management service for storing in the MERVA ESA queues. The message in the TOF remains unchanged.

This service can process large messages (> 32KB) in queue format. If a message in queue format is too large to fit into the provided buffer and the dynamic buffer option is specified, the service allocates a larger buffer on behalf of the caller.

From Queue Format to TOF: This service transforms a message from the MERVA ESA queue format to the TOF. The TOF supervisor function TYPE=MERGE is used.

On completion, all QUEUE=YES fields of the message are back in the TOF in exactly the format they had before they were transformed into the queue format. Permanent fields on TOF nesting identifier 0 that were in the TOF before the merge remain unchanged.

This service can process large messages (> 32KB) in queue format.

DSLMLFP—Line Formatter

The functions of DSLMLFP are:

- Transforming a message from the line buffer to the TOF (GET)
- Transforming a message from the TOF to the line buffer (PUT)

DSLMLFP uses the formats for external network lines in the MCB of the message being processed.

This service can process large messages (> 32KB) in an external network format.

DSLMLFP also calls the MFS user exit 54 (DSLMU054) to determine the message type (refer to the *MERVA for ESA Customization Guide* for more details about this user exit). The network identifier provided by the calling program and the message type are used to find the message type table entry. Then the appropriate MCB is loaded and the TOF is initialized. The line format supplied by the caller is used to find the format for the external network in the MCB.

For each nested message this process is repeated until all data from the buffer is moved into the TOF.

If a message type cannot be determined, DSLMLFP assumes the message identification 0DSL. This message identification contains only one field named DSLLFBUF to which the complete line buffer, or whatever is left, is written.

When there is remaining data in the buffer that cannot be mapped into the TOF because the MCB end is reached, this data is written into the TOF as field DSLLFBUF.

A syntactically incorrect message (that is, a message that contains an erroneous field, or where, for example, a mandatory field is missing) is accepted. The errors in the message are detected during later processing.

Line buffer to TOF (GET)

The input data is transformed from the external line format into the TOF. The data in the line buffer is separated into fields according to the definitions in the MCB, and stored in the TOF. The fields in the buffer are recognized by their field tags. Tags can also be connected to exit points in MCBs; these exit points define a block structure for a message.

The line formatter program scans the buffer contents and tries to find a matching tag in the MCB definition. If a tag found in the buffer is not found in the MCB, the data up to the next tag in the buffer is added to the previous field as additional data areas.

If a LENGTH specification is given in the DSLLNFLD statement, the data is taken in the required length from the buffer and written into the TOF. In this case, the separator is optional. For all other fields a separator is necessary.

For DSLLUNIT statements, SEQTYPE=VAR can be defined to show that the tag matching process should always start at the beginning of the unit instead of continuing at the current position. Thus it is possible to have tag fields occur in any sequence in the buffer when the tags are all different and the fields occur within the same unit.

If data found in the buffer cannot be assigned to a TOF field, this data is discarded. This may happen when tag-only fields are specified in the DSLLDEV TYPE=NET section; tag-only fields have no TOF field name specification. Such events can be traced for the system programmer’s information.

Update Function: The line formatting program can be called with OPT=CONT to add input data to an existing TOF (for example, defined in a separate MCB and for a different exit field).

TOF to Line Buffer (PUT): The function PUT is used to transform a message from the TOF into an external network format according to a message identification and a line format identification.

If a message in an external network format is too large to fit into the provided buffer and the dynamic buffer option is specified, the service allocates a larger buffer on behalf of the caller.

The line formatting program DSLMLFP requests a load of the MCB that is either defined by the calling program in the message identification of the MSGID parameter or contained in the exit field of the TOF. The field name of the exit field is contained in the NLEXIT field of the TOF. The default field name used for all internal MERVA ESA and SWIFT messages is DSLEXIT. Under control of the line format as provided by the calling program and defined in the appropriate DSLLDEV TYPE=NET section of the MCB, the data contained in the various TOF fields is read and transferred to the line buffer. The mapping is done by processing a TYPE=NET device description sequentially. Conditions defined within the MCB are evaluated according to the TOF contents. DSLMLFP adds control characters as defined in the MCB. An item to be mapped can be a tag, a separator, or a data component.

The line formatter program is able to map whole fields, subfields, or single data areas. The optional tag is written in front of the first data component of a field. The tag may contain an option; when the option cannot be read from the TOF, the option space in the tag is filled with hyphen characters (X'60'). A separator is written after each data area of a field. For mapping of fixed length fields, the separator is optional. Both types, field and data area mapping, can be mixed in the same MCB.

ALWAYS for tags and separators indicates that tags or separators must be written into the buffer, even when the field is empty.

If a LENGTH specification is given in the DSLLNFLD statement the data component is written in the required length, even when the data component is empty or shorter than required. Padding with blanks is performed.

For DSLLEXIT statements, tags and separators are mapped only when data has been mapped by the embedded MCB.

When an edit routine is specified for a field in the format for the external network, this routine is called to edit the data. This edit routine is called after all data areas of the field have been moved to the buffer.

The message does not contain any frame control characters required for its transmission over a network. These characters are added by the appropriate network link component.

DSLMLEF—External Line Format Program

The functions of DSLMLEF are:

- Transforming a message in the TOF from external line buffer to the tokenized format
- Transforming a message in the TOF that is in tokenized format to the external line format

These operations are executed in the TOF; a line buffer is not mandatory. If a line buffer is specified as parameter, it is used for the intermediate result and contains the message in external line format.

DSLMPxxx—Print and Edit Services

The functions of the MFS print and edit services are:

- Transforming a message from the TOF to the external formats required for a screen terminal, hardcopy printer, or system printer (PUT)
- Transforming a message from the format of a screen terminal to the TOF (GET)

For this mapping process, an intermediate format is used. This format is called *Logical Data Stream* (LDS). The logical data stream is then transformed into a physical data stream that can be understood by the presentation devices supported by MERVA ESA. The device types supported by MERVA ESA are:

- 3270 screen devices
- 3270 compatible terminal printers
- SCS printers
- System printers

Processing Modes

Messages can be displayed in two processing modes:

PROMPT	The messages are formatted and mapped according to the external formats defined for the devices in the MCBs. The positions and the attributes of the fields and their descriptive labels are defined in the MCBs.
NOPROMPT	The messages are formatted using the format descriptions for external networks in the MCBs. The fields are displayed or printed with their field tags, but without the separators. The program DSLMNOP is called for preparing the display lines.

The compression specification defined for the function with the PRFORM parameter determines the processing mode for the printers.

Page Structure: The messages are structured in logical pages that are processed as units for display or printing. Each page has three areas:

- The top frame displayed at the top of the page
- The message area displayed after the top frame
- The bottom frame displayed after the message area at the bottom of the page

The top and bottom frames are defined by the frame MCBs (FRAME parameter of the DSLFNT macro described in the *MERVA for ESA Macro Reference*). Either the top or the bottom frame must be available for a screen device to hold the command line (field DSLCMDL) and the error message line (field DSLERR). The MFS user exit DSLMU003 is called to put user data and information into the TOF for display in the top and bottom frame (refer to the *MERVA for ESA Customization Guide* for more details about this user exit).

The message area is mapped according to the number of lines that remain on the physical page between the top and bottom frames.

Editing: When mapping the data for the device, field data can be converted from the TOF format to presentation format by editing exit programs that are specified in the field definition table and the MCBs.

Output to 3270 Devices: The 3270 device status is determined and the model-dependent size and features (color display and extended highlighting) are retrieved from the terminal user control block (TUCB) provided by the calling program. The calling program gets this information from the CICS terminal definitions or MERVA ESA terminal feature definition table DSLTFDT.

The mapping operation produces a 3270 data stream or an SCS data stream in the I/O buffer that can be sent to the device, using an EXEC CICS send command in CICS or an insert (ISRT) request in IMS.

Output to System Printers: For system printers, one print line is produced with one call to the MFS. An ASA print control character is generated at the first position in the buffer. A reason code shows whether more lines are available (reenter necessary), or if the message is complete.

Input from Display Stations: Data received from a screen is moved into an I/O buffer by an EXEC CICS receive command in CICS or get unique (GU) and get next (GN) requests in IMS. The data in the I/O buffer is mapped into the LDS and from there into the TOF. De-editing of data is performed if required. The keyboard function is evaluated according to the received attention ID. Program function keys, PA keys, and ENTER are defined in the program function key table. When a command is entered, it is put into the TOF field DSLCMDL. The MFS screen command interpreter must be called for the execution of screen commands.

DSLMNOP—NOPROMPT Mapping

In NOPROMPT display mode, the MFS NOPROMPT mapping program DSLMNOP is called by the MFS print and edit services when preparing the logical data stream (LDS) for display, and when processing the input from a display station. DSLMNOP calls the MFS line formatter DSLMLFP to map a TOF into a line buffer or vice versa. The line format 'X' or 'Y' is used for this mapping process. The separator carriage return line feed (CRLF) is used to split a message into several lines for display on a display station.

The line buffer is used to process the data for the logical data stream (LDS).

DSLMCHE—MFS Checking Interface

This MFS service is used as an interface for checking messages and message fields. The field characteristics and field checking modules are specified in the field definition table and the MCBs. The message checking modules are specified in the message type table. The checking of message and field contents is always carried out on the data as it is contained in the TOF.

Field checking is carried out for each screen page and for the complete message in NOPROMPT mode. Message checking is carried out by DSLMCHE when a message is completely mapped to the TOF.

Checking consists of:

- Calling MFS field checking programs (DSLMC*nnn*) to check the field contents according to special, message-dependent criteria. *nnn* is specified by the CHECK=*nnn* parameter of the field definition table or an MCB
- Checking the number of occurrences of a repeatable sequence of fields against what is defined in the REPSEQ parameter
- Calling a message checking exit to check the interdependencies between fields, permitted message types, and message nesting as specified in the message type table
- Calling the MFS user exit DSLMU009 to carry out final checks on the message according to user specifications (refer to the *MERVA for ESA Customization Guide*

for more detail about this user exit). The result of the previous checking operations may be changed by this routine.

DSLMXPND—MFS Expansion Interface

This MFS service is used as an interface for the expansion of message fields with the expansion exit programs provided by MERVA ESA. During online message processing at a display station, the expansion program is called for each screen cycle. In the transaction for message checking and expansion DSLCXT, the expansion program is called once for the whole message.

The MERVA ESA expansion facility allows field-specific exit programs to add data to message fields.

DSLMXPND calls the expansion exits depending on:

- The field specification in the MERVA ESA field definition table DSLFDTT (see the EXPAND parameter of the DSLLFLD macro)
- The function definition in the MERVA ESA function table (see the EXPAND and EXPNAM parameters of the DSLFNT macro).

SWIFT Link uses the expansion function and provides expansion exits for the following:

- Expanding SWIFT addresses (BICs) into correspondents' names for all SWIFT address fields in financial transactions
- Expanding user-provided nicknames for financial institutions into SWIFT addresses and into correspondents' names for all SWIFT address fields in financial transactions
- Initializing embedded (nested) message types within SWIFT message types x92, x95, and x96
- Filling in the option area for a field when only one option character is allowed for this field

Telex Link uses expansion for the sender's and receiver's addresses when preparing telex messages.

MFS Exits

MFS provides exits to exit routines, which are routines called by the MFS that can do additional processing. MERVA ESA provides sample exit routines, but you can also write your own, provided you follow the coding rules for such routines. The sample routines and the coding rules are described in the *MERVA for ESA Customization Guide*. MFS exit routines can be written in the following programming languages:

- Assembler
- COBOL
- C/370
- PL/I

Exit routines can be of the following types:

Checking
Checking exit routines validate the contents of specific message fields stored in the TOF. The definition of a field in the field definition table or in an MCB contains a number from 1 to 32767, which is used to find the checking module in DSLMPTT.

MERVA ESA provides standard checking modules numbered from 901 to 913 for character set and date-and-time format checking. SWIFT Link provides checking modules for all SWIFT fields. Telex Link provides checking modules for the telex fields.

Give your checking exit routines names of the form DSLMC*xxx*, where *xxx* is a string of any 3 characters except the numbers 900 to 999 (these are reserved for use by MERVA ESA).

Default setting

Default setting exit routines are called to set default data into specific message fields:

- During the initialization of a field
- When the field is read and initialized in the TOF, but contains no data
- When the field is written and is not yet in the TOF, it is initialized implicitly and default setting called therefore

The definition of a field in the field definition table or in an MCB contains a number from 1 to 32767, which is used to find the default setting module in DSLMPTT.

MERVA ESA provides default setting modules numbered from 904 to 910 to set defaults for date and time into TOF fields. SWIFT Link provides default setting modules for the SWIFT message headers and some of the field options. Telex Link provides default setting modules for the telex header fields.

Give your default-setting exit routines names of the form DSLMD*xxx*, where *xxx* is a string of any 3 characters except the numbers 900 to 999 (these are reserved for use by MERVA ESA).

Editing

Editing exit routines insert special characters into the data of a message field when the field is read from the TOF to be displayed, or remove special characters from data when read in from the screen to be stored in the TOF. The definition of a field in the field definition table or in an MCB contains a number from 1 to 32767, which is used to find the editing module in DSLMPTT.

MERVA ESA provides:

- Two amount editing modules (one for European format, one for American format)
- Modules to change the display attributes for a field on a screen device
- A module to replace leading zeros by blanks

Give your editing exit routines names of the form DSLME*xxx*, where *xxx* is a string of any 3 characters except the numbers 900 to 999 (these are reserved for use by MERVA ESA).

Expansion

Expansion exit routines change the contents of TOF fields or carry out actions based on the contents of TOF fields. The definition of a field in the field definition table or in an MCB contains a number from 1 to 32767, which is used to find the expansion module in DSLMPTT.

SWIFT Link and Telex Link provide expansion modules that use expansion as described in “DSLMXPND—MFS Expansion Interface” on page 66.

Give your expansion exit routines names of the form DSLMX*xxx*, where *xxx* is a string of any 3 characters except the numbers 900 to 999 (these are reserved for use by MERVA ESA).

Separation
Separation exit routines separate the data in message fields into subfields when a subfield is accessed, or write a subfield into the data of a field. The definition of a field in the field definition table or in an MCB contains a number from 1 to 32767, which is used to find the separation module in DSLMPTT.

MERVA ESA provides separation modules to separate fields with a fixed structure according to the specification in the field definition table, and to access system fields. SWIFT Link provides the separation modules for all subfields defined for SWIFT fields.

Give your separation exit routines names of the form DSLMS*xxx*, where *xxx* is a string of any 3 characters except the numbers 900 to 999 (these are reserved for use by MERVA ESA).

Other The input for exit routines that do other things consists of the TOF and the terminal user control block (TUCB). A user exit communication field is also provided by MFS. The TOF contains all message-dependent information, which can be accessed or changed by any of the MFS user exits. The TUCB contains information about the status of the message in process, the message-processing function, and general user information. All this information can be used, but only some of the information in the TUCB can be changed by exit routines.

Give your exits for purposes other than the ones listed above names of the form DSLMU*xxx*, where *xxx* is a string of any 3 characters except the numbers 001 to 999 (these are reserved for use by MERVA ESA).

End of Product-Sensitive Programming Interface

Chapter 9. Queue Services (VSAM)

This chapter describes the message queue services available in MERVA ESA for application programs using a VSAM queue data set.

The MERVA ESA queue services are concerned with storing and retrieving messages in MERVA ESA queues. The following components are involved with the queue services:

- The queues defined in the MERVA ESA function table (FNT)
- The queue data set (QDS)
- The queue management program DSLQMGT
- The queue management I/O program DSLQMGIO or DSLQMCNV
- The large message cluster (LMC)
- The large message service program DSLQLRG
- The routing of messages between queues
- The queue data-set utility DSLQDSUT
- The large message cluster maintenance utility DSLQMNT

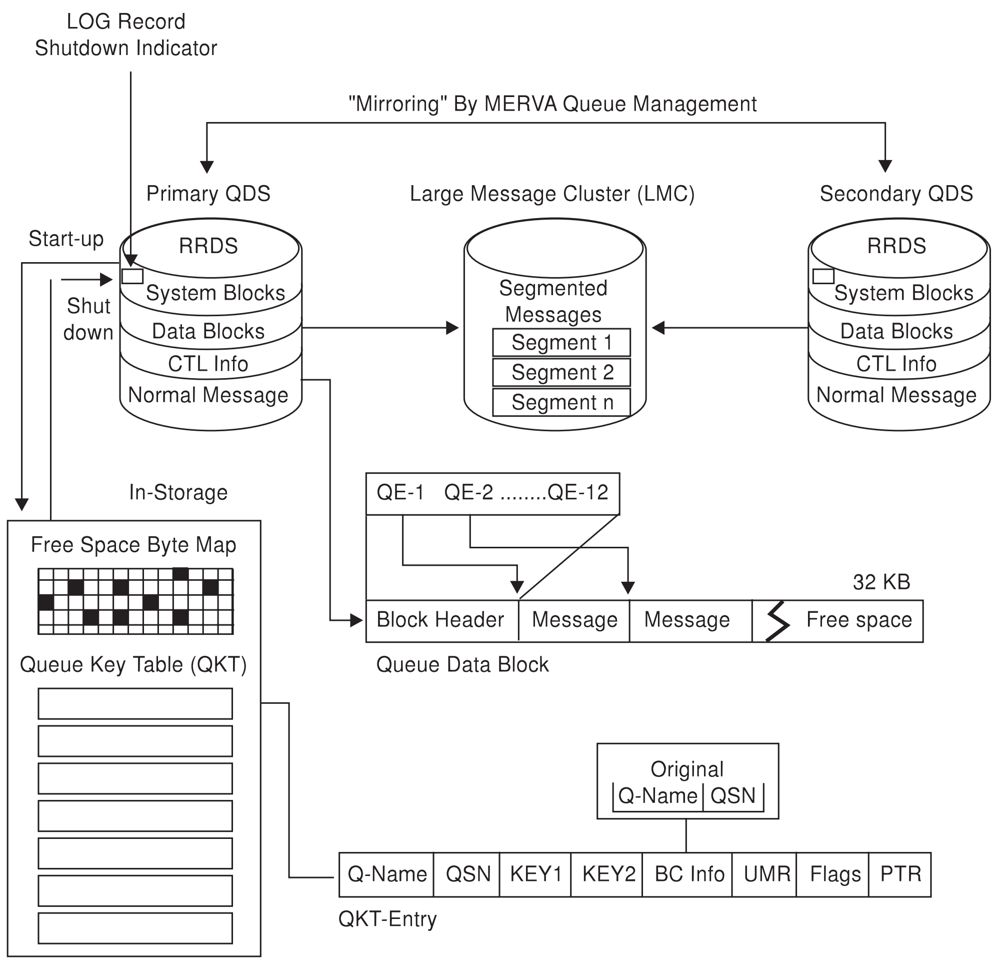


Figure 10. The Queue Services (VSAM)

DSLQDSUT is described in "Queue Data Set Utility" on page 177. DSLQMNT is described in "Large Message Cluster (LMC) Maintenance Utility" on page 179. The other components are described in the following.

Definition of Queues

The MERVA ESA queues are defined in the MERVA ESA function table by means of the DSLFNT macro. The parameter QUEUE=YES defines an input queue for a message-processing function (see the *MERVA for ESA Macro Reference* for details). Not all functions need a queue. Functions that do not need a queue are, for example, the operator command function CMD or the user file maintenance functions USRn.

Even some message-processing functions need not have a queue, for example, the functions that create new messages (data entry functions).

MERVA ESA queues can have the following attributes:

- Use of one or two keys for direct message retrieval. The keys can be provided by queue management from the contents of the message, or can be supplied by the program that stores the message in the queue.

- Routing criteria. There can be a fixed definition for the next queue, or the name of a routing table can be specified to make the next queue dependent on the contents of the message.
- The length of the stored messages (see "Definition of Small and Large Messages" on page 75).
- A transaction can be started when a message is written to the queue, and a terminal can be specified for the transaction (for example, for hardcopy printing).
- Dummy queues. These can be subject to any queue management operation. For retrieve requests, these queues are always considered to be empty. For put requests, messages are never stored, but the additional functions such as starting a transaction are performed.

Queue Data Set (QDS)

All messages stored in the MERVA ESA queues are stored in the MERVA ESA queue data set (QDS). The QDS is a VSAM RRDS (relative record data set) with a fixed record length of 32760 bytes. You define the number of QDS blocks in the VSAM cluster, and you use the queue data-set utility DSLQDSUT to format the queue data set before you use it for the first time (see "Queue Data Set Utility" on page 177).

The following information is contained in the QDS:

- One log record. This is the first block, and it contains status information, for example the date and time of the last usage, and a large message counter.
- Two byte map records. They contain information on how the data blocks are used.
- The key table records. They start with the fourth block, and their number is determined by DSLQDSUT (see "Queue Data Set Utility" on page 177 for details). They contain the index information, that is, the *queue key table* entries, for the messages stored in the data records.
- The data records. They start after the key table records until the end of the VSAM cluster and are used by DSLQMGT to store messages. A maximum of 65512 data records can be supported. The messages in the data records are called *queue elements*. The queue elements have variable lengths in a data record. Each queue element can be associated with up to 12 queues; each contains information indicating to which queues it belongs, and which keys are used in these queues.

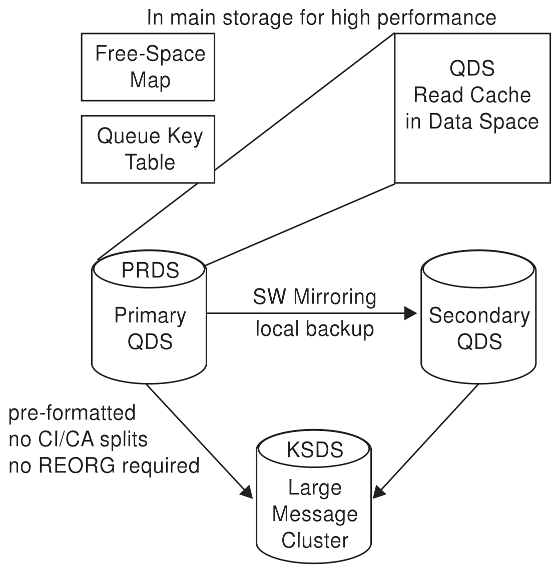


Figure 11. The Queue Data Set

The QDS is allocated to the partition or region where MERVA ESA is active, that is, where the MERVA ESA nucleus has been started. Only one MERVA ESA system can use a QDS at a time.

For safety reasons, a second QDS can be used to keep an exact copy if there are hardware errors or other errors. You can specify a second QDS in the MERVA ESA customizing parameters DSLPRM (see the *MERVA for ESA Macro Reference*). When a duplicate QDS is used, all read operations are performed with the first QDS, and all write operations are performed in both queue data sets.

When two queue data sets are used, and a system breakdown or an I/O error occurs, the status of the two data sets can differ. In this case, the *best* QDS must be duplicated before starting MERVA ESA again. DSLQMGT informs the MERVA ESA operator which QDS should be used.

You can also create a duplicate of the QDS using the user exit DSLQPUT. This exit is called whenever a QDS block is written to the queue data set.

You can use DSLQDSUT to increase the size of the QDS. The QDS need not be empty.

Queue Management Program DSLQMGT

Product-Sensitive Programming Interface

The MERVA ESA queue management program DSLQMGT is controlled by DSLNUC. DSLQMGT performs the following tasks:

- Initialization
- Servicing message queue requests
- Termination
- Processing the unique message reference
- Tracing queue operations
- Calling DSLQMGIO or DSLQMCNV for the VSAM access to the QDS

- Calling DSLQLRG for the data of large messages
- Calling user exits

Initialization of DSLQMGT

During the startup of MERVA ESA, DSLNUC requests the queue management INIT function. The initialization is carried out depending on the status of the QDS:

- If two queue data sets are used, check whether they are identical. If they are, the initialization continues. If they are not, error messages show what is different and what should be done before trying to initialize DSLQMGT again.
- If the QDS log record contains a large message counter > 0, large message support must be specified in DSLPRM, otherwise initialization stops.
- If large message support is specified in DSLPRM, check the integrity of QDS and LMC. If QDS and LMC are inconsistent, initialization stops.
- Get main storage for the processing of DSLQMGT.
- If large message support is specified in DSLPRM, call the program DSLQLRG for initialization.
- Start normally if the previous termination of DSLQMGT was successful or if the QDS was formatted with the queue data-set utility DSLQDSUT.
- Perform a restart if the QDS shows that the previous work was not correctly terminated, or if the QDS was processed by the DSLQDSUT MODIFY function.
- Call the MERVA ESA routing scanner DSLRTNSC for initialization.

Normal Start of DSLQMGT

During a normal start, DSLQMGT gets the following information from the queue data set for saving in main storage:

- The byte map from the byte map records (second and third record). The byte map informs DSLQMGT of how much space is used in each data block by the queue elements stored there, and how much space is free for storing more queue elements.
- The queue key tables from the key table blocks (from the fourth record up to as many as DSLQDSUT has allocated during the formatting). The queue key tables inform DSLQMGT about the queue elements contained in the data blocks:
  - The queue to which a queue element belongs
  - The sequence of the queue elements in a queue indicated by the *queue sequence number* (QSN)
  - The keys of the messages

After formatting, the queue key tables and the byte map show an empty QDS.

Restart of DSLQMGT

In a restart, DSLQMGT cannot use the information contained in the log record, the byte map records, and the key table records of the QDS for processing. Instead, DSLQMGT reads all data records to find out:

- How much space is used in them (for the byte map)
- To which queues the queue elements belong
- What their sequence in the queues is, and which keys they have (for the queue key tables)

As the restart may take some time depending on the size of the QDS, the following list gives information about how to avoid restarts if possible.

DSLQMGT performs a restart after:

- A system breakdown, when DSLQMGT could not perform its termination, for example, when terminating CICS without terminating MERVA ESA
- Having used the MODIFY function of DSLQDSUT (the restart after MODIFY can be shorter because DSLQDSUT indicates to DSLQMGT which data blocks contain queue elements)
- Having found, during initialization, a queue element that belongs to a queue that was not found in the MERVA ESA function table after this table has been changed (to avoid the restart, run the MODIFY function of DSLQDSUT with the EXCLUDE FNT control statement)
- Having found, during termination, that the key table records cannot hold the queue key tables, because the number of queue elements in the MERVA ESA customization parameters DSLPRM or the key specifications in the MERVA ESA function table have been changed (to avoid always having a restart, use the MODIFY function of DSLQDSUT; this causes only one more restart)

After an abnormal end of MERVA ESA, DSLQMGT may not perform its termination when an operation with the QDS is incomplete.

During the restart, DSLQMGT may find out that it was interrupted in the previous run between a store operation of a queue element and the *automatic delete* of the original queue element in another queue. Automatic delete is requested from DSLQMGT in a store operation by specifying the queue name and the queue sequence number of the original queue element. This information is saved in the data records of the QDS together with the queue element. Using this information, DSLQMGT can complete the interrupted operation, thus avoiding that an abnormal end duplicates a message in the QDS.

All MERVA ESA applications use automatic delete where possible. There can be only one queue element in the QDS that requires deletion in a restart of DSLQMGT if your MERVA ESA applications also use automatic delete whenever possible. Not using automatic delete can create duplicate queue elements that cannot be recognized by DSLQMGT in a restart.

Termination of DSLQMGT

During the termination of MERVA ESA, DSLNUC requests the queue management TERM function. The following steps are performed:

- Save the byte map in the byte map records and save the queue key tables in the key table records of the QDS. This is not done if:
  - DSLQMGT initialization has found messages that belong to function queues that have been removed from the MERVA ESA function table
  - The termination of DSLQMGT is entered after an abnormal end and an operation with the QDS is incomplete
- Write the log record to the QDS to indicate normal end of work.

  This is not done if there is a reason to not save the queue key tables in the key table records, or if the queue key tables do not fit into the key table records. Instead, a restart is performed during the next initialization of DSLQMGT (refer to “Restart of DSLQMGT” on page 73 for details).
- Free all main storage obtained during initialization.
- If large message support is specified in DSLPRM, call the program DSLQLRG for termination.
- Close the QDS.
- Call the MERVA ESA routing scanner DSLRTNSC for termination.

Servicing Message Queue Requests

DSLQMGT is invoked when a MERVA ESA application program requests to:

- Store a message in a queue
- Retrieve a message from a queue
- Delete a message in a queue
- Get status information

The processing functions of DSLQMGT are invoked by the programs:

- Link-edited to DSLNUC directly with the DSLQMG macro
- Not link-edited to DSLNUC via the MERVA ESA intertask communication. The task server DSLNTS invokes DSLQMGT as a central service, and the DSLQMG macro is used by the requestor to prepare the queue parameter list for the request

The DSLQMG macro is described in the *MERVA for ESA Macro Reference*.

Definition of Small and Large Messages

A distinction is made between normal, small, and large messages:

- **Normal** messages fit into one QDS block and are handled completely by DSLQMGT.
- **Small** messages have a size limitation defined with the STORE=(SMALL,nnnnn) parameter of the DSLFNT macro for this target queue. If the message is longer than nnnnn, it cannot be stored in the target queue. Small messages are handled completely by DSLQMGT.
- **Large** messages are supported in MERVA ESA for MVS and VSE/ESA Version 1.3 or later only. They are not supported under earlier versions of VSE.

  To support large messages, the LRGMSG=(YES,nnnnn) parameter of the DSLPARM macro must be specified in DSLPRM. If a message is longer than nnnnn, it is considered to be a large message. For large messages, DSLQMGT only stores status information in the QDS, the message data is given to DSLQLRG for storage in the large message cluster (LMC).

  The size nnnnn can be defined individually for each queue with the STORE=(LARGE,nnnnn) parameter of the DSLFNT macro.

  Large messages cannot be stored in queues for which the STORE=(SMALL,nnnnn) parameter of the DSLFNT macro is used.

The DSLFNT and DSLPARM macros are described in the *MERVA for ESA Macro Reference*.

Storing Messages in Queues

Messages can be stored in MERVA ESA queues using one of the following request types:

- Single PUT. The calling program requests to put the message into one specific queue. The calling program supplies the message, the queue name, and the information for automatic delete if this message was retrieved from a queue and is to be deleted there.
- Single PUT with RESTORE modifier. The calling program requests to restore the message into its former queue with its former QSN. The calling program supplies the message, the queue name, and the QSN.

- Multiple PUT. The calling program requests to put the message into up to three specific queues. The calling program supplies the message, 1 to 3 queue names, and the information for automatic delete if this message was retrieved from a queue and is to be deleted there.
- ROUTE only. The calling program requests to invoke the routing scanner DSLRTNSC to find out in which queues the message would be stored. Routing to up to 12 queues is possible. The calling program supplies the message and either the name of the processing function or the address of a routing table that is to be used for the routing decision.
- ROUTE with multiple PUT. The calling program requests to invoke the routing scanner DSLRTNSC to find out in which queues to store the message, and to store the message in these queues. Routing and storing in up to 12 queues is possible. The calling program supplies the message, the name of the processing function that is to be used for the routing decision, and the information for automatic delete if this message was retrieved from a queue and is to be deleted there.
- REPLACE. The calling program has retrieved a message from a queue and requests to replace the message there. The message keeps its position in the sequence of the messages in this queue. Internally, REPLACE is processed like a single PUT with automatic delete. The calling program supplies the message (the message may be longer or shorter than before), the queue name, and information about its position in the queue (indicated by the queue sequence number).

  The *in-service* indicator remains unchanged (see "Retrieving a Message from a Queue" on page 78).

Except for REPLACE, the messages are stored at the end of the relevant queues.

When storing a message in one or more queues, the following steps are performed:

- Invoking the routing scanner DSLRTNSC if routing is requested.

  Programs link-edited to DSLNUC can supply a function name or the address of a routing table for the routing decision.

  Programs not link-edited to DSLNUC can only supply a function name for the routing decision.

  In the specified function, either a NEXT function or a routing table is available.

  When a routing table is used, the contents of the message can be used for the determination of the target queues.

  The target queue names are made available to the requesting program in the queue parameter list. If routing to more than 3 queues takes place, the queue parameter list extension specified with the EXT=YES parameter of the DSLQMG macro is needed to return all target queue names.
- Finding a data block where the message fits.

  The length of a queue element is the length of the message data and the length of the status information. For large messages, only status information with an extension is stored in the QDS. The message data is given to DSLQLRG for storage.

  Queue elements that exceed the maximum capacity of a data block cannot be stored. DSLQMGT uses the byte map to find a block where the queue element fits. If no block is found, the queue data set is full and the queue element cannot be stored. It may happen that the remainders of free storage in all data blocks cannot hold a long queue element, but a shorter one may still fit.

If DSLQLRG cannot store a large message because the large message cluster (LMC) is full, the queue element is also not stored.

- Getting the data block and creating the new queue element.

  If a data record is found with sufficient free space, it is read from the queue data set. The status information and, for normal and small messages, the message data are moved to the data block.

- Determining the keys that are used in the 1 to 12 queues.

  Each target queue can be defined with one or two keys for direct retrieval of messages. Keys have a maximum length of 24 characters. Only as many characters are used during storage and retrieval as are defined for the queues. Shorter keys must be padded with blanks or binary zeros. If blanks are used, they are not considered during storage and retrieval.

  There are four ways to determine the keys of a message. No matter how the key or keys are supplied, no check is made for the keys being unique in the same queue.

  The keys can be supplied as follows:

  1. DSLQMGT retrieves the keys from the message. The message must be in the MERVA ESA queue format (this is a compressed form of the MERVA ESA internal message buffer TOF), and the TOF field names must be defined in the *KEY1* and *KEY2* parameters of the appropriate function-table entry. The KEY=(0,0) parameter must be specified in the DSLQMG macro, or the two key fields in the calling parameter list (queue parameter list, QPL) must be cleared to binary zeros before the DSLQMG or DSLNIC macro.
  2. Use the KEY parameter in the DSLQMG macro: *KEY=(KEY1,KEY2)*. KEY1 and KEY2 must be fields of 24 bytes each. If the key is shorter, it must be padded with binary zeros or blanks. DSLQMGT takes from each key only the length specified in the function-table entry for each queue.
  3. Use the DSLQMG macro without the KEY parameter. The KEY parameter can be omitted if values for KEY1 and KEY2 are stored in the fields *QPLKEY1* and *QPLKEY2* of the QPL.
  4. Use the user exit DSLQKEY to supply keys independent of the methods outlined in 1 to 3. The distribution material of MERVA ESA contains a sample that explains the interface and makes the setup for that interface. In DSLQKEY, any keys can be provided in the actual queue parameter list (for each queue of a multiple PUT or ROUTE) after inspecting the queue parameter list and the message in the data buffer. If the user exit does not provide keys, and no key has been provided by the requestor of the queue management service, DSLQMGT tries to get the keys in the way described in 1.

  No matter how the keys are provided, DSLQMGT uses the keys only when they are defined for the particular queue in the associated function-table entry.

- Creating 1 to 12 queue key table entries in main storage and provide the same information in the queue element status information.

  For later access of the stored queue elements, DSLQMGT provides a queue key table entry in main storage for each queue to which a queue element belongs. The queue key table entries of one queue are chained to the function-table entry in the sequence of the queue sequence numbers. Each queue key table entry contains:

  – The queue sequence number

  – The keys if defined for the function

  – The number of the data block where the queue element is stored

  – A status indicator

  Similar information is added to the queue element in the data block. For each queue to which the queue element belongs, the following status information is provided:
  – The queue name
  – The queue sequence number
  – The keys if defined for the function

  The information for automatic delete of the original message is also added to the status information if available.

  The status information is used to find the queue element in the data block and to recreate the queue key tables in a DSLQMGT restart.
- Starting 1 to 12 transactions if necessary.

  Each of the target queues of a single PUT, multiple PUT, or ROUTE with multiple PUT can have a transaction code defined (with or without a logical terminal name). If so, DSLQMGT starts this transaction if the message-processing function is in NOHOLD status, or if *ignore activated* (IGNACT) is specified, also in ACTIVATED status. A CICS START command or IMS change, insert, and purge requests are used. The status of the function is set to ACTIVATED. The status changes from ACTIVATED to NOHOLD if the transaction processes the queue completely, that is, when the queue is empty, or when GETNEXT requests reach the end of the queue.

  You can define a set of up to three related functions that are used by the transaction to process the queue. These three functions are always set to ACTIVATED, NOHOLD, or HOLD status at the same time.
- Post 1 to 12 event control blocks if necessary.

  Programs link-edited to DSLNUC via DSLNPTT can request posting of an event control block (ECB) after a message has been stored in a queue. Posting is performed after single PUT, multiple PUT, ROUTE with multiple PUT, and FREE (see “Retrieving a Message from a Queue”).

  The program that wants an ECB to be posted must indicate this to DSLQMGT by means of the SET request. Only one program can request posting for one queue. After each store request, DSLQMGT checks if an ECB is to be posted. The program that gets its ECB posted is later given control by DSLNUC to process the message just stored in the queue.

  When ECB posting is no longer necessary, the program owning the ECB must use the RESET request of DSLQMGT to stop posting.
- Delete the original queue element if automatic delete is requested.

  If a program retrieves a message from a queue, stores it in another queue, and then wants to delete it from the input queue, automatic delete must be used. This ensures that no messages are duplicated in case of a system breakdown. Automatic delete is indicated by giving the input queue name and input queue-sequence number in the store request. DSLQMGT then deletes the message in the input queue.

  A REPLACE request internally always performs a PUT and DELETE request.

Retrieving a Message from a Queue

Messages can be retrieved from MERVA ESA queues using one of the following request types:

- GET. The queue element to be retrieved is identified either by one of its keys or by the queue sequence number (QSN). Keys need not be unique in a queue, but the QSN is unique in each queue. If neither a key nor a QSN is specified, a GETNEXT is carried out.
- GETNEXT. A QSN is specified, and the queue element with the next higher QSN is to be retrieved. To start the retrieval with the first queue element of a queue, a QSN of zero must be specified.
- GETLAST. The queue element to be retrieved is the one with the highest QSN in the queue, that is, the last one in the sequence of queue elements.
- FREE. This request does not retrieve a queue element, but it resets the *in-service* indicator of a queue element that was retrieved with one of the GET requests.

When retrieving a message from a queue, the following steps are performed:

- Checking the HOLD status.

  A queue can be set to the HOLD status. A retrieval request for a queue in HOLD status results in the return code *queue empty*. To bypass the HOLD status, the request modifier *ignore hold* can be used.
- Finding the key or QSN in the queue key table considering the *in-service* indicator.

  To prevent a queue element from being accessed by more than one MERVA ESA application at a time, an *in-service* indicator is used by DSLQMGT to indicate that one application has already retrieved a message. Another retrieval request for the same message with the *in-service* indicator on is rejected.

  When retrieving a queue element, DSLQMGT scans the queue key table to find the key or QSN. When the key or QSN is found and the queue element is *in-service*, the request is rejected. For a GETNEXT request, the scan continues until a queue element is found that is not *in-service*.

  The *in-service* indicator can be bypassed with the request modifier *ignore in-service*. If the queue element is found and is *in-service*, the requestor gets the queue element and is informed about the *in-service* status.

  As the *in-service* indicator is lost after terminating MERVA ESA and starting it again, the request modifier *write-back* can be used. This modifier flags the queue element in the QDS for being read. When a queue element is later read again with the *write-back* indicator on, the requestor is informed about this status.

  The FREE request can be used to reset the *in-service* indicator and the *write-back* indicator of a queue element. If the *write-back* indicator is to be reset, the FREE request must read the relevant data block and write it back.

  The request modifier *free* can be used to *not* set the *in-service* indicator when retrieving a queue element. This can be useful if a program wants to retrieve the messages only without updating and further storing.
- Reading the data block from the QDS and locating the queue element.

  After the requested queue element has been found in the queue key table, DSLQMGT calls the associated I/O module to read the relevant data block from the QDS. When the data block is in main storage, DSLQMGT finds the queue element in the block using the queue name and the QSN that are contained in the status information of the queue element.

  If the queue element indicates that it is a large message, DSLQMGT calls DSLQLRG to get the message data from the large message cluster (LMC).
- Providing the message and status information for the requestor.

  After the queue element has been found, the message is moved to the requestor's buffer, and the keys and QSN are moved to the requestor's queue parameter list. The return code indicates if the message had the *in-service* or *write-back* indicator set on before this retrieval request. The *in-service* indicator is set in the relevant queue key table entry, unless the request modifier *free* was used.

Deleting a Message in a Queue

Messages are deleted in a MERVA ESA queue with the request type DELETE. The message to be deleted must be identified by its QSN. One of the keys can also be used, but the keys are not necessarily unique, and the wrong message could be deleted.

If a message was taken from a queue, stored in other queues with a single PUT, multiple PUT, or ROUTE request, it must be deleted by automatic delete, as described earlier to prevent duplication of messages if there is a task or system breakdown. The DELETE request must only be used in cases where automatic delete cannot be used; for example, when the messages of a queue are deleted after printing, writing to a sequential file, or when they are no longer needed.

When deleting a message from a queue, the following steps are performed (this is also true for automatic delete):

- Find the QSN or key in the queue key table.
- Read the data block from the QDS after the queue key table entry has been found. When the data block is in main storage, DSLQMGT finds the queue element in the block using the queue name and the QSN that are contained in the status information of the queue element.
- Remove the queue element from the data block. There are two cases:
  1. The queue element still belongs to at least one other queue. Then only the status information is removed from the queue element for the relevant queue, and the queue element remains in the data block.
  2. The queue element belongs only to the queue from which it is to be deleted. Then the queue element is removed from the data block. The information about used and free space in the data block and the byte map are updated. The free space is available for storing other queue elements.
- Remove the queue key table entry from the queue. The queue key table entry can be used again when storing a message into any queue.
- Write the data block back to the QDS to make sure that the queue element is not found again after a restart.
- If the queue element indicated that it was a large message, and it belonged only to the queue from which it was deleted, DSLQMGT calls DSLQLRG to delete the message data from the large message cluster (LMC) after the status information was deleted in the QDS data block and this data block was written back to the QDS.

Getting Status Information

You can get status information about a queue with the following request types:

- TEST request.

  The TEST request returns the following information in the queue parameter list:
  - Actual number of queue elements waiting in the queue
  - Threshold number of queue elements for the queue
  - Information about whether the queue threshold is reached by an appropriate return code
  - The highest queue sequence number (QSN) used in this queue

The same status information is obtained after each retrieval or single storage request.

- LIST request.

  The LIST request is used to get queue key table information for one queue. The requesting program specifies the number of queue key table entries to return, and the position in the queue where to begin the list. The list can be limited to queue key table entries that have the *in-service* indicator on, and/or key 1 and/or key 2 contain specific characters or sequences of characters.

  The following information is returned in the response buffer:

  – Name of the function queue
  – Actual number of queue elements waiting in the queue
  – Threshold number of queue elements for the queue
  – Number of end users that have currently selected the function
  – First and last queue sequence number (QSN) currently used in the queue
  – Highest QSN already used in this queue
  – Field names and lengths of the key fields defined for the queue
  – Length of one queue key table entry of this queue
  – Number of queue key table entries contained in the current response

  This information is followed by the queue key table entries that match the LIST request. Each queue key table entry contains:

  – The QSN of the message
  – The number of the data block where the message is stored
  – The *in-service* status
  – The keys of the message
  – The large message indicator

  The highest QSN of the current LIST response is returned in the queue parameter list.

A reason code indicates the end of the queue to the requesting program.

Unique Message Reference

The unique message reference (UMR) option can be used to assign a unique identification to each message that enters the MERVA ESA queues. To do so, UMR=YES is specified in the MERVA ESA customization parameters (DSLPRM). The processing of the unique message references is controlled by DSLQMGT.

The unique message reference contains:

Identifier
: 8 characters from the NAME parameter of DSLPRM. The identifier is used to determine if a message has a UMR in the local MERVA ESA system (local UMR), or if a UMR is from a different MERVA ESA system, that is, the UMR has a different identifier.

Sequence number
: 8 digits in the range from 00000001 to 99999999. The numbers are assigned in ascending order, and after 99999999 the number wraps around to 00000001.

System date
6 digits in the format *YYMMDD*. This is the date when this UMR was assigned.

System time
6 digits in the format *HHMMSS*. This is the time when this UMR was assigned.

System date and time indicate when a message was stored in a MERVA ESA queue the first time, and also helps to distinguish equal sequence numbers after a wraparound.

The UMR is assigned when a message without a local UMR is stored in a MERVA ESA queue, and it is stored in the queue together with the message. When the message is retrieved from the queue, the UMR is available in the field DSLUMR of the MERVA ESA internal message buffer (TOF).

If a message contains a UMR from a different MERVA ESA system, this UMR is kept and the local UMR also assigned. When a message contains more than one UMR, the local UMR is always contained in the first data area of the DSLUMR field.

When storing a message into a MERVA ESA queue, the assignment of the UMR can be modified by using the request modifier *newumr* to force the assignment of another UMR, or by using the request modifier *noumr* to not assign a UMR.

When unique message references are used in a MERVA ESA system, the UMR sequence number of the message is displayed on the end-user screen or in the hardcopy print. All parts of the UMR can be seen on a help panel when the command **show umr** is entered.

The UMR can be preassigned at a display station during creation of messages by specifying UMR=(YES,IMM) in the MERVA ESA customization parameters DSLPRM. In this case, the End-User Driver (DSLEUD) requests a UMR from DSLQMGT and displays it together with the new message.

When processing messages with the batch program DSLSDO, the UMR can be added to the line format used for the sequential file. If the UMR is not added to this line format, the UMR is lost. To keep the UMR with the message, the MERVA ESA queue format can be used (the *MERVA for ESA Operations Guide* describes how to run DSLSDO).

The UMR sequence number (with or without the UMR identifier) can be used as key field in MERVA ESA queues and for routing decisions.

The last UMR assigned in a MERVA ESA system is displayed during MERVA ESA startup and termination, and can be seen in the response to the MERVA ESA operator command **dq status**.

The last UMR can be adjusted using the LASTUMR control statement during the FORMAT or MODIFY function of the queue data-set utility DSLQDSUT.

Queue Trace

DSLQMGT can trace all requests with queue elements (general queue trace), or the requests for individual queues. The queue trace is available in the MERVA ESA journal with the journal record identification 23 (X'17'). The queue trace is helpful when testing user applications.

The state of the general queue trace is controlled by the QTRACE parameter of the DSLPARM macro, and you can switch the state while MERVA ESA is running. You can define the state for the queue trace of individual queues only while MERVA ESA is running (see the **qswitch** command in the *MERVA for ESA Operations Guide*). The following states are available:

OFF
The queue trace is inactive.

SMALL
The queue trace is active. The queue trace record will contain the queue parameter list showing which queue request was executed for which queue(s), the queue element prefix (status information) that shows to which queues the retrieved or new queue element belongs, and the keys and QSNs the queue element has in each of these queues.

LARGE
The queue trace is active. The queue trace record will contain the same information as for SMALL, and in addition the whole queue element. If the message is a small or normal message, all message data is shown. If the message is a large message, only the status information is shown. This type of queue trace should only be used when a sufficiently large journal data set is available.

Details of the layout of the queue trace records can be found in “Appendix A. Journal Record Layouts” on page 185.

User Exits in DSLQMGT

The following user exits are available in DSLQMGT:

DSLQUMR
This program is called by DSLQMGT when a new UMR is assigned during a store request. The queue parameter list with the new UMR and the message buffer of the requesting program are available. DSLQUMR allows for adding the message to another file or database, using the UMR as a unique identification.

DSLQKEY
This program is called by DSLQMGT when determining the keys for a queue element in a store request. DSLQKEY can inspect the queue parameter list and the message data and supply keys in the queue parameter list. When DSLQKEY is called by DSLQMGT is explained 77.

DSLQTRA
This program is called by DSLQMGT with the information prepared for the queue trace LARGE (see “Queue Trace”), no matter which level of queue trace is used. DSLQTRA can process this information and indicate by a return code if DSLQMGT should write the queue trace or not.

DSLQPUT
This program is called by DSLQMGIO before a QDS block is written to the queue data set. The complete QDS block is available to DSLQPUT, for example, to write it to a duplicate QDS in addition to the one provided by MERVA ESA. DSLQPUT allows for creating a duplicate QDS in a remote location.

The distribution material of MERVA ESA contains samples of these user exits that explain the interfaces and make the setup for these interfaces.

End of Product-Sensitive Programming Interface

Large Message Cluster (LMC)

The data part of large messages is not stored in the queue data set (QDS) but in the large messages cluster (LMC). The large message cluster (LMC) is a VSAM KSDS (key-sequenced data set) cluster with variable record length.

The maximum length of a record is defined by the record size parameter and depends on the maximum size of a control area for the disk type used.

For record sizes bigger than the defined control interval size, 'spanned' must be specified for the LMC.

The LMC can be defined with primary allocation only or with primary and secondary allocation.

The status and statistics of the large message cluster (LMC) can be displayed by the MERVA ESA operators using the operator commands **dlmc** (display large message cluster) and **dlmct** (display large message cluster for tuning).

Only one LMC can be associated with a QDS. There is no LMC duplicate. If duplicate write is required, the hardware-supported Dual Copy facility must be used.

LMC and QDS are considered to be a unit. The QDS log record and the initial load record (ILR) of the LMC contain the same timestamp for identification.

Large Message Service Program DSLQLRG

The MERVA ESA large message service program DSLQLRG is only called by the queue management program DSLQMGT for the following services:
- Initialization
- Termination
- Storing the data of large messages
- Retrieving the data of large messages
- Deleting the data of large messages

Routing

Routing determines the next function queues where a message will be stored. The next function may be a fixed name defined for the message-processing function, or the next function may be a variable, depending on the message content.

Variable routing consists of:
- Defining routing criteria in routing tables
- Interpreting the routing tables with the routing scanner DSLRTNSC

The result of routing is up to 12 queue names used to store the message.

Routing tables can be tested with the routing trace.

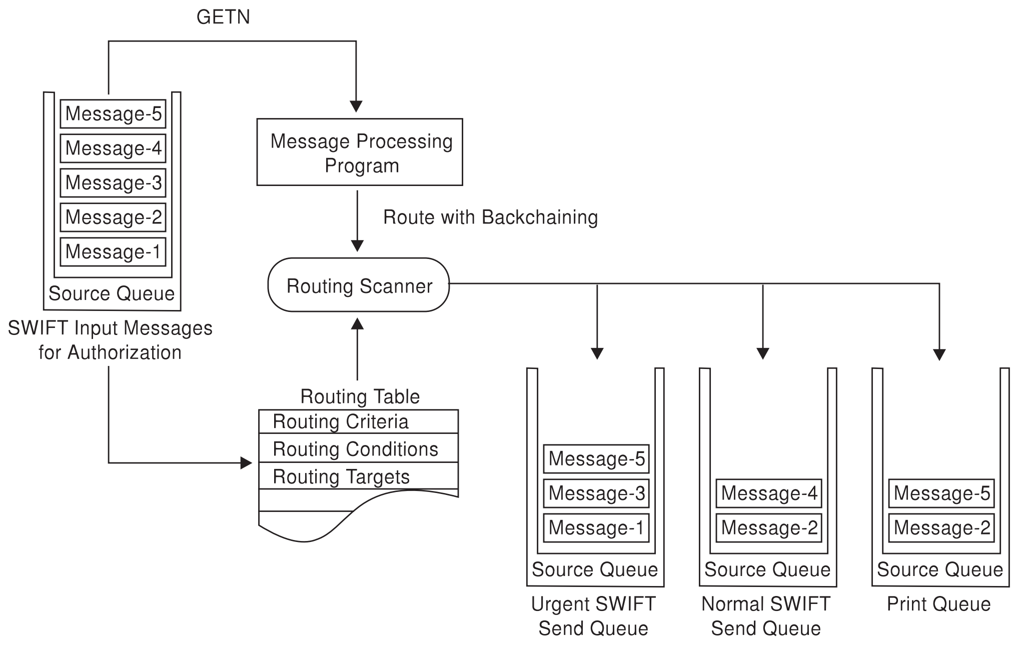


Figure 12. Routing a Message

Definition of Routing Tables

In MERVA ESA, routing criteria are defined in routing tables. The basic functions of routing tables are:

- Defining variable fields to be used later in the routing table, giving each a name. The data of a variable field is taken from a field of the MERVA ESA internal message buffer (TOF) or from a literal. For a TOF field, a displacement and length can be specified to take only a part of the data.

  Defining variable fields includes branching to labels within the routing table if:

  – The field is found, empty, or not found.

  – The field is considered empty also if the displacement is greater than the actual length of the TOF field data, so there is no data left for the variable field.

  Up to 20 variable fields, each of up to 32 characters in length, can be defined in one routing table.

- Testing the contents of the defined variable fields against other defined variable fields or literals. You can specify exactly how to do so by testing the two operands:

  1. As they are, for example, with different lengths
  2. In the length of the shorter operand
  3. In the length of the longer operand; the shorter one is then padded with binary zeros
  4. As numbers with a decimal comma (amounts); the operands are adjusted and a numeric comparison takes place

  The result of the test is used to branch to labels within the routing table. The branch conditions are:

  – True

  - False
  - Variable field not found

  For a numeric test, the false label is used if the adjustment of the operands leads to an overflow.
- Setting a target function. The function name can be a concatenation of up to eight variable fields or literals. A maximum of 12 target functions can be set when processing the routing table for a particular message. Thus, the message can be routed to up to 12 MERVA ESA queues or functions.

  Setting a target function includes branching within the routing table for the following conditions:
  - Unconditionally after the function has been set.
  - Variable field not found.
  - Too many target functions: When 12 target functions are set and another one is attempted, it is ignored. Processing branches to the final routing decision. The MERVA ESA routing trace will show that you tried to set too many functions.
  - The target function is not found in the MERVA ESA function table. All target functions found up to then are ignored, and the final routing decision is taken. The routing trace shows the incorrect function.
- Drop variable field names. In complex routing tables more than 20 variable field names may be needed. In order to not exceed the possible number of 20 variable fields, the field names that are no longer needed can be dropped to make room for new field definitions.
- Make a final routing decision. The last entry of each routing table may contain a target function name. When processing of the routing table leads to an error or does not find any target function, a final routing decision is attempted as follows:
  - When the calling function was specified and it contains a NEXT function, the NEXT function is verified in the MERVA ESA function table.
  - If no NEXT function is available, the target function of the final routing table entry is verified.

  If the NEXT or final target function is valid, DSLRTNSC considers the routing process successful but a warning return code and a reason code are given.

  If neither the NEXT nor the final target function is valid, the routing process has failed, and an error return code and a reason code are given showing the original error.

The routing target functions are used by DSLQMGT to process a multiple PUT, and they are returned to the requesting program in the queue parameter list or the queue parameter list extension.

The logic flow of sample routing tables is explained in the *MERVA for ESA Customization Guide*.

Routing Scanner Program DSLRTNSC

The routing scanner DSLRTNSC is called by queue management. DSLRTNSC performs the following functions:

- Initialization: Load the routing tables defined in the MERVA ESA function table. Get main storage for the routing trace depending on the specification of the JRNBUF customization parameter in module DSLPRM.
- Processing: Perform routing. Trace routing if requested.
- Termination: Delete the routing tables defined in the MERVA ESA function table. Release the storage needed for the routing trace.

Routing Trace

While interpreting a routing table, DSLRTNSC can, on request, trace all its activities. The routing trace is available in the MERVA ESA journal with the journal record identification 25 (X'19'). When printed with Access Method Services of VSAM (IDCAMS), the routing trace can easily be understood from the character part of the printout. You can also use the batch utility DSLBA13R to print the routing trace. DSLBA13R enables you to specify that only routing trace entries of the journal should be printed. See the appendix of the *MERVA for ESA Application Programming Interface Guide* for details.

There is a general routing trace state and a routing trace state of individual routing tables. You can define the initial state of the general routing trace in the MERVA ESA customizing parameters (see the RTRACE parameter described in the *MERVA for ESA Macro Reference*), and you can switch the state while MERVA ESA is running. You can define the state of the routing trace of individual routing tables only while MERVA ESA is running (see the **rswitch** command in the *MERVA for ESA Operations Guide*). The following states of the routing trace are available:

OFF	The routing trace is inactive.
SEVERE	The routing trace is active, but traces only the activities that led to a severe error. If no errors or only warning errors are detected, the trace is not written to the MERVA ESA journal.
WARNING	The routing trace is active, but traces only the activities that led to a warning or severe error. If no errors are detected, the trace is not written to the MERVA ESA journal.
ALL	The routing trace is active, and all the activities are traced and can be found in the MERVA ESA journal. ALL can be used to test routing tables when customizing MERVA ESA.

The routing trace will contain one or more entries for each routing table entry processed. Each routing trace entry has a length of 32 bytes, and it is aligned in the MERVA ESA journal so that it is just in one line of the IDCAMS printout.

Most routing trace entries show, in the first 3 bytes, a number that shows which routing table entry was processed. During the assembly of a routing table, each DSLROUTE macro gets a number that can be seen in the assembler listing, or you count the DSLROUTE macros in the source program. Using this together with the abbreviated function code in bytes 5 to 7, it is easy to follow the sequence of events through the routing table. Whenever a condition or branch label is used in a routing table entry, the routing trace shows with the next trace entry which label was used.

The routing trace entries are described in detail in “Appendix G. Layout of the Routing Trace Entries” on page 209.

Special TOF Fields for Routing Decisions

During the processing of MERVA ESA, some fields in the MERVA ESA internal message buffer (TOF) contain special information. The use of these fields in a routing table for routing decisions is explained in the following.

MERVA Link provides the field EKACLASS for routing decisions (see "MERVA Link Message Classes" on page 142 for details).

The Unique Message Reference (UMR)

When a message contains a unique message reference (UMR), the field DSLUMR can be used for routing decisions. If the message contains more than one UMR, the first 10 are available for routing decisions.

If the current routing causes a UMR to be assigned, the value is already available as the first data area of the field DSLUMR. However, if storing of the message fails, this UMR is assigned to the next message.

If a ROUTE request specifies the modifier *route only*, the UMR is only available for the routing decision if the message already had a UMR.

The field DSLUMR has the following layout:

Offset	Length	Subfield	Contents
0	28	DSLUMR	Unique message reference
0	16	DSLUMRIN	Identifier and sequence number
0	8	DSLUMRID	Identifier
8	8	DSLUMRNO	Sequence number
16	6	DSLUMRDA	Date YYMMDD
22	6	DSLUMRTI	Time HHMMSS

The MSGTRACE Field

The MSGTRACE field is created by all MERVA ESA application programs when they have processed a message and passed it to DSLQMGT for storing in queues. Another data area is added by these programs for each further processing step. Therefore the MSGTRACE field shows the path of a message through MERVA ESA.

Each data area consists of a mandatory part and an optional part that is generated only when the information is available. The mandatory part has a length of 32 bytes and a fixed format.

The following subfields are defined in the field definition table for each data area of the MSGTRACE field:

Offset	Length	Subfield	Contents
0	8	MSGTRUID	User ID when the MSGTRACE field is written by DSLEUD. The other MERVA ESA application programs write their program name into this field, for example, DSLSDI.

Offset	Length	Subfield	Contents
8	8	MSGTRFUN	Input queue function processed. • For DSLSDI, this is the intermediate queue function. • For DWSDGPA and generated messages and messages received from the SWIFT network, DWSDGPA is used. • For Telex Link and messages received from the telex network, TELEX is used. • For MERVA Link, the send queue or the control queue is used. • For MERVA-MQI Attachment, the send queue, the control queue, the wait queue, or the word MQSERIES for messages received from the MQSeries is used.
16	4	MSGTRERR	Error reason code of the MERVA ESA message format service. The sample transactions DSLCESTR and DSLCSETR give a component code followed by the reason code: **C** For a DSLCSE1 error **C1** For a DSLCES1 error **C2** For a DSLCES2 error **0** For a DSLMMFS error **N** For a DSLNIC error **Q** For a DSLQMGT error **S** For a DSLSRVP error **T** For a DSLTOFSV error No error is indicated by 0000.
20	6	MSGTRDAT	Date in the form YYMMDD (year, month, day).
26	6	MSGTRTIM	Time in the form HHMMSS (hour, minute, second).
32	8	MSGTRTRM	Terminal name. Shows the screen terminal where the message was processed by an end user. This subfield is optional.

You can use the information in these subfields during message routing, to make routing dependent on the last processing user or program, the last processed function, or the error reason code.

The MSGOK Field

The MSGOK field contains the parameters of the end-user commands **ok** and **route**. The field can be tested to find out if the message is authorized for onward routing. In most cases, the authorization controls access to an external network.

For example, if the MSGOK field contains YES after an **ok yes** command, the message is authorized to be sent to the SWIFT network and is therefore routed to the SWIFT ready queues. But, if the MSGOK field contains NO after an **ok no** command, the message is *not* authorized to be sent to the SWIFT network and is therefore routed to a verification queue for correction.

If the MSGOK field contains another value, it can be either a function name, or a signature of up to 8 characters. The function name can be verified if it is permitted with the command. A signature can be used for a more specific routing decision, that is, it can be translated into a function name.

The MSGOK field is deleted from the TOF when the message is retrieved again from a queue by an end user to provide a new signature. When the message is retrieved by a program, the last MSGOK field is still available.

The MSGACK Field

The MSGACK field is used by SWIFT Link to indicate the status of a message sent to or received from the SWIFT network. The programs DWSDGPA and DWSDLSK write the information to the MSGACK field.

The error messages of DWSDGPA, DWSDLSK, and DWSAUTP mentioned below are described in detail in *MERVA for ESA Messages and Codes*.

MSGACK Field for SWIFT Input Messages: A message sent to the SWIFT network can contain the following information in the MSGACK field:

- An error message starting with DWS. In this case, an error was detected when preparing the message for sending. DWS6*nn* indicates that the error was detected by DWSDGPA, DWS7*nn* indicates that the authentication failed, and the error was detected by the authentication program DWSAUTP.
- The SWIFT acknowledgment message starting with {1:A21 for the application control (APC), or with {1:F21 for the financial application (FIN). In this case, the message was sent to the SWIFT network and the system acknowledgment was received.

  The acknowledgment is positive if the field 451 contains a “0”.

  The acknowledgment is negative if the field 451 contains a “1”.

  This “0” or “1” is found at displacement 53 in the MSGACK field.

The field MSGACK1 is defined as a subfield of the MSGACK field. When the MSGACK field contains the SWIFT system acknowledgment (APDU 21), the field MSGACK1 can be read to get the ACK or NAK message in the SWIFT I format, that is, starting with the characters ACK or NAK.

MSGACK Field for SWIFT Output Messages: A message received from the SWIFT network can contain an error message starting with DWS. DWS6*nn* indicates that an error was detected by DWSDGPA. DWS7*nn* indicates the positive or negative result of the authentication as indicated by the digits *nn*, *nn* being the reason code of DWSAUTP.

MSGACK Field for SLS Session Key Messages: A session key message received from the SWIFT USE workstation can contain an error message starting with DWS6. The message contains at offset 8 the name of the program DWSDGPA or DWSDLSK that detected the error.

Chapter 10. Queue Services (DB2)

This chapter describes the queue services available in MERVA ESA with queue management using DB2. Since it is completely transparent to MERVA ESA application programs and end users whether MERVA ESA runs with queue management using VSAM or DB2, only the implementation differences are described in this chapter. For a general description of the MERVA ESA queue services, see “Chapter 9. Queue Services (VSAM)” on page 69.

The MERVA ESA DB2 queue services are used when storing messages to and retrieving messages from a DB2 database. This chapter describes the following aspects:

- The queues defined in the MERVA ESA function table (FNT)
- The DB2 objects, such as tables and indices
- The queue management program DSLQMGD
- The queue management I/O programs DSLQMDIO, DSLQMDLI, and DSLQMDXK

There is no special service for large messages.

Definition of Queues

With queue management using DB2, you can define the following additional attributes in the MERVA ESA function table DSLFNTT:

- PARTID to allow a partitioning index to be defined on the message table
- XKEYS to allow extra keys for a function

See the *MERVA for ESA Macro Reference* for details.

DB2 Objects

The following DB2 tables are used to store the messages:

DSLTQUEL	Queue element table. The table is used to store queue element control data.
DSLTQXDEF	Extra-key definition table. The table is used to define the extra keys for a queue.
DSLTQXKEY	Queue extra-key table. The table is used to store the extra-key data.
DSLTQBUSY	Busy table. The table contains one entry for each message currently in use (‘busy’).
DSLTQMSG	Message table. The table is used to store the message data.
DSLTQFUN	Function control table. The table is used to store control information about queues.
DSLTQSTAT	MERVA status table. The table is used to store the last used MSGTABLENO and UMR.

See “Appendix F. Layout of the DB2 Tables” on page 203 for a description of the layout of all used DB2 tables.

Queue Management Program DSLQMGD

Product-Sensitive Programming Interface

DSLQMGD controls the queue services.

For queue management using DB2 there are two modes of operation, either centralized or direct. When working centralized, which is the normal mode of operation, DSLQMGD is controlled by DSLNUC. Only for batch programs DSLSDxy and API programs for MVS is direct processing possible, that is, the programs can access queue services directly. You would then specify DSLPRM SDDB2=YES or switch the API customization parameter APICQDIR on.

DSLQMGD performs the following tasks:
- Initialization
- Servicing queue and status requests
- Processing the unique message reference
- Calling DSLQMDIO, DSLQMDLI, and DSLQMDXK for DB2 access
- Calling user exits
- Tracing queue operations
- Termination

Initialization of DSLQMGD for Central Processing

During the startup of MERVA ESA, DSLNUC requests the queue management INIT function to:
- Get main storage for DSLQMGD and the DB2 I/O programs DSLQMDIO, DSLQMDLI, and DSLQMDXK
- Call DSLQMDIO with request type INIT:
  1. Call DSLHLI2 to get the apropriate DB2 language interface (MVS only): CICS: DSNCLI, IMS: DFSLI000, BATCH: DSNHLI2
  2. Delete all central entries from the DSLTQBUSY table
  3. Reset all entries in the function control table DSLTQFUN to their initial state as defined in DSLFNTT (insert an entry for all functions defined in the DSLFNTT with STATUS=HOLD or TRAN=tran)
  4. Return the last UMR used (if no last UMR can be found, initialize the DSLTQSTAT table)
- Call the MERVA ESA routing scanner DSLRTNSC for initialization

Note: There is no special restart processing with queue management using DB2.

Termination of DSLQMGD for Central Processing

During the termination of MERVA ESA, DSLNUC invokes the queue management TERM function. The following steps are performed:
- Call DSLQMDIO with request type TERM:
  1. Return the last UMR used (if no last UMR can be found, initialize the DSLTQSTAT table)
  2. Call DSLHLI2 to release the DB2 language interface (MVS only)
- Free all main storage obtained during initialization
- Call the MERVA ESA routing scanner DSLRTNSC for termination

Servicing Queue Requests

See “Servicing Message Queue Requests” on page 75. Whether MERVA ESA runs with queue management using VSAM or DB2 is completely transparent to MERVA ESA application programs.

Getting Status Information

See “Servicing Message Queue Requests” on page 75. In addition to the TEST, STATUS, and LIST request you can get SQL error information with the SQLSTAT request.

Unique Message Reference

See “Unique Message Reference” on page 81. With queue management using DB2 the current UMR is also stored in the DB2 queue element table DSLTQUEL.

Commit

As far as commiting is concerned, there is no difference between queue management using VSAM and queue management using DB2: all queue management requests are committed immediately. However, API programs running with queue management using DB2 on MVS can specify that they want to handle the commit processing themselves. For more information about commit processing, refer to the ‘Advanced Topics’ chapter of the *MERVA for ESA Application Programming Interface Guide*.

Queue Trace

See “Queue Trace” on page 83. The queue trace facilities are the same whether you use DB2 or VSAM, but the trace information is different:

- For queue trace state SMALL only the queue parameter list is traced.
- For queue trace state LARGE the queue parameter list, DB2 return information, queue desriptors, and the message are traced.

The queue trace is written to the MERVA ESA journal with the journal record identification 26 (X'1A').

Details of the layout of the queue trace records can be found in “Appendix A. Journal Record Layouts” on page 185.

User Exits in DSLQMGD

See “User Exits in DSLQMGT” on page 83.

The following user exits are available in DSLQMGD:

- DSLQUMR
- DSLQKEY
- DSLQTRAB (corresponds to user exit DSLQTRA called by DSLQMGT)

The distribution material of MERVA ESA contains samples of these user exits that explain and set up the interfaces.

End of Product-Sensitive Programming Interface

Queue Management I/O Programs for DB2

The programs DSLQMDIO, DSLQMDLI, and DSLQMDXK are the interface between queue management control program DSLQMGD and the DB2 subsystem:

- DSLQMDIO

  DSLQMDIO is called by DSLQMGD for the following services:
  - Initialization
  - Termination
  - Status and Test
  - Storing of messages
  - Retrieving of messages
  - Deleting of messages
  - Starting of functions
  - Holding of functions
  - Commit and rollback of database changes
- DSLQMDLI

  DSLQMDLI is called by DSLQMGD for the following service:
  - List queue elements by QSN and key
- DSLQMDXK

  DSLQMDXK is called by DSLQMGD for the following services:
  - Insertion of extra keys for queue elements
  - Deletion of extra keys for queue elements

Routing

Routing determines the function queue or queues where a message is next to be stored. There is no difference in routing between queue management using VSAM and DB2. For more information about routing, refer to “Routing” on page 84.

Extra Keys

With queue management using DB2 you can specify that additional keys, called *extra keys*, should be stored for messages. You would specify XKEYS=YES for the function concerned in the function table DSLFNTT and define the extra keys in the DB2 table DSLTQXDEF. Note that a large number of extra keys may significantly reduce the performance of all insert, update, and delete operations. For more information about extra keys, refer to the *MERVA for ESA Customization Guide*.

Chapter 11. Message Processing

The MERVA ESA message processing components provide for the creation and further processing of messages that you use for communication within your business or outside your business.

MERVA ESA offers the following message processing components:

- Transaction programs:
  - End-user driver DSLEUD
  - MERVA Message Processing Client Server
  - Hardcopy print program DSLHCP
  - Checking and expansion program DSLCXT
- Batch programs:
  - Sequential data set input DSLSDI and DSLSDIR
  - Sequential data set load DSLSDLR
  - Sequential data set output DSLSDO and DSLSDOR
  - Sequential data set unload DSLSDUR
  - System printer program DSLSDY and DSLSDYR

End-User Driver DSLEUD

The end-user driver (DSLEUD) controls the interaction between a user and MERVA ESA at a display station. DSLEUD prepares panels and sends them to the display station and receives responses from the display station using CICS or IMS facilities.

A user invokes the end-user driver by entering the transaction code defined for DSLEUD in CICS or IMS at a display station, for example, DSLE.

In a MERVA ESA CICS installation, DSLEUD is a reentrant CICS pseudo-conversational transaction. DSLEUD uses the dynamic transaction backout (DTB) function of CICS. In a MERVA ESA IMS installation, DSLEUD is a reusable conversational transaction running in an IMS MPP region.

The DSLEUD transaction can only start if MERVA ESA is active. DSLEUD is *not* link-edited to DSLNUC, and uses therefore the MERVA ESA central services via the intertask communication facility.

The MERVA ESA end-user driver consists of the main programs DSLEUD and DSLEFUN, and the function programs.

Figure 13 on page 96 shows the main components of DSLEUD.

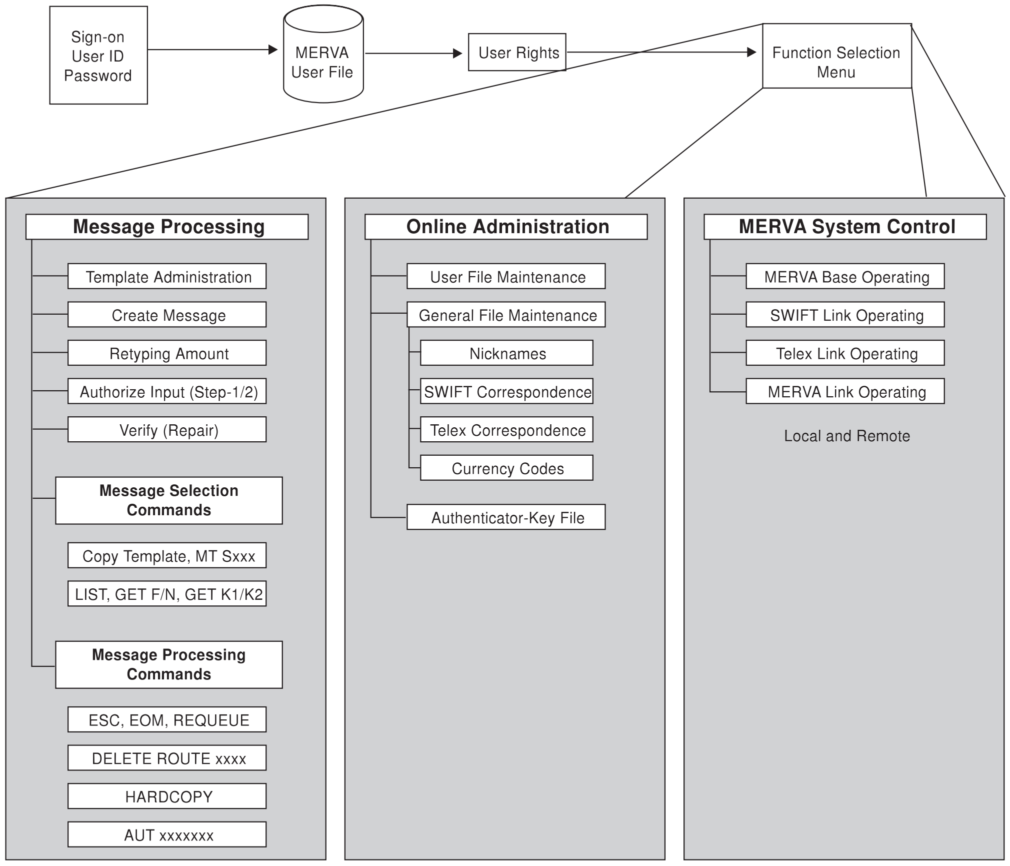


Figure 13. The Main Components of DSLEUD

For each conversation step with the display station, DSLEUD and DSLEFUN perform initialization, processing, and termination.

Initialization of DSLEUD

To initialize DSLEUD the following steps are required:

- Load modules that are needed by the programs that are link-edited to DSLEUD.
- Get main storage that is needed by the programs that are link-edited to DSLEUD, and by the MERVA ESA direct service programs that are used.
- If it is the continuation of a conversation, get the permanent storage or scratch pad area (SPA):
  - In CICS, get the SPA from CICS temporary storage (this can be in main storage or on disk).
  - In IMS, get the 320 bytes SPA from IMS (also at the start of a conversation), and call the MERVA ESA SPA file program DSLEOSPA to get the SPA from the MERVA ESA SPA file. MERVA ESA provides an alternative SPA file program DSLEOSPB that uses an IMS HDAM database instead of the BDAM data set for storing the SPA parts.

  The MERVA ESA SPA consists of four parts:
  - The working storage of DSLEUD
  - The TOF, or for a dynamic TOF the index part only
  - The logical data stream (LDS) buffer
  - The data part of a dynamic TOF

Each of the first three parts is always smaller than 32KB. The fourth part is used only when a large message is processed that does not fit into a single TOF buffer. The total length of the index part and the data part of a dynamic TOF will not exceed the value specified in the MAXBUF parameter in the customization parameters DSLPRM.

- Establish the MERVA ESA intertask communication. This step is only successful if MERVA ESA is ready.
- Initialize the MERVA ESA internal message buffer (TOF) and the MERVA ESA message format service (MFS).

Processing of DSLEUD

The processing routine of DSLEUD performs the following:

- Gets the input from the display station:
  - In CICS, the CICS RECEIVE command is used.
  - In IMS, a GET NEXT (GN) request is used.
- If it is not input from the signon panel, calls the user exit DSLEU002.
- Depending on the status of the conversation and on the input from the display station, performs the following:
  - User signon/signoff
  - Function selection
  - Processes user commands
  - Calls one of the function programs, listed in “Function Programs of DSLEUD” on page 99
- Prepares the response for the display station and sends it:
  - In CICS, the CICS command SEND is used.
  - In IMS, an INSERT (ISRT) request is used.

DSLEUD performs all processing for the display station. This means, if in one of the following sections another program prepares a panel, DSLEUD sends it to the display station.

The following describes the tasks of DSLEUD processing in more detail.

User Signon

Signon establishes the connection between a user’sdisplay station and MERVA ESA. After the conversation has started, there are two ways to sign on:

- DSLEUD sends the signon panel to the display station, and the user enters the user ID and password, and optionally a function.
- DSLEUD skips the signon panel and sends the function selection panel to the display station.

  This requires a preceding signon to CICS or IMS, and an application that starts DSLEUD (program-to-MERVA switch).

The following steps are performed during signon:

- The user exit DSLEU001 is called with the data received from the signon panel or data passed by the program-to-MERVA switch. DSLEU001 can check the user ID, password (if entered on the signon panel), and function and inform DSLEUD whether the signon is accepted or not.
- A signon request for the MERVA ESA user file program DSLNUSR is prepared.
- DSLEUD invokes the user file service via the MERVA ESA intertask communication.

- If DSLNUSR accepts the signon request, the user can continue with function selection. If not, the signon panel is displayed again with an appropriate error message.

The user can change the password during signon if the signon panel was displayed.

User Signoff

Signoff releases the connection between a user's display station and MERVA ESA, and ends the conversation. The user enters the **signoff** command, or an automatic signoff takes place after a severe error or when MERVA ESA has terminated.

The following steps are performed during signoff:

- The user exit DSLEU003 is called.
- A signoff request for the MERVA ESA user file program DSLNUSR is prepared.
- DSLEUD invokes the user file service via the MERVA ESA intertask communication.

  The signoff panel is displayed. After an error or MERVA ESA termination, the signoff panel shows an appropriate error message.

Function Selection

Function Selection is performed by the program DSLEFUN, which is part of DSLEUD. After successful signon or when returning from a function, DSLEFUN uses the user file record information to prepare the Function-Selection Menu for sending to the user's display station. After the user has selected a function, DSLEFUN:

- Prepares a function-selection request for the MERVA ESA user file program DSLNUSR.
- Invokes the user file service via the MERVA ESA intertask communication.

If DSLNUSR accepts the function-selection request, the selected function can be processed. If not, the Function-Selection menu is displayed with an appropriate error message.

Processing of User Commands

The user at the display station communicates with DSLEUD by means of user commands. All user commands are described in detail in the MERVA for ESA User's Guide. The user commands are processed by DSLEFUN as follows, it:

- Calls the user exit DSLMU004 with the command entered at the display station
- Calls the MERVA ESA command parser DSLNPAR for formal analysis of the command
- Calls the user exit DSLMU005 with the output from DSLNPAR
- Checks if the user is authorized to use the command (if the user is not authorized, the command is not executed)
- Processes the command:
  - If it is a **screen** command, it lets the MERVA ESA message format service execute the command.
  - If it is a **session** command, DSLEFUN executes the command.
  - If it is a **function** command, it calls the function program to execute the command.

Depending on the command, a command response or an error message may be available for the next panel to display.

Calling Function Programs

The function programs of DSLEUD are defined in the end-user driver program table DSLEPTT, and they are referenced in the functions defined in the MERVA ESA function table DSLFNTT. You can add your own function programs to the end-user driver by adding them to DSLEPTT and referencing them in a function of DSLFNTT.

The following function programs are available in DSLEPTT:

- DSLECMD, the operator command program
- DSLEFLM, the general files maintenance program
- DSLEMSG, the message processing program
- DSLEUSR, the user file maintenance program
- EKAEMSC, the MERVA system control facility program
- DWSEAUT, the authenticator-key file maintenance program of SWIFT Link
- User-written function programs

After having processed the user commands (see "Processing of User Commands" on page 98), DSLEFUN calls the function program that was defined for the selected function. The function programs are described in more detail in "Function Programs of DSLEUD".

Termination of DSLEUD

The termination of DSLEUD takes place at the end of every conversation step, after having given the next panel for the display station to CICS or IMS. The following steps are performed:

- Terminate the MERVA ESA message format service.
- Write the scratch pad area (SPA):
  - In CICS, write the SPA to CICS temporary storage (this can be in main storage or on disk)
  - In IMS, call the MERVA ESA SPA file program DSLEOSPA to write the SPA to the MERVA ESA SPA file, and use an INSERT (ISRT) call to give the 320 bytes SPA to IMS.

  After signoff of the user, the SPA is not given to CICS, and in IMS, both DSLEOSPA and IMS are informed that the conversation has ended.
- Release the MERVA ESA intertask communication.
- Delete all modules that have been loaded.
- Free all main storage.
- Return to CICS or IMS.

Function Programs of DSLEUD

The following describes the function programs of DSLEUD supplied with MERVA ESA.

Operator Command Program DSLECMD

The operator command program DSLECMD is invoked after the user has selected the CMD function (or another function, for which the program DSLECMD is specified).

DSLECMD prepares the operator command processing panel for sending to the user's display station. After the user has entered a MERVA ESA operator command (see the *MERVA for ESA Operations Guide* for details), DSLECMD performs the following steps, it:

- Prepares the command and response buffer for the MERVA ESA command server DSLNCS.
- Invokes the command service via the MERVA ESA intertask communication. After the command has been performed, DSLNCS provides the command response in the MERVA ESA command and response buffer for both correct and erroneous commands.
- Prepares the operator command processing panel with the command response and enables the user to enter the next command.

General File Maintenance Program DSLEFLM

When a user has selected the General File maintenancefunction FLM, the program DSLEFLM is invoked. General files are described in "General File Service" on page 168.

The authorization to maintain MERVA ESA General files is defined in the user file record of the user (see the *MERVA for ESA User's Guide* for details).

DSLEFLM prepares the File-Selection menu, which contains the identifications and descriptions of the files that have been defined as *available for online maintenance* in the MERVA ESA File Table. For a shared file, both the private records of the user, and the records common to all users, can be offered for maintenance.

After the user has selected a file for maintenance, the following tasks can be performed depending on the authorization of the user:

- Add new records
- Delete records
- Display records
- List records
- Replace records (that is, display a record, change it, and replace it in the file)

These tasks are described in detail in the *MERVA for ESA User's Guide*. Whenever DSLEFLM accepts an appropriate command from the user, DSLEFLM calls the MERVA ESA file service program DSLFLVP for the requested service. The user is informed whether the request was successful or not.

Message Processing Program DSLEMSG

When a user has selected one of the message processing functions, the program DSLEMSG is invoked. DSLEMSG prepares the message selection panel or the Queue List panel for message selection. When a message has been selected, DSLEMSG prepares the appropriate message panel.

Depending on the selected function, a new message can be created or an existing message can be retrieved from the associated queue. The user can enter, verify, or correct the message data, and tell DSLEMSG by means of user commands how to complete the message. This is described in detail in the *MERVA for ESA User's Guide*.

User-File Maintenance Program DSLEUSR

When a user has selected one of the user file maintenance functions USRn, the program DSLEUSR is invoked. User file maintenance requests the user to enter the password again for more security. To check the password, DSLEUSR invokes the user exit DSLEU004 and DSLNUSR via the MERVA ESA intertask communication. It can be customized in the DSLPRM module that DSLEU004 uses the services of a security manager (ESM or BSM) to check the password. If the password is correct, the user can maintain the user file.

Depending on the selected user file maintenance function, the user is allowed to perform some or all of the following tasks with user file records:

- Add new records
- Authorize records changed by another user
- Delete records
- Display records
- List records
- Reject changes to records
- Replace records (that is, display a record, change it, and replace it in the file)

These tasks are described in detail in the *MERVA for ESA User's Guide*. Whenever DSLEUSR accepts an appropriate command from the user, DSLEUSR prepares a request to DSLNUSR and invokes the MERVA ESA intertask communication for the requested service. The user is informed whether the request was successful or not.

MERVA System Control Facility Program EKAEMSC

When a user has selected the MERVA system control function MSC, the program EKAEMSC is invoked. EKAEMSC lets you:

- Issue MERVA ESA operator commands at the local MERVA ESA system, similar to the program DSLECMD (see “Operator Command Program DSLECMD” on page 99).
- Issue MERVA Link commands to control the MERVA Link connections at the local MERVA ESA system.
- Switch to a remote MERVA ESA system and both issue MERVA ESA operator commands and control the MERVA Link connections at that system.

EKAEMSC displays:

- A list of the MERVA Link application support processes (ASPs)
- The status and parameters of a specific MERVA Link application support process (ASP) and its message transfer process (MTP)
- A list of the local MERVA Link system control processes (SCPs)

The user can enter the MERVA ESA operator commands and the MERVA Link commands. Refer to the *MERVA for ESA Operations Guide* for details.

After the command has been performed, EKAEMSC prepares the appropriate panel with the command response and enables the user to enter the next command.

Authenticator-Key File Maintenance Program DWSEAUT

When a user has selected one of the Authenticator-key file maintenance functions AUTn, the program DWSEAUT of SWIFT Link is invoked. Online maintenance is only possible if the authenticator-key file program DWSAUTP has been initialized.

A user can maintain only the authenticator keys of the financial institution whose SWIFT bank identifier code (BIC) matches the origin identifier of the user’s user file record in the first 4, 6 or 8 characters.

Depending on the selected authenticator-key file maintenance function, the user is allowed to perform the following tasks with authenticator-key file records:

- Add new records
- Authorize records changed by another user
- Delete records
- Display records
- Exchange keys in records
- List records
- Reject changes to records
- Replace records (that is, display a record, change it, and replace it in the file)

These tasks are described in detail in the *MERVA for ESA User's Guide*. Whenever DWSEAUT accepts an appropriate command from the user, DWSEAUT prepares a request to the authenticator-key file program DWSAUTP and invokes the MERVA ESA intertask communication for the requested service. The user is informed whether the request was successful or not.

With the introduction of the SWIFT Bilateral Key Exchange (BKE), the need for using the functions of DWSEAUT will reduce as the authenticator keys will mostly be generated automatically without user intervention.

DSLEUD User Exits

The following user exits are available in the MERVA ESA end-user driver:

DSLEU001	Called at signon time. You can check the user ID and password with this user exit. RACF (MVS only) can be invoked to control signon. Associated user applications can be started.
DSLEU002	Called whenever terminal input is available after signon. This exit enables processing of user applications.
DSLEU003	Called at signoff time. RACF (MVS only) can be invoked. User applications can be stopped.
DSLEU004	Called when you enter the user file maintenance. You can check the password with this user exit. It can be customized in the DSLPRM module that DSLEU004 uses the services of a security manager (ESM or BSM) to check the password.

For processing in DSLEUD, the MFS user exits described in 68 are also available. Refer to the *MERVA for ESA Customization Guide* for more information about these user exits.

MERVA Message Processing Client/Server

The MERVA Message Processing Client/Server lets users at a MERVA Message Processing Client workstation process messages in a MERVA ESA system. The workstation can communicate with the server using APPC or TCP/IP communication protocols. APPC support is for LU Type 6.2 mapped conversations with no synchronization. For information about MERVA Message Processing Client, refer to the *MERVA ESA Components Message Processing Client for Windows NT User's Guide*.

The workstation server is comprised of two components:

- A listener, which controls the connection of workstations to MERVA ESA

- A transaction, which manages a conversation with a workstation after a connection has been established

The APPC solution for the CICS environment is implemented as a CICS transaction, all other solutions are implemented as batch programs. In CICS there is no listener, the listener service is performed by CICS itself. The components are summarized in the following table:

Table 1. MERVA Message Processing Client/Server

Environment	Listener	Transaction
APPC CICS	—	DSLAFM01
APPC/MVS	DSLAFA01	DSLAFM01
TCP/IP MVS	DSLAFATM	DSLAFMTM
TCP/IP VSE	DSLAFA04	DSLAFMTM

MERVA ESA Permissions for MERVA Message Processing Client Users

The MERVA Message Processing Client Server uses the MERVA ESA user file dervice, DSLNUSR (see “User File Service” on page 163), to manage client signons. All MERVA ESA users, whether accessing MERVA ESA from a host terminal or a client workstation, must be defined in the MERVA ESA user file.

Maintenance of the user file cannot be carried out from a client workstation, the end-user driver (DSLEUD) must be used (see “End-User Driver DSLEUD” on page 95).

The number of conversations that can be active simultaneously is limited by the CLIENTS parameter of the module DSLPRM. The MERVA Message Processing Client is a usage-based feature and must be licensed separately. See your IBM sales representative for information about how to order the appropriate number of authorized copies.

CICS LU 6.2 Transactions

In the MERVA ESA CICS environment, the MERVA Message Processing Client server is a reentrant CICS transaction that is initiated by a request from a client for conversation allocation. The transaction remains active for the duration of the conversation, and terminates following deallocation by the client from MERVA ESA.

Each workstation must be defined to CICS as a CONNECTION resource, and the logical link as a SESSION resource.

APPC/MVS LU 6.2 Transactions

For MERVA ESA IMS installations, an alternative MERVA Message Processing Client server solution is provided by the APPC/MVS server DSLAFA01. This program runs in an MVS batch region and functions as an APPC transaction scheduler for the DSLAFM01 APPC transaction program.

DSLAFA01 can be initiated as a started task, or by the MVS batch initiator. It registers with APPC/MVS as a transaction server for any allocate requests from MERVA Message Processing Clients. The LU and TP names for which DSLAFA01 registers are defined in parameter WSASRV of module DSLPRM.

DSLAFA01 attaches the transaction program DSLAFM01 as a subtask on receipt of a client request for APPC conversation allocation. One subtask is initiated for each conversation.

DSLAFA01 terminates when it recognizes a MERVA ESA termination, and can also be terminated by the system operator.

TCP/IP Listener Program DSLAFATM

The TCP/IP solution is similar to the APPC/MVS solution. The listener program, DSLAFATM, runs in an MVS batch region and functions as a transaction scheduler for the DSLAFMTM transaction program.

DSLAFATM can be initiated as a started task, or by the MVS batch initiator. It registers with TCP/IP as a concurrent server for any connection requests for the port number specified in parameter WSTSRV of module DSLPRM.

DSLAFATM attaches the transaction program DSLAFMTM as a subtask on receipt of a client connection request. One subtask is initiated for each conversation.

DSLAFATM terminates when it recognizes a MERVA ESA termination, and it can also be terminated by the system operator.

In the VSE environment, the number of simultaneous conversations is limited by the number of subtasks the operating system supports.

Hardcopy Printer Program DSLHCP

The MERVA ESA hardcopy printer program DSLHCP prints messages, contained in a MERVA ESA queue, on a hardcopy printer.

- **Under CICS**, DSLHCP is a multithread CICS transaction that is started with a CICS START command using the transaction code and the logical terminal name specified in the relevant function table entry. The transaction code must be defined in the CICS transaction definitions, and the terminal name must be specified in the terminal definitions.
- **Under IMS**, DSLHCP is a non-conversational MPP that is started by inserting a message into the IMS message queue using the transaction code specified in the relevant function table entry. DSLHCP uses the logical terminal name of the function table entry for printing messages. Both the transaction code and the logical terminal name must also be specified in the IMS nucleus.

If there is an entry for the transaction code in the MERVA ESA transaction table, and if this entry specifies a logical terminal (LTERM), then this specification takes precedence over any logical terminal specified in the function table entry, and issuing the **change function** command to try to change the logical terminal has no effect.

DSLHCP is started automatically by DSLQMGT when a message is written to the MERVA ESA queue with which the printing transaction is associated. The function must be in NOHOLD status. DSLHCP can also be started by the MERVA ESA operator using the MERVA ESA **sf** (start function) command.

Messages are printed according to the specifications of the following parameters of the function table entry of the input queue:

MSGID	This parameter determines which parts of a message are to be printed, if available: • The MERVA Link control information • The telex header and other telex information • The SWIFT message • Other information
PRFORM	This parameter determines the format of the printing.

The program DSLHCP performs address expansion according to the specifications in the function table entry of the input queue. Alternatively, address expansion can be done before routing the messages to the hardcopy print queue, for example, using the program DSLCXT (see "Transaction for Message Checking and Expansion DSLCXT").

The following steps are performed for each message:

- Get the message from the input queue.
- Call the user exit 24 (DSLMU024) for additional processing of the message or to skip the message.
- Check the message for formal correctness if the function of the hardcopy print queue specifies CHECK=YES. If errors are found, error messages are printed at the end of the message.
- Format the message for the printer as specified by MSGID and PRFORM parameters.
- Call the user exit 27 (DSLMU027) for manipulation of the output data stream. Refer to the *MERVA for ESA Customization Guide* for more information on this user exit. DSLMU027 is called:
  – Before the first output segment is prepared. Additional initialization of the output device may be performed, for example, sending a FORMFEED to the printer, or modifying the device characteristic. In IMS, the exit is called before the printer terminal is selected by an IMS CHANGE call. The printer name can be changed by overwriting the field TUCBLTN.
  – For each output segment. The output segment may be modified or routed to an additional destination, for example, a spooling system.
  – After the last output segment has been sent. Additional termination steps on the output device may be performed.
- Send the message panels to the hardcopy printer using the CICS send command or the IMS insert (ISRT) request.
- Delete the message in the hardcopy print queue if the function of this queue specifies KEEPMSG=NO. Do not delete the message if KEEPMSG=YES is specified. If the messages are not deleted, DSLHCP does not repeat printing the same messages if it is started again automatically by DSLQMGT. However, if the MERVA ESA operator starts DSLHCP with the **sf** (start function) command, printing starts with the first message in the hardcopy print queue.

If the execution of DSLHCP fails during processing of a message, this message is printed again the next time DSLHCP is started for this hardcopy print queue.

Transaction for Message Checking and Expansion DSLCXT

The MERVA ESA transaction for message checking and expansion (DSLCXT) checks, expands, and routes messages. DSLCXT can also be used to transform messages from the tokenized format to the external line format, or vice versa.

DSLCXT is started automatically by DSLQMGT when a message is written to the MERVA ESA queue with which DSLCXT is associated. The function must be in NOHOLD status. DSLCXT can also be started by a MERVA ESA operator with the **sf** (start function) command.

Under CICS, DSLCXT is a multithread CICS transaction that is started using a CICS START command with the transaction code specified in the relevant function table entry. The transaction code must also be defined in the CICS transaction definitions.

Under IMS, DSLCXT is a non-conversational MPP that is started by inserting a message into the IMS message queue using the transaction code specified in the relevant function table entry. The transaction code must also be specified in the IMS nucleus.

The following steps are performed for each message:

- Get the message from the input queue.
- Call the user exit 23 (DSLMU023) for additional processing or to skip or delete the message. Refer to the *MERVA for ESA Customization Guide* for more information about this user exit. The number of the user exit can be changed by specifying the required number in the UAPL parameter of the function table entry.

  If one of the following code word values is specified in the UAPL parameter, a special user exit processes message transformations for the external line format:

ELF	The message is transformed from tokenized format to external line format.
ELF+	The message in external line format is added to a message in tokenized format.
TOF	The message is transformed from external line format to tokenized format.
TOF+	The message in tokenized format is added to a message in external line format.

- Carry out expansion if the function of the input queue specifies expansion. For example, the SWIFT bank identifier codes (BIC) are expanded to the full address.
- Check the message for formal correctness if the function of the input queue specifies CHECK=YES.
- Route the message and delete it from the input queue. The routing decision can depend on the result of checking.

If DSLCXT execution fails during processing of a message, DSLQMGT ensures that this message is not lost nor duplicated when DSLCXT is started the next time for this message processing function.

Sequential Data Set (SDS) Batch Programs

MERVA ESA provides the following batch programs for sequential data set (SDS) processing:

- For input: DSLSDI and DSLSDIR
- For output: DSLSDO and DSLSDOR
- For loading: DSLSDLR
- For unloading: DSLSDUR

- For printing: DSLSDY and DSLSDYR

The batch programs run in a region or partition not used by MERVA ESA. They can run only when MERVA ESA is active, because they use central services via the MERVA ESA intertask communication.

Runtime parameters for the batch programs:
- DSLSDI, DSLSDO, and DSLSDY are specified with the PARM parameter of the JCL EXEC statement.
- DSLSDIR, DSLSDOR, DSLSDLR, DSLSDUR, and DSLSDYR are specified in the DD statements SYSTSIN and SYSIPT in the form KEYWORD=VALUE.

Refer to the *MERVA for ESA Operations Guide* for details.

The input or output device is allocated while the batch program is running.

SDS Input Program DSLSDI

DSLSDI reads a batch of messages from a sequentialdata set and transfers them to queues of MERVA ESA message processing functions.

The operational unit for DSLSDI is always a complete batch of messages, that is, a complete data set with messages. If necessary, DSLSDI performs a restart on this basis.

Processing takes place in two stages:

Stage 1

The messages are read from the data set and stored in an intermediate queue. This queue must be reserved exclusively to this run of DSLSDI. It is possible to run several DSLSDI programs simultaneously if each of them has its own reserved intermediate queue.

An intermediate queue for DSLSDI must never be the target of routing, because DSLSDI deletes all messages that have entered that queue in any way other than through DSLSDI.

For each message, the following steps are performed:

1. Get the message from the input data set on tape or disk, using the queued sequential access method (QSAM).

   Usually there is one message in one logical record. In addition, MERVA ESA supports segmentation of messages in which case one message may span more than one logical record of the input data set. Whether segmentation of messages is used or not is determined by a processing parameter in the JCL EXEC statement. When it is used it applies to all records in the input data set. In this case the message length can be larger than 32KB.

   The first byte in each record is the segment indicator. This byte can contain one of the following values:

0 (X'F0')	Only segment of a message
1 (X'F1')	First segment of a multi-segment message
2 (X'F2')	Last segment of a multi-segment message
3 (X'F3')	Middle segment of a multi-segment message

   The logical records belonging to one message are assembled internally.

2. Present the message to the message format service for transformation into the MERVA ESA internal format (TOF format). The PARM parameter of the JCL EXEC statement specifies the format of the input record.
3. If the intermediate function queue specifies CHECK=YES, perform formal checking of the message. For a message in queue format no checking is performed.
4. An incorrect message is handled according to the specification of the disposition parameter (ACCEPT, DROP, or CANCEL).
5. Call the user exit 20 (DSLMU020) for additional processing of the message, or to skip the message. Refer to the MERVA for ESA Customization Guide for more information on this user exit.
6. Store the correct or accepted message in the intermediate function queue.

After the last message has been stored in the intermediate queue, DSLSDI stores a specific restart message at the end of this queue for the case of a system breakdown.

Stage 2

The messages are routed from the intermediate queue to other MERVA ESA queues. For each message, the following steps are performed:

- Get the message from the intermediate queue.
- Route the message according to the routing criteria defined for the intermediate function.
- Delete the message in the intermediate queue.

When DSLSDI gets the restart message from the intermediate queue, this message is deleted and processing is complete.

If the processing of DSLSDI is interrupted for any reason in one of the two stages, the same job with the same sequential input data set and the same intermediate function must be started again. No messages are duplicated, and no messages are lost. DSLSDI can determine how to complete the job:

- If the intermediate queue is empty, normal processing is done as explained earlier.
- If the intermediate queue is not empty, and the last message in this queue is not the restart message: DSLSDI was interrupted in stage 1. Therefore all messages in the queue are deleted, and stage 1 and stage 2 are performed.
- If the intermediate queue is not empty, and the last message in this queue is the restart message: DSLSDI was interrupted in stage 2. Therefore stage 2 is continued, that is, all messages in the queue are routed, deleted, and finally the restart message is deleted. No messages are read from the sequential input file.

SDS Input Program DSLSDIR

DSLSDIR provides the functionality of DSLSDI plus many features and options. The principle mode of processing is the same.

Program flow:

1. Print the header.
2. Read and check the specified runtime parameters.
3. Under MVS, ENQ the intermediate queue.

4. If using the function PUT, ROUTB, or ROUTD, read the intermediate queue and check for a restart situation:
   - If the intermediate queue is empty, perform stage 1 and stage 2.
   - If the intermediate queue is not empty:
     - If the last message is not the restart message, DSLSDIR was interrupted in stage 1: delete all messages in the intermediate queue, perform stage 1 and stage 2.
     - If the last message is the restart message, DSLSDIR was interrupted in stage 2: remember the QSN of the restart message, perform stage 2 only, delete the restart message.
5. Stage 1
   a. Read the input data set with EXECIO.
   b. Put the messages to the intermediate queue (LAZY on):
      - Loop over the records read with EXECIO.
      - If selection criteria were specified, check whether the message matches the specified MSGDST, MSGNET, or SWBHLT.
      - Use the API function WRIT to write a new MSGTRACE entry.
      - PUT the message to the intermediate queue.
   c. If using the function PUT, ROUTB, or ROUTD, write the restart message (LAZY off).
6. Stage 2
   a. Loop through the intermediate queue and PUTB (PUT), ROUB (ROUTB), or ROUN (ROUTD) all queue elements but the restart message to the target queue or queues (LAZY on).
   b. Delete the restart message in the intermediate queue.
   c. Under MVS, DEQ the intermediate queue.

   Note: There is no stage-2 processing for the CHECK function.
7. Print the report.

SDS Load Program DSLSDLR

DSLSDLR loads the messages unloaded with the DSLSDUR program back to their MERVA queues preserving the queue name, QSN, key values, and the *write back* indicator.

It is checked that on the MERVA ESA queue no queue elements with the same or a higher QSN exist.

Program flow:
1. Print the header.
2. Read and check the specified runtime parameters.
3. Determine with the **du** command whether users are currently active in MERVA. Depending on the value of the runtime parameter ACTIVEUSERS, this might not be allowed.
4. Check that the input data set is FB1024.
5. Loop over the input records. As the input data set may be very large, DSLSDLR cannot read all input records with one EXECIO.
   a. If it is an ‘A-’ line, check whether it is valid and contains a wanted queue name.

   b. Assemble the lines belonging to one 'A-' line into variable *mymsg*.
   c. Input the message with API function PUTR.
6. Close input data set.
7. Print report.

SDS Output Program DSLSDO

DSLSDO reads a batch of messages from a MERVA ESA queue and writes them to a sequential data set.

The operational unit for DSLSDO is the batch of messages found in the queue for which DSLSDO was started (input queue). If necessary, DSLSDO performs a restart on this basis. The input queue must not be processed by any other program as long as DSLSDO is running. Never start DSLSDO for a queue that was processed by the hardcopy print program DSLHCP.

The input queue can still be the target of a routing operation while DSLSDO is running.

Processing takes place in two stages:

Stage 1

The messages are read from the input queue and written to the sequential data set.

To determine the batch of messages, DSLSDO stores a specific restart message at the end of the input queue. Messages that enter the queue after the restart message are ignored in this run of DSLSDO, and after a system breakdown, DSLSDO can determine where the end of this batch was.

It is possible to run several DSLSDO programs simultaneously if each of them has its own reserved input queue.

For each message, the following steps are performed:

1. Get the message from the input queue.
2. Call the user exit 22 (DSLMU022) for additional processing of the message, or to skip the message. Refer to the *MERVA for ESA Customization Guide* for more information on this user exit.
3. If the function of the input queue specifies CHECK=YES, perform formal checking of the message. For a message in queue format no checking is performed.

   Handle an incorrect message according to the specification of the disposition parameter (ACCEPT, ROUTE, or CANCEL). Routed messages are immediately deleted from the input queue.

   Note: Even with disposition ACCEPT, a message that cannot be formatted for the sequential file is routed.
4. Format a correct or accepted message as specified by the PARM parameter of the JCL EXEC statement.
5. Write a correct or accepted message to the sequential output data set.

   Usually there is one message in one logical record. In addition, MERVA ESA supports segmentation of messages in which case one message may span more than one logical record of the output data set. Whether segmentation of messages is used or not is determined by a processing parameter in the JCL EXEC statement. When it is used it applies to all records of the output data set. In this case the message length can be larger than 32KB.

The first byte in each record is the segment indicator. This byte may contain one of the following four values:

0 (X'F0')	Only segment of a message
1 (X'F1')	First segment of a multi-segment message
2 (X'F2')	Last segment of a multi-segment message
3 (X'F3')	Middle segment of a multi-segment message

A message is divided into one or more logical records internally. Each of the records together with the appropriate segment indicator byte is written to the output data set.

After all messages up to the restart message have been processed, the output data set is closed.

Stage 2

The messages in the input queue are deleted. The following steps are performed:

- Flag the first message in the queue to indicate the start of the deletion.
- Delete all messages from the second one up to the restart message.
- Delete the first message and the restart message.

If the processing of DSLSDO is interrupted for any reason in one of the two stages, the same job with the same sequential output data set and the same input function must be started again. No messages are duplicated, no messages are lost. DSLSDO can determine how to complete the job:

- If the first message in the input queue does not have the flag on that indicates that the deletion of messages was started: DSLSDO was interrupted during stage 1. Therefore a new restart message is written to the end of the input queue, and stage 1 is processed. All messages that have been routed to the input queue between the previous and the actual run are included in the output data set. If an earlier restart message is found, it is immediately deleted. A new output data set is created.
- If the first message in the input queue has the flag on that indicates that the deletion of messages was started: DSLSDO was interrupted during stage 2. Therefore all messages in the input queue up to the restart message are deleted, the output data set is not changed as it was already complete.

SDS Output Program DSLSDOR

DSLSDOR provides the functionality of DSLSDO plus other features and options. The principle mode of processing is the same.

Program flow:

1. Print the header.
2. Read and check the specified runtime parameters.
3. Build a list of the queue elements of the input queue that match the specified selection criteria (from QSN, to QSN, from KEY1, to KEY1, from KEY2, and to KEY2).

4. If using the DELETE function, check for restart situation. (This step is only necessary for the DELETE function; the functions CHECK and KEEP do not modify the MERVA ESA queue data set in any way.)

   The first message in the input queue that matches the specified selection criteria has a flag, called the *write back indicator*, that indicates whether the deletion of messages was started. If the write back indicator indicates that the deletion of messages:

   - *Was* started, then DSLSDOR was interrupted during stage 1, and a new restart message is written to the end of the input queue, and stage 1 is processed. All messages that have been routed to the input queue between the previous and the actual run are included in the output data set. If an earlier restart message is found, it will be deleted. A new output data set is created.
   - *Was not* started, then DSLSDOR was interrupted during stage 2, and all (matching) messages in the input queue up to the restart message are deleted, the output data set is not changed as it was already complete.
5. Stage 1
   a. If using the DELETE function, write the restart message.
   b. Read the input queue, that is, loop over the queue elements found in step 3, and select those that match the specified selection criteria (from UMR, to UMR, MSGDST, MSGNET, SWBHLT).
   c. Write the messages with EXECIO to the output data sets.
6. Stage 2

   If using the DELETE function, delete all matching messages in the input queue:
   a. A flag (the *write back* indicator) is set for the first matching message in the queue to indicate that stage 2 has started.
   b. Delete the subsequent matching messages up to the restart message.
   c. Delete the first matching message and the restart message.
7. Print the report.

SDS Unload Program DSLSDUR

DSLSDUR unloads the messages of all or specified MERVA ESA queues to a sequential data set preserving the queue name, QSN, key values, and the *write back* indicator. The messages can be reloaded with the DSLSDLR utility.

Program flow:
1. Print the header.
2. Read and check the specified runtime parameters.
3. Use the **du** command to determine whether users are currently active in MERVA. Depending on the value of the runtime parameter ACTIVEUSERS, this might not be allowed.
4. Get all function names and store them in stem variable *qn*.
5. Check that the output data set is FB1024.
6. Loop over the queue names in stem variable *qn*.
   a. Write the line *** **Queue *qname*** to the output data set, where ***qname*** is the name of the queue.
   b. Loop over the messages in the queue.
      1) Read the next message with API function GETU.
      2) Map the message with API function MSGG to external format.

    3) Write the 'A-' line to the output data set.
    4) Split the message into chunks of 1023 characters.
    5) Prefix each chunk with the segment character (to a total line length of 1024 characters).
    6) Write the message segments to the output data set.
7. Close the output data set.
8. Print the report.

System Printer Program DSLSDY

DSLSDY reads a batch of messages from a MERVA ESA queue and prints them on a SYSOUT printer.

Messages are printed according to the specifications of the following parameters of the function table entry of the input queue:

- MSGID. This parameter determines which parts of a message are to be printed, if available:
  - The MERVA Link control information
  - The telex header and other telex information
  - The SWIFT message
  - Other information
- PRFORM. This parameter determines the format of the printing.

The format of the printing can be modified by the specification of the PARM parameter of the JCL EXEC statement.

DSLSDY does not perform address expansion even if specified in the function table entry of the input queue. If expanded addresses are to be printed, the expansion must be done before routing the messages to the printer queue, for example, using the program DSLCXT.

The following steps are performed for each message:

- Get the message from the input queue.
- Call the user exit 21 (DSLMU021) for additional processing of the message or to skip the message. Refer to the *MERVA for ESA Customization Guide* for more information on this user exit.
- Check the message for formal correctness if the function of the input queue specifies CHECK=YES.
- Format the message for the printer as specified by message ID, PRFORM and PARM parameter of the JCL EXEC statement.
- Write the message pages, line-by-line, to the printer.
- Delete the message in the input queue if the function of the input queue specifies KEEPMSG=NO. Do not delete the message if KEEPMSG=YES is specified. If the messages are not deleted, DSLSDY prints all messages again in a subsequent run, and the ones that have been routed to the input function in the meantime.

System Printer Program DSLSDYR

DSLSDYR provides the functionality of DSLSDY plus many features and options. The principle mode of processing is the same. The print lines are written to a sequential file.

Program flow:

1. Print the header.
2. Read and check the specified runtime parameters.
3. Build a list with the queue elements in the input queue that match the specified selection criteria (from QSN, to QSN, from KEY1, to KEY1, from KEY2, and to KEY2).
4. Use the API function PRTI to initialize the printing environment.
5. Customize the printing environment by setting the appropriate TUCB fields.
6. Loop over the queue elements found in step 3:
   a. Read the queue element with API function GET.
   b. Check whether the queue element matches the specified selection criteria (from UMR, to UMR, MSGDST, MSGNET, SWBHLT), if any.
   c. Use the API function WRIT to write the contents of field DSLSDYNO (the running number) to the TOF.
   d. If required, print the separator page to the output data set.
   e. Loop with API function PRTL over the print lines of one message, save the lines to be printed in the stem variable **outrec**.
   f. Write the lines with EXECIO to the output data set.
   g. If using the DELETE function, delete the queue element.
7. Use the API function PRTT to terminate the printing environment.
8. Close the output data set.
9. Print the report.

Converting Messages to Other Formats

MERVA ESA provides programs to:

- Convert EDIFACT FINPAY messages into SWIFT MT121 messages (using envelope and de-compose), and vice versa.
- Convert EDIFACT messages (of any type, including FINPAY) into SWIFT MT105 or MT106 messages (using envelope and de-compose), and vice versa (using de-envelope and compose).

The following descriptions assume that you are familiar with the S.W.I.F.T. EDI Handbook. The term *EDIFACT message* refers here to to a SWIFT EDIFACT messages (for example FINPAY) that can be exchanged by banks using the S.W.I.F.T. EDI service.

Converting EDIFACT FINPAY Messages into MT121 Messages, and Vice Versa

Use the batch utility DSLSDIR to convert EDIFACT FINPAY messages to SWIFT MT121 messages. It can process input FINPAY messages, for example those created by a bank application, and append the necessary SWIFT headers and tags to make them into SWIFT MT121 input messages.

Use the batch utility DSLSDOR to convert SWIFT MT121 messages to EDIFACT FINPAY messages. Only the bank message part is written.

See the *MERVA for ESA Operations Guide* for details.

Converting EDIFACT Messages into SWIFT MT105 or MT106 Messages

The conversion of an EDIFACT message into a sequence of MTs 105 or 106 is performed as followings:

- Create the SWIFT EDIFACT envelope MT 105 or 106.
- Split the EDIFACT message into parts that fit into the field 77F of MT 105 or field 77G of MT 106.

The conversion of EDIFACT messages into the message types 105 or 106 is possible using:

- Customer-written batch programs
- Customer-written transaction programs

The conversion uses the programs DSLCES1 and DSLCES2. Calling DSLCES1 or DSLCES2 is possible from a DSLAPI environment.

The sample transaction program DSLCESTR shows how a customer-written program can invoke DSLCES1.

In customer-written programs, the conversion can be combined with the invocation of a program that processes EDIFACT messages, for example, IBM Data Interchange.

The information for the fields of the message types 105 or 106 is obtained from the EDIFACT message. This requires that you use the formats that are described in the S.W.I.F.T. EDI Handbook. User exit DSLMU242 is called to modify the fields, for example, to determine the message type 105 or 106 depending on the receiver, or to set the transaction reference number. The content of field 27 (sequence of total) is created by the conversion program (see below) depending on the number of MT105s or MT106s created.

The EDIFACT character set A is supported by MERVA ESA. Note that when using Telex Link for Message Types 105 and 106, not all characters of EDIFACT character set A are allowed in a Telex transmission.

The following programs are used for the conversion:

- The conversion program DSLCES1. DSLCES1 invokes the MERVA ESA services necessary for the conversion, and the program DSLCES2.
- The conversion program DSLCES2. DSLCES2 creates the MT105s or MT106s in SWIFT format from the EDIFACT message.
- The user exit DSLMU242 for retrieval or modification of the fields for the message types 105 or 106.
- The sample transaction program DSLCESTR that uses DSLCES1 for the conversion.

DSLCES1 uses an intermediate queue that must not be used by other programs (for example, DSLSDI) and vice versa. When converting EDIFACT/FINPAY messages into SWIFT MT105s or MT106s, the intermediate queue must be empty.

Conversion Program DSLCES1

DSLCES1 needs a MERVA ESA environment that is initialized and terminated by the caller. The caller can be:

- The sample transaction DSLCESTR

- Another customer-written program

DSLCES1 is called with one EDIFACT message in a buffer in main storage and the name of the intermediate function. The intermediate function is used by DSLCES1 to perform a restart similar to the one performed by DSLSDI if necessary.

DSLCES1 performs the following steps:

- Use DSLQMG TYPE=GETLAST to find out the processing status:
  - If the intermediate queue is empty, go to normal processing.
  - If the message retrieved is the restart message of DSLCES1, processing was interrupted during routing of the previous message. Routing is resumed. When complete, there are two cases:
    - The new EDIFACT message *is* identical with the one in the intermediate queue. In this case, processing is finished after routing is complete.
    - The new EDIFACT message is *not* identical with the one in the intermediate queue. In this case, the new EDIFACT message is processed after routing is complete.
  - If the message retrieved is *not* the restart message of DSLCES1, processing was interrupted during creation of the MT105s or MT106s from an EDIFACT message. There are also two cases:
    - The new EDIFACT message *is* identical with the one in the intermediate queue. In this case, the processing of this message can be resumed and completed.
    - The new EDIFACT message is *not* identical with the one in the intermediate queue. In this case, DSLCES1 informs the calling program by an appropriate return code. The calling program must decide what to do with these messages, for example, delete them or route them to another queue. When the intermediate queue is empty, DSLCES1 can be called to process the new EDIFACT message.
- Call DSLCES2 to start conversion of the EDIFACT message. DSLCES2 will give a return and reason code that indicates if the EDIFACT message can be processed. If not, the following steps are skipped. If so, the first MT105 or MT106 is returned.
- Call the user exit DSLMU240 to allow inspection or modification of the MT 105 or 106.
- Store the MT105 or MT106 in the intermediate queue.
- If it was not the last MT105 or MT106, call DSLCES2 again and process all MT105s or MT106s until all parts are stored in the intermediate queue.
- Finally store the DSLCES2 restart message in the intermediate queue.
- Route the messages from the intermediate queue to user-defined queues.
- Finally delete the restart message.
- Return to the caller with appropriate return information.

Conversion Program DSLCES2

The caller of DSLCES2 can be:

- DSLCES1
- A customer-written program

The calling program must provide the working storage of DSLAPI and must have initialized it.

DSLCES2 is called with an EDIFACT message in a buffer in main storage. When called the first time for this EDIFACT message, DSLCES2 performs the following steps:

- Call the user exit DSLMU242 that retrieves the data for the fields of the MT105s or MT106s from the EDIFACT message.

  The relation between fields used in a message type 105 or 106 and a FINPAY message is defined in the S.W.I.F.T. EDI Handbook. The EDIFACT message types FINPAY and REMADV are supported by the user exit for automatic conversion. When other EDIFACT messages are to be converted it might be necessary to modify this user exit.

  The Sub-Message Type field 12 is filled according to the definition given in the S.W.I.F.T. EDI Handbook. The contents of EDIFACT Field BGM is used to fill the Related Reference field 21 as well as Transaction Reference Number field 20 in the SWIFT message. The standard user exit fills the same value for all resulting parts of an EDIFACT message (SWIFT MTs 105 or 106) into the Transaction Reference Number field 20. It is not required to have the same value in field 20 of the SWIFT MTs 105 and 106. The Related Reference Number field 21 must be the same for each message in a series.

  Note: The Related Reference Number field 21 must be unique for each EDIFACT message. This allows for deleting duplicates when converting the SWIFT messages to the EDIFACT message.
- Check the EDIFACT message for the level-A character set. If an error is found, all MT105s or MT106s are marked in error in the MSGTRACE field to allow error routing of all parts, no matter in which part the error really is.
- The EDIFACT message cannot be processed if it is too long. This is indicated by a return and reason code to the caller.
- If everything is OK, create the first MT105 or MT106 and indicate to the caller if there are more parts or if there was only one.

  The decision whether to convert into SWIFT message types 105 or 106 is controlled by the user exit DSLMU242. In the delivered sample exit the decision is made according to the BIC Codes in field UNB of the FINPAY/EDIFACT message and a table in DSLMU242.

  A message type 106 is created if the BIC codes for home address and correspondent address in the message are found in a table coded in the user exit. Message type 105 is created if the BIC codes are not found in this table. This table must be modified according to the installation's requirements.

If called for the second or other part, create this part and give appropriate return information.

Sample Transaction Program DSLCESTR

DSLCESTR is a CICS or IMS transaction that invokes DSLCES1 to converr an EDIFACT message into an MT105 or MT106. DSLCESTR initializes and terminates the interfaces to MERVA ESA using DSLAPI. DSLCESTR performs the following steps:

1. Get an EDIFACT message from, for example, a MERVA ESA queue, a data set, or IBM Data Interchange. The sample delivered with MERVA ESA is set up to retrieve a message from a queue.
2. Call DSLCES1 to check the restart conditions and to convert the EDIFACT message into an MT105 or MT106.
3. Handle the return information from DSLCES1.

The source code of DSLCESTR can be found in the MERVA ESA source library under the names:

- DSLCESTI for IMS
- DSLCESTC for CICS/MVS
- DSLCESTV for CICS/VSE

Converting SWIFT MT105 or MT106 Messages into EDIFACT Messages

The conversion of a sequence of MTs 105 or 106 into an EDIFACT message is performed in the following way:

- Get the data from field 77
- If several MTs 105 or 106 contain parts of one EDIFACT message, concatenate the field 77 data in the correct order

The conversion of MT105s or MT106s into an EDIFACT message requires that all parts have been received from the SWIFT network and are available in one MERVA ESA queue.

MERVA ESA cannot process the MT105s or MT106s that never become complete and therefore remain in the queue. The particular installation must decide what to do with incomplete messages, for example, to process them manually or to delete them. When converting SWIFT MTs 105 and 106 into EDIFACT/FINPAY the input queue must contain only these message types.

The conversion of MT105s or MT106s into EDIFACT messages is possible using:

- Customer-written batch programs
- Customer-written transaction programs

The conversion uses the program DSLCSE1, which can be called by a customer-written program. Calling DSLCSE1 is possible from a DSLAPI environment. The sample transaction program DSLCSETR shows how to invoke DSLCSE1.

In customer-written programs, the invocation of the conversion can be combined with the invocation of a program that processes EDIFACT messages, for example, IBM Data Interchange.

The following programs are used for the conversion:

- The conversion program DSLCSE1 that invokes the MERVA ESA services necessary for the conversion
- The sample transaction program DSLCSETR that uses DSLCSE1 for the conversion

The contents of the SWIFT message fields are lost after the conversion. For that reason, the user exit DSLMU241 is called for each MT105 or MT106 to save this information if necessary, for example, in a database.

The input queue used by DSLCSE1 must not be used by other programs (for example, DSLSDO) and vice versa.

Conversion Program DSLCSE1

DSLCSE1 needs a MERVA ESA environment that is initialized and terminated by the caller. The caller can be:

- The sample transaction DSLCSETR
- Another customer-written program

DSLCSE1 is called with one MT105 or MT106 in the MERVA ESA queue buffer and the name of the input function. This queue must be defined with the field 21 (related reference) as key 1 and field 27 (sequence of total) as key 2.

DSLCSE1 performs two main requests:

1. Status check and conversion
2. Deleting, moving or routing the MT105 or MT106 of the input queue

When processing status check and conversion, DSLCSE1 performs the following steps:

- Use DSLQMG TYPE=LIST to find out if all parts of the EDIFACT message are in the queue using field 21 (related reference) as list argument for key 1. It is necessary that EDIFACT messages have different fields 21 if they come from the same sender. Field 27 (sequence of total) is returned in key 2 and allows to check completeness. When processing the individual parts, the sender is verified to avoid mixture of parts that have the same field 27 but come from different senders. Messages from other senders are removed from the list immediately. The following conditions can occur:
  - If not all parts are available, return to the caller with an appropriate return and reason code.
  - If all parts are available, start the processing loop. Some parts may be available more than once, they are considered only once.
- Processing loop:
  - Get the MT105s or MT106s in the order of field 27 using the queue sequence number (QSN) from the list.
  - Call the user exit DSLMU241 to inspect the MT105s or MT106s.
  - Get the data areas of field 77F or 77G and create the EDIFACT message in a buffer.
- If the EDIFACT message is complete, indicate completion and return to the caller.

DSLCSETR and customer-written programs call DSLCSE1 for the deletion of the MT105s or MT106s in the input queue after they have processed the EDIFACT message. This deletion uses the same queue list. After deletion, the processing of one EDIFACT message is complete.

Sample Transaction Program DSLCSETR

DSLCSETR is a CICS or IMS transaction that invokes DSLCSE1 for conversion of MT105s or MT106s into an EDIFACT message. DSLCSETR initializes and terminates the interfaces to MERVA ESA using DSLAPI.

The source code of DSLCSETR can be found in the MERVA ESA source library under the names:

- DSLCSETI for IMS
- DSLCSETC for CICS/MVS
- DSLCSETV for CICS/VSE

DSLCSETR processing is different depending on how it was started:

- When started by an **sf** command, processing starts with the first message in the queue and processes the whole queue.
- When started because a message was stored in the input queue, processing starts by retrieving this message using the queue sequence number (QSN) in the terminal and user control block (TUCB).

After having retrieved a message from the input queue, DSLCSETR performs the following steps:

1. Call DSLCSE1 for status check and conversion. Further processing depends on the return information of DSLCSE1:
   - If not all parts of the EDIFACT message are available, the conversion can not be done.
   - If the EDIFACT message has been created, process it further, for example, give it to IBM Data Interchange. The sample supplied contains the code for storing in a queue.
2. If the EDIFACT message has been processed successfully, call DSLCSE1 for deletion of the MT105 or MT106.

Continue with the next message in the input queue until the end of the queue is reached.

User-Written Application Programs

The purpose of user-written application programs is determined by your installation. The *MERVA for ESA Customization Guide* and the *MERVA for ESA Application Programming Interface Guide* provide information on how to use MERVA ESA services in user-written application programs.

Chapter 12. Communication Links

This chapter describes the communication components provided by MERVA ESA:
- SWIFT Link
- Telex Link
- MERVA Link
- MERVA-MQI Attachment

SWIFT Link

SWIFT Link enables communication with the SWIFT network; it supports X.25 connections 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. Refer to the *MERVA Extended Connectivity Installation and User's Guide* for more information about this product. SWIFT Link performs all actions required by the SWIFT network. It can be adapted to the operational and functional requirements of your business. You can use up to 30 lines to the SWIFT network.

If multiple logical terminals are used in an MVS installation and high throughput is required, you can customize MERVA ESA so that it uses multiple independent SWIFT Link servers running as parallel tasks.

SWIFT Link calls messages sent to the SWIFT network *input messages* because they are input to the SWIFT network, and they contain an input sequence number (ISN). The acknowledgment messages for input messages are called *ISN ACKs* because they are received for a specific input sequence number (ISN). In SWIFT terminology, these are system acknowledgments.

SWIFT Link calls messages received from the SWIFT network *output messages* because they are output from the SWIFT network, and they contain an output sequence number (OSN). In SWIFT Link, the acknowledgment messages for output messages are called *OSN ACKs* because they are sent for a specific output sequence number (OSN). In SWIFT terminology, these are user acknowledgments.

The formats of all SWIFT messages are described in the *S.W.I.F.T. User Handbook*.

Overview of the SWIFT Link

The architecture of the SWIFT network uses layers that are derived from the OSI reference model of the International Organization for Standardization (ISO). This architecture is described in the *S.W.I.F.T. User Handbook*. SWIFT Link reflects the layered structure.

SWIFT Link supports the two parts of the SWIFT User Security Enhancements (USE):
1. Secure Login/Select (SLS)
2. Bilateral Key Exchange (BKE)

Many functions of the SWIFT USE are implemented running on a workstation, called the *USE workstation* in this book. For the communication between MERVA ESA and the USE workstation, one of the following can be used:

- MERVA Link with VTAM LU 6.2 (advanced program-to-program communications, often abbreviated to APPC). Refer to “The MERVA Link” on page 134 for more information about the MERVA Link.
- MERVA-MQI Attachment using MQSeries. Refer to “MERVA-MQI Attachment” on page 152 for more information about the MERVA-MQI Attachment.

An overview of SWIFT Link is shown in Figure 14.

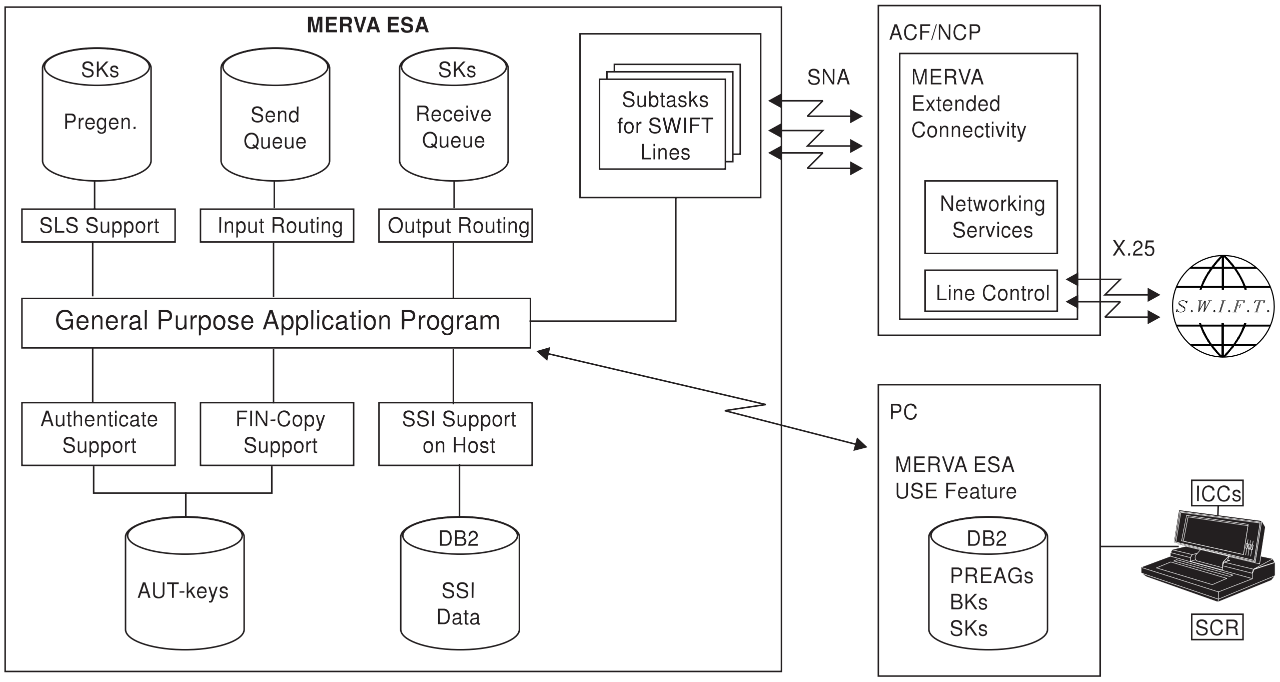


Figure 14. SWIFT Link Structure

The main program of the SWIFT Link is the general purpose application program DWSDGPA. It prepares all messages for sending to the SWIFT network, and processes all received messages. Note that DWSDGPA can exist several times in a single MERVA ESA installation.

DWSDGPA is controlled by the MERVA ESA nucleus program table DSLNPTT. One or more servers with the DWSDGPA program can be specified. Each of these servers uses a different descriptive name (DESC parameter). The descriptive name of the main SWIFT Link server is SWIFTII. The parallel servers must be defined as SWIFTII*x*, where *x* represents the letter A, B, or C. The additional SWIFT Link servers must be defined in the nucleus server table (DSLNSVT) to run as subtasks.

As the parallel servers share the centralized resources of MERVA ESA there is limit of parallelization; it is not useful to have more than three additional SWIFT Link servers defined.

You can use from 1 to 30 lines to connect to the SWIFT network. For each line, a subtask is attached that runs under the control of the operating system MVS or VSE. The protocol used is X.25.

The lines can be shared by parallel SWIFT Link servers. Each of these lines is a VTAM connection; that is, the lines as seen by SWIFT Link are logical lines that finally end up as switched virtual circuits (SVCs) on a physical X.25 line. The physical X.25 line to the SWIFT network is controlled by MERVA Extended Connectivity. It is possible to run multiple SVCs on one physical X.25 line. Thus, the parallel SWIFT Link servers can share a single physical X.25 line. For more information, refer to “Connection to the SWIFT X.25 Network” on page 125.

The main program of the subtask is the event control program DWSNAEVV.

DWSNAEVV 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 (DWSTxxxx)
- Link layer (DWSNLNK)

DWSNLNK switches control to the network link program for the X.25 connection to the SWIFT network.

DWSNLNKV 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. For more information, refer to “Connection to the SWIFT X.25 Network” on page 125.

DWSAUTP, the authentication support, is called by DWSDGPA to authenticate SWIFT messages when required. DWSPREM, the SWIFT PREMIUM support, is called by DWSAUTP when the PREMIUM support is installed. DWSFCPY, the SWIFT FIN-Copy support, is called by DWSAUTP when the FIN-Copy support is installed. For more details refer to the *MERVA for ESA Customization Guide*.

Authenticator keys enter the authenticator file in one of the following ways:
- During authenticator key file online maintenance using the program DWSEAUT
- Off-line using the authenticator key file utility DWSAUTLD
- Receiving an update from the USE workstation (for example, after completion of a bilateral key exchange, BKE) that is processed by the transaction DWSAUTT

Most SWIFT Link programs require customizing data defined in DWSPRM.

SWIFT Link appears to the SWIFT network as a computer-based terminal (CBT).

The connection to the SWIFT network is controlled by the following components of SWIFT Link:
- Operator commands start, stop, and monitor the connection with the SWIFT network. These commands are defined in the operator command table DSLNCMT, and they are described in detail in the *MERVA for ESA Operations Guide*.
- The general purpose application program DWSDGPA processes the protocol of a SWIFT network connection on the application level.
- The event control program DWSNAEVV which processes the SWIFT layer protocols in the line subtasks. There is a special communication between DWSDGPA and the line subtasks using event control blocks (ECBs) and the WAIT/POST facilities of the operating system.

Logical Terminal Table DWSLTT

SWIFT Link can handle several different SWIFT master logical terminals (SWIFT addresses). Therefore several financial institutions can share one MERVA ESA installation.

Each SWIFT Link server has its own logical terminal table. The name of the table is customized with the PARM parameter of the DSLNPT definition statement. It is recommended that you use the naming convention DWSLTT*x*, where *x* is the letter A, B, or C, and corresponds to the descriptive name SWIFTII*x* of the parallel SWIFT Link server.

Master logical terminals are used to log in to the SWIFT network. Login is used for the general purpose application (GPA, also called the application control, or APC) of this master logical terminal. Each master logical terminal has its own range of session numbers, login sequence numbers (LSN), input sequence numbers (ISN), and output sequence numbers (OSN).

For each master logical terminal, a financial application (FIN) can be defined. Each FIN application has its own range of session numbers, select sequence numbers (SSN), input sequence numbers (ISN), and output sequence numbers (OSN).

One or more synonym logical terminals can be defined for each master logical terminal. Synonym logical terminals share the session number, LSN, ISN, and OSN range of the GPA with their master logical terminal, and they share the session number, SSN, ISN, and OSN of the FIN application of their master logical terminal. When a master logical terminal is logged in or has selected the financial application, its synonyms share both the LOGIN status of the APC and the OPEN status of the FIN.

All master logical terminals, their FIN applications, and their synonyms are defined in the logical terminal table DWSLTT using the DWSLT macro (see the *MERVA for ESA Macro Reference* for details). The name of the logical terminal table can be changed in the customizing parameters (DWSPRM).

The following attributes can be defined in DWSLTT:

- The name of the master or synonym logical terminal.
- The type of entry: master, synonym, or financial application.
- The number of the line a master logical terminal uses for the communication with the SWIFT network. The number of the line determines the name of the line module to be used. The master logical terminals of one SWIFT Link can share one line to the SWIFT network, or they can use several lines in any combination. The synonyms and the financial applications always use the same line as the master.
- The technology that is used for login and select: paper tables or SLS within the SWIFT USE.
- The names of the routing tables used for routing input and output messages. The master logical terminal and the financial applications can use different routing tables. The synonyms share the routing tables of the master or FIN.
- The names of up to four ready queues. The master logical terminal and the financial applications must use different ready queues. The synonyms share the ready queues of the master or FIN.
- The name of the session key queue if SLS and pregenerating of session keys is used.

- The default window for the login of the master logical terminal or for the select of the FIN application.
- The default delivery subset mnemonics for the select of the FIN application.

Some information of DWSLTT is stored in a MERVA ESA queue between a SWIFT Link termination and the next startup. The name of this queue is defined in DWSPRM.

General Purpose Application Program DWSDGPA

The general purpose application program DWSDGPA controls the communication with the SWIFT network. DWSDGPA controls login, logout, select, quit, abort, sending, and receiving of SWIFT messages and acknowledgment messages.

DWSDGPA is defined in the MERVA ESA nucleus program table DSLNPTT with the descriptive name SWIFTII. DWSDGPA can be started automatically during the MERVA ESA startup, or it can be started and stopped by an authorized MERVA ESA operator using the MERVA ESA **start** and **stop** commands.

When customizing the DSLNPT macro of DWSDGPA in DSLNPTT, only the parameters DESC, AUTO, PRTY, STRT, STOP, or PARM can be changed.

DWSDGPA is composed of several subprograms. These programs act like one program, and referencing DWSDGPA always means all of them.

DWSDGPA performs the initialization, processing, and termination of the SWIFT Link.

DWSDGPA is a reentrant program and allows the execution of multiple SWIFT Link servers exploiting the parallel processing concept of MERVA ESA.

Connection to the SWIFT X.25 Network

MERVA ESA uses the separate product MERVA Extended Connectivity to connect to the SWIFT X.25 network.

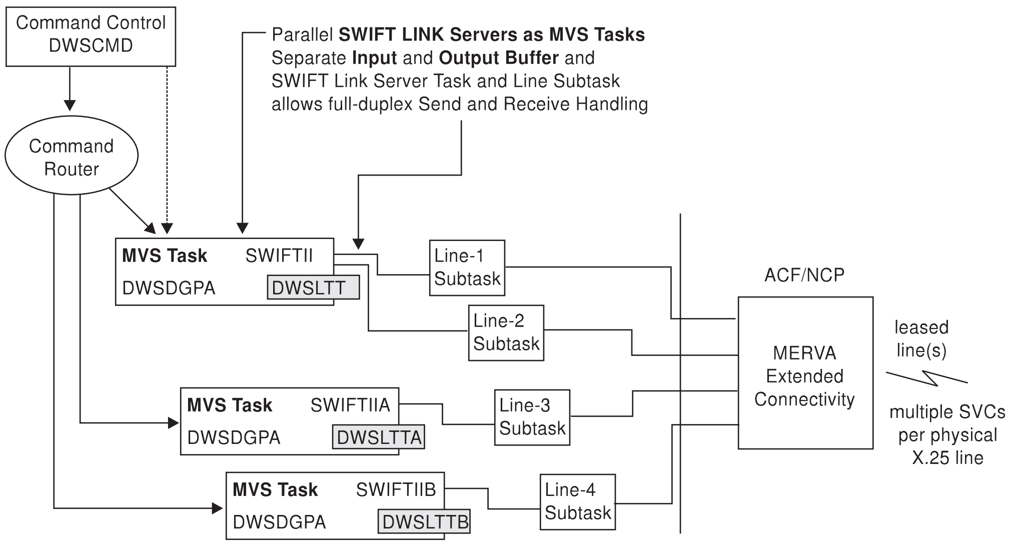

Command Control DWSCMD
Parallel SWIFT LINK Servers as MVS Tasks
Separate Input and Output Buffer and
SWIFT Link Server Task and Line Subtask
allows full-duplex Send and Receive Handling
Command Router
MVS Task SWIFTII
DWSDGPA DWSLTT
Line-1 Subtask
Line-2 Subtask
ACF/NCP
MERVA Extended Connectivity
leased line(s)
multiple SVCs per physical X.25 line
MVS Task SWIFTIIA
DWSDGPA DWSLTTA
Line-3 Subtask
MVS Task SWIFTIIB
DWSDGPA DWSLTTB
Line-4 Subtask

Figure 15. MERVA ESA Connection to the SWIFT X.25 Network

The MERVA ESA SWIFT Link that uses X.25 has two main parts:

- The general purpose application program DWSDGPA runs directly under control of the MERVA ESA nucleus. DWSDGPA retrieves and stores SWIFT messages in the MERVA ESA queues and formats these messages for the SWIFT network. It provides the interface for the operator control of SWIFT Link.
- The event control program DWSNAEVV runs as a subtask in asynchronous mode. It communicates with DWSDGPA using a shared storage area. DWSNAEVV is started on operator request for a specific line number. The LOGIN and the SETLT commands are used to start a line subtask. Up to 30 line subtasks can be started and operate in parallel.

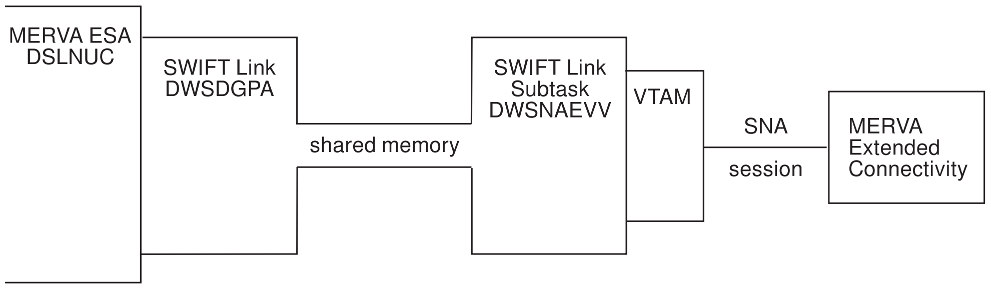


Figure 16. MERVA ESA SWIFT Link Structure

The line subtask processes the layers defined for the SWIFT network. It provides the correct protocol depending on the action requested. For example, the resumption of a suspended switched line is processed automatically when DWSDGPA provides a message for sending to the SWIFT network.

The main program of the subtask is the event control program DWSNAEVV. DWSNAEVV waits for the following events:

- ECB for VTAM event. The network layer program DWSNLNKV is called if this ECB is posted by an asynchronous VTAM exit program. Depending on the event, DWSNLNKV can call the transport layer to further process received data or to provide more data for sending to DWSVTMLC.
- Send ECB. DWSDGPA posts this ECB to indicate that there is a message prepared for sending to the SWIFT network. DWSNAEVV calls the appropriate application program (DWSNLTCK, DWSNAPCK, or DWSNFIN) to process the message.
- Timer ECB of the applications. This ECB is posted by DSLTIMP when a request for association, disassociation, suspension, or resumption is not answered by the SWIFT network in time. The application program (DWSNLTCx, DWSNAPCx, or DWSNFIN) is called to process the time-out.
- Receive ECB. When a message was received from the SWIFT network, the line subtask gives the message to DWSDGPA. DWSDGPA posts the receive ECB after the message has been stored in the MERVA ESA queues.

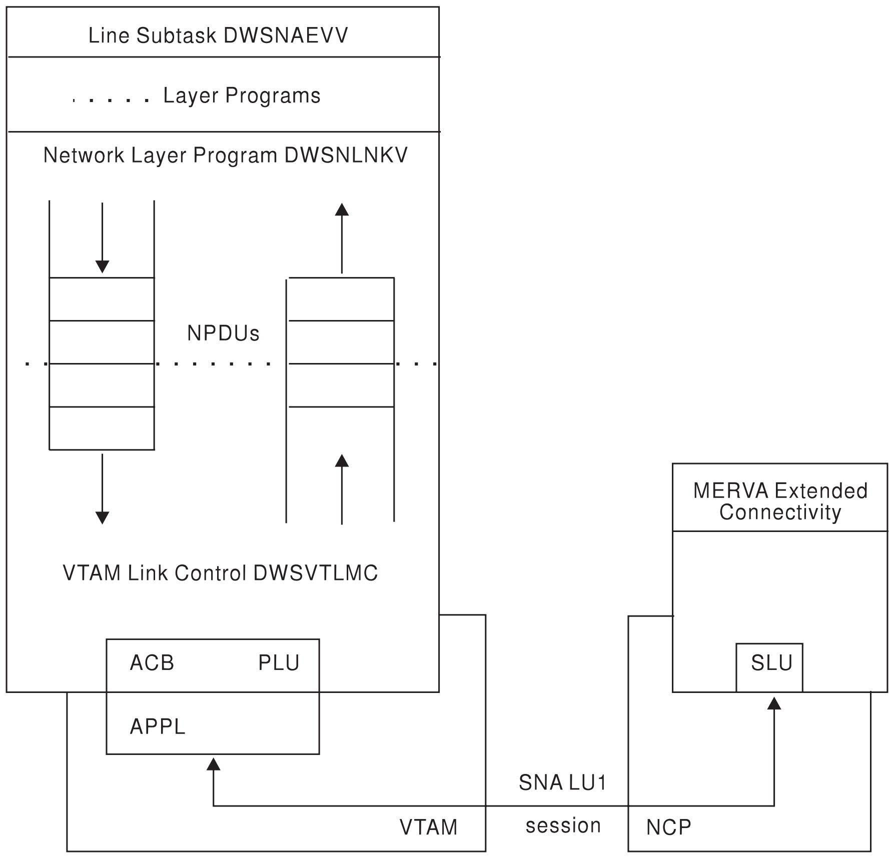


Figure 17. Network Layer of SWIFT Link Subtask

SWIFT Layer Programs

When communicating with the SWIFT network, the SWIFT layer programs perform the following functions as defined by SWIFT:

- The application interface DWSNAIST performs:
  - Associations
  - Suspensions
  - Resumptions
  - Disassociations
  - Data transfers
- The transport layer programs DWSTxxxx:
  - Establish transport connections
  - Release transport connection (disconnect)
  - Transfer data
  - Acknowledge data transfer (transport layer acknowledgments)
- The network layer program DWSNLNKV:
  - Calls DWSVTMLC to initialize an SNA LU1 session to MERVA Extended Connectivity
  - Builds requests to communicate with MERVA Extended Connectivity
  - Calls DWSVTMLC to transfer data to MERVA Extended Connectivity
  - Receives requests from MERVA Extended Connectivity

The data transfer between DWSNLNKV and DWSVTMLC is performed asynchronously by using a queuing mechanism. This mechanism allows the overlapping of data preparation and data transmission operations.

- The VTAM control program DWSVTMLC:
  - Opens and closes the communication path to MERVA Extended Connectivity running on a 37xx controller
  - Communicates with the logical unit controlled by MERVA Extended Connectivity
  - Processes VTAM exits and events asynchronously
  - Transfers data (the data transfer is executed asynchronously to maximize throughput)

MERVA Extended Connectivity

MERVA Extended Connectivity is a licensed program that enables access from MERVA ESA to the SWIFT X.25 network through a 37xx communications controller. MERVA ESA running on the host interacts with MERVA Extended Connectivity at networking services level.

SWIFT Link line subtask acts as the CBT that has a connection to the SWIFT network through an X.25 logical channnel. A switched virtual circuit (SVC) on an X.25 physical connection appears to the subtask as a virtual line on the 37xx controller.

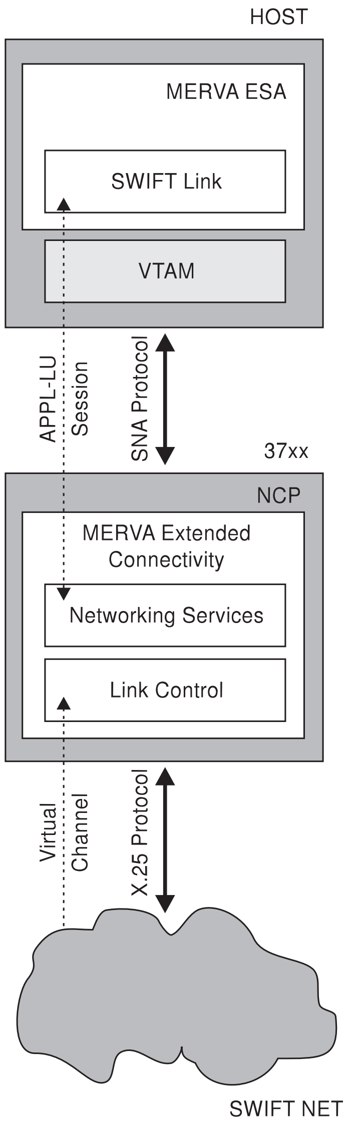


Figure 18. MERVA Extended Connectivity System Configuration

Refer to the *MERVA Extended Connectivity Installation and User's Guide* for further information on the MERVA Extended Connectivity product.

Load Session Keys Program DWSDLSK

The load session keys program DWSDLSK is used together with the SWIFT secure login/select (SLS). It is possible to pregenerate the session keys for login and select on the USE workstation and send them to MERVA ESA via the MERVA Link, or the MERVA-MQI Attachment. The program DWSDLSK receives the session keys and writes them into the session key queues defined for the master logical terminals or the financial applications. The session keys in these queues are encrypted, and they can neither be displayed nor printed.

DWSDLSK is defined in the MERVA ESA nucleus program table DSLNPTT with the descriptive name SWLOADSK. DWSDLSK can be started automatically during the MERVA ESA startup, or it can be started and stopped by an authorized MERVA ESA operator using the MERVA ESA **start** and **stop** commands.

When customizing the DSLNPT macro of DWSDLSK in DSLNPTT, only the parameters DESC, AUTO, PRTY, STRT, or STOP can be changed. DWSDLSK is designed to run with a low priority in DSLNPTT to not compete with the SWIFT Link and other MERVA ESA applications.

DWSDLSK performs an initialization, processing, and a termination.

DWSDLSK Initialization

A MERVA ESA **start** command is used during the MERVA ESA startup or entered by a MERVA ESA operator. Then the DWSDLSK initialization is performed:

- Load the parameter module DWSPRM.
- A DSLQMG TYPE=SET macro is used to provide the address of the ECB of DWSDLSK in the load session keys queue function defined with the LSKQUE parameter of the DWSPARM macro. If this call to DSLQMGT fails, DWSDLSK cannot operate and the initialization does not continue.
- Post the own ECB to get control later in processing.
- Load the logical terminal table DWSLTT.
- Get main storage for buffers.
- Initialize the internal message buffer (TOF) and the message format service.
- Finally, the list of the event control block (ECB) addresses of DWSDLSK is prepared for DSLNUC, and control returns to DSLNUC. This list contains only one ECB address.

If any of these steps fails, a dump of the error is taken, and the DWSDLSK termination is invoked to reset all successful initialization steps. DSLNUC is informed by an appropriate return code that the initialization of DWSDLSK failed.

DWSDLSK Processing

DWSDLSK processing is invoked when the event control block (ECB) of DWSDLSK is posted for one of the following events:

- DWSDLSK initialization is complete.
- A message was routed to the load session keys queue.
- DWSDLSK has written one session key to a session key queue of a master logical terminal or its financial application, and there are more session keys. DWSDLSK interrupts its processing to allow DSLNUC to give control to a program of DSLNPTT with a higher priority. Processing is resumed later.
- DWSDGPA has received a login ACK (LAK) or a select ACK (SAK) and indicates to DWSDLSK that the session key for this LSN or SSN is not needed anymore. DWSDLSK then deletes this session key and the previous two if they exist.

  Older session keys must be deleted manually by an operator.

DWSDLSK Termination

A MERVA ESA **stop** command is entered by a MERVA ESA operator or used during the MERVA ESA termination. Then the DWSDLSK termination is performed:

- Call the message format service for termination.
- Free main storage obtained during initialization.
- Use a DSLQMG TYPE=RESET macro to remove the ECB address from the load session key queue function.
- Delete all modules loaded during initialization.

Errors during the termination are ignored.

Telex Link

Telex Link enables communication with the public telex network using the Telex Plus/2[3] on PC or alternatively the Headoffice Telex on a fault-tolerant system[3].

Telex Link via Fault-Tolerant System

Telex Link via a fault-tolerant system communicates with the Headoffice Telex in the Telex Substation. Telex Link performs all actions required for this communication. The telex substation is connected to the telex network.

Telex Link provides an interface for a test-key calculation program.

Telex Link can be adapted to the operational and functional requirements of your business.

An overview of Telex Link is shown in Figure 19.

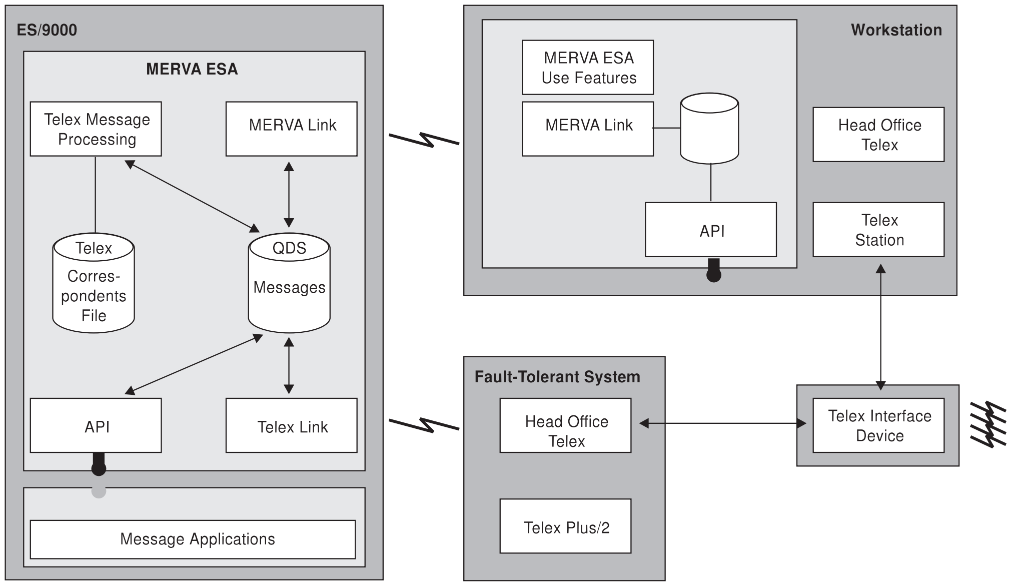


Figure 19. Overview of Telex Link

The main program of Telex Link is the station program. It consists of two parts. One part is the program ENLSTPL, which is linked to DSLNUC and controlled by DSLNPTT. When the Telex Link via a fault-tolerant system is started, ENLSTPL dynamically loads the other part, program ENLSTP. 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 telex interface program in the Telex Substation.

Operator commands can start, stop, and monitor the communication with the telex interface program. These commands are defined in the operator command table DSLNCMT, and they are described in detail in the *MERVA for ESA Operations Guide*.

3. "Telex Plus/2" and "Headoffice Telex" are IBM vendor logo products from *Intercope*. Currently, only these two Telex Interfaces can be used to connect to the Telex network.

Station Program ENLSTP

The station program ENLSTP controls the communication with the telex interface program. ENLSTP controls signon and signoff of the session, and sending and receiving of the telex messages and acknowledgments.

The load program ENLSTPL is defined in the MERVA ESA nucleus program table DSLNPTT with the descriptive name TELEX. ENLSTP can be started automatically during the MERVA ESA startup, or it can be started and stopped by an authorized MERVA ESA operator using the MERVA ESA **start** and **stop** commands.

When customizing the DSLNPT macro of ENLSTP in DSLNPTT, only the parameters DESC, AUTO, STRT, or STOP can be changed.

ENLSTPL loads ENLSTP, which in turn performs the initialization, processing, and termination of Telex Link.

ENLSTP Initialization: A MERVA ESA **start** command is used during the MERVA ESA startup or entered by a MERVA ESA operator. Then the ENLSTP initialization is performed:

- Load the customizing parameter module ENLPRM.
- Get main storage for buffers.
- Initialize the message format service.
- Verify the queue names defined in ENLPRM.
- Route messages contained in the receive and send queues from the last session.
- Perform automatic signon to the telex interface program if specified in ENLPRM.
- Finally, the list of the event control block (ECB) addresses of ENLSTP is prepared for DSLNUC, and control returns to DSLNUC.

If one of these steps fails, an error message is issued, and the ENLSTP termination is invoked to reset all successful initialization steps. DSLNUC is informed by an appropriate return code that the initialization of ENLSTP failed.

ENLSTP Processing: ENLSTP processing is invoked when one of the event control blocks (ECBs) of ENLSTP is posted. The following ECBs are used:

- The receive ECB indicates that a message was received from the telex interface program and was stored in the receive queue by the interface transaction ENLHCF1.
- The queue ECB indicates that a message was written to a ready queue.
- The program ECB indicates that a command of Telex Link was accepted (for example, a **txon** or **txoff** command), or that more work needs to be done for an event.

In all cases, DSLNUC gives control to ENLSTP. Depending on the event, ENLSTP:

- Signs on a session with the telex interface program. A signon command is sent to the telex interface program, and a signon acknowledgment from the telex interface program confirms the signon.
- Signs off a session with the telex interface program. A signoff command is sent to the telex interface program, and a signoff acknowledgment from the telex interface program confirms the signoff. After certain errors, the current session is signed off automatically.
- Prepares messages for sending to the telex interface program from the ready queues. The messages are stored in the send queue for ENLHCF1, which in turn gives them to CICS or IMS for sending.

ENLSTP uses a possible duplicate emitted queue (PDE queue) where one message waits for the logical acknowledgment from the telex interface program and is sent again in case of a transmission failure.

- Receives the logical acknowledgment message that confirms the receipt of the telex message by the telex interface program.

  ENLSTP uses an ACK wait queue where the messages wait for the transmission acknowledgments.
- Receives transmission acknowledgment messages that indicate if sending the telex message via the telex network to the receiver was successful or not.

  The messages are then routed from the ACK wait queue according to a user-defined routing table.
- Receives telex messages that the telex interface program has received from the telex network.

  The messages have been received by the interface transaction ENLHCF1 and stored in the receive queue.

  ENLSTP uses a last-received queue to determine if a message was received twice in case of a transmission failure. Duplicate messages are dropped.

  Received telex messages are routed according to a user-defined routing table.
- Sends an acknowledgment message to the telex interface program for each received telex message.
- Receives status reports from the telex interface program. The status can be displayed with the operator command **txdisp**.
- Writes all traffic with the telex interface program to the MERVA ESA journal if specified in ENLPRM.
- Counts all traffic with the telex interface program in two MERVA ESA message counters, one for incoming telex messages and one for outgoing telex messages.

ENLSTP Termination: A MERVA ESA **stop** command is entered by a MERVA ESA operator or used during the MERVA ESA termination. Then the ENLSTP termination is performed:

- Sign off the session with the telex interface program without waiting for the acknowledgment.
- Call the message format service for termination.
- Free main storage obtained during initialization.
- Delete the customizing parameter module ENLPRM.
- Issue the Telex Link termination message.

Errors during the termination are ignored.

Recovery after Restart

If the communication between Telex Link and the telex interface program fails, Telex Link provides for a recovery after the next successful signon.

Recovery for Messages Sent to the telex interface program: If no logical acknowledgment is received from the telex interface program for the last telex message sent during a session, this message is sent again as the first message after a restart. It includes a reference to the first transmission (the original session and sequence numbers).

The message is sent from the PDE queue, which contains the last unacknowledged message sent to the telex interface program. There can be only one message in the PDE queue.

Recovery of Messages Received from the telex interface program: If the first telex message received from the telex interface program in a session contains the indicator for being a possible duplicate, Telex Link checks whether this telex message has already been received. For this purpose, Telex Link uses the last received queue (LR queue). There can be only one message in this queue:

- If the message in the LR queue matches the received telex message, Telex Link considers the one just received as a duplicate and discards it. An information message is issued to the MERVA ESA operators and an acknowledgment is sent to the telex interface program.
- If the message in the LR queue does not match the message just received, Telex Link considers the one just received as a new message.
- If the LR queue is empty, Telex Link cannot decide whether the message just received is a duplicate. User-defined routing can route this message to a possible duplicate received queue (PDR queue).

  A user must then decide whether it is a duplicate.

The MERVA Link

MERVA Link is comprised of two components:

- **MERVA Link ESA**, which lets you transfer messages and commands between MERVA installations. It is associated with a particular MERVA ESA installation and runs in a CICS or IMS environment.
- **MERVA Link USS**, which lets you route MERVA Link conversations synchronously from an SNA APPC network to a TCP/IP network, and vice versa. It is not associated with a particular MERVA ESA installation, and executes in an OS/390 UNIX System Services (USS) environment.

MERVA Link ESA

MERVA Link ESA enables:

- The communication between customer application programs handling outgoing or incoming messages, and MERVA Link Application Support Programs
- The transfer of any kind of messages between MERVA ESA installations, for example, SWIFT messages, telex messages, and customer-defined messages
- The communication between a MERVA ESA end user and the MERVA ESA Command Facility in a partner MERVA ESA system

MERVA Link ESA lets you set up a global system of cooperating MERVA installations, only one of which has a connection to the SWIFT network. The other installations use MERVA Link ESA to send their SWIFT input messages to the one installation with the SWIFT connection, which in turn uses MERVA Link ESA to return the SWIFT acknowledgments and distribute the SWIFT output messages.

SWIFT acknowledgments can be handled in two ways:

- The entire input message with the acknowledgment data in the MSGACK field is returned to the originating installation. The original message and the acknowledged SWIFT input message are handled as separate messages. The two messages are not correlated by MERVA Link.
- Only the SWIFT acknowledgment data is returned in a MERVA Link acknowledgment message to the originating installation. The original message and the acknowledgment message are correlated, and the SWIFT acknowledgment data is added to the original message. This means, MERVA Link creates an exact copy of the acknowledged input message.

Figure 20 shows an overview of MERVA Link in cooperating MERVA ESA installations.

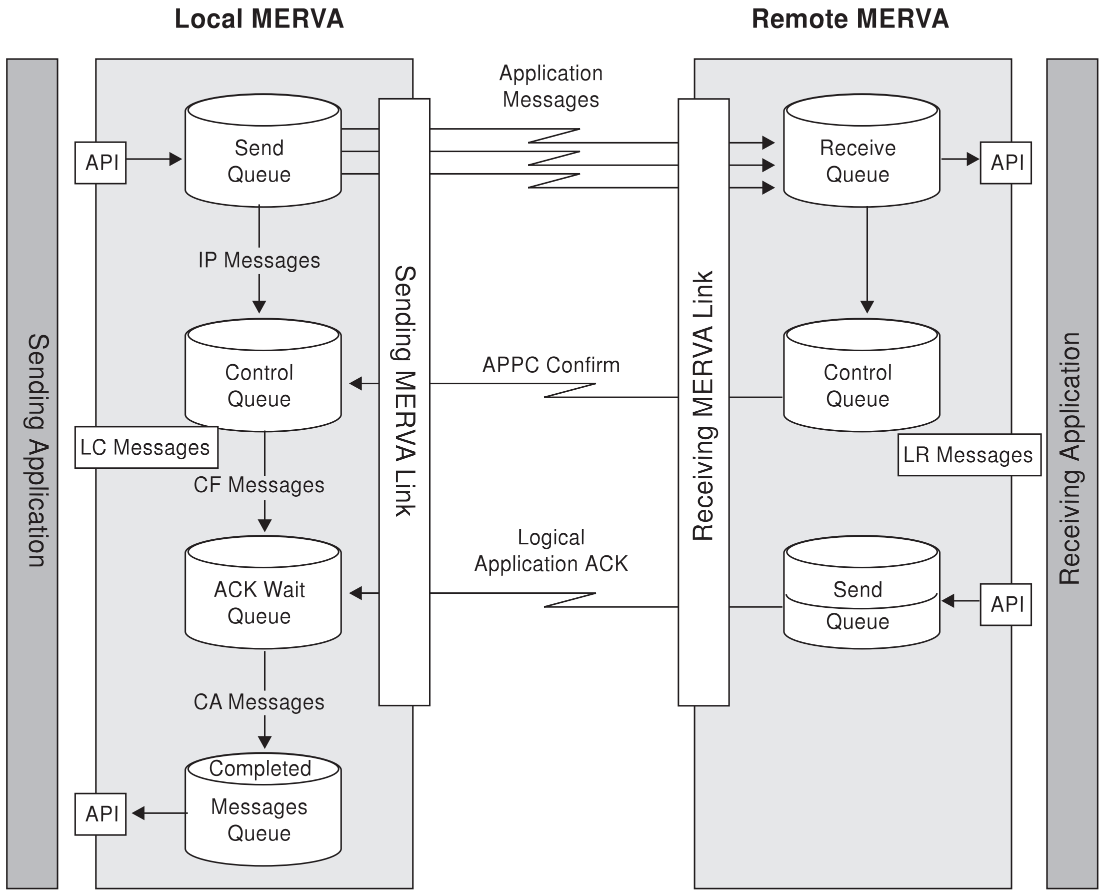


Figure 20. MERVA Link Overview

The MERVA System Control Facility (program EKAEMSC) is an end-user function that provides the means to supervise the MERVA ESA and MERVA Link functions in the local MERVA ESA system and in a partner MERVA ESA system. For details about the MERVA System Control Facility, refer to the MERVA for ESA Operations Guide.

The MERVA Link Partner Table (EKAPT) defines and describes the following MERVA Link resources:

- An application support process (ASP) entry describes the characteristics and the requirements of a customer's message transfer application.
- A message transfer process (MTP) entry describes the characteristics and the identifiers of its partner MTP. An MTP supports the transfer of messages between the partner MERVA ESA systems.
- A system control process (SCP) entry describes the characteristics and the identifiers of its partner SCP. An SCP supports the transfer of MERVA ESA commands to the partner system, and the execution of a command in the partner system. An SCP is not shown in Figure 20.

CICS, APPC/MVS, or APPC/IMS services are used by MERVA Link to send and receive messages. The sending transaction is started by DSLQMGT when a message is stored in one of the send queues defined for MERVA Link. The sending transaction may also be started by an end user in the MERVA Link control function MSC or by the MERVA Link CICS ASP Monitor. The receiving transaction is started by CICS, APPC/MVS, or APPC/IMS when a message is received from a partner.

The connections to other MERVA systems use SNA APPC (LU 6.2) services provided by CICS or APPC/MVS. MERVA Link USS supports connections to other MERVA systems using SNA APPC or TCP/IP services.

MERVA Link Programs

There are two groups of MERVA Link programs in a MERVA ESA installation:

- Application support programs, which, when executed, start application support processes (ASPs)
- Message transfer programs, which, when executed, start message transfer processes (MTPs)

The Application Support Programs: The application support programs perform all functions that require MERVA ESA services. The group includes:

- The sending application support program (EKAAS10), which processes outgoing messages
- The receiving application support program (EKAAR10), which processes incoming messages

Each application support process (ASP) is associated with one partner ASP in a partner system. It can send messages to and receive messages from only this partner ASP. A specific protocol is used between the partner ASPs: the peer-to-peer protocol at the Application Support Layer, called the MERVA Link Application Support Protocol (P2).

Data is exchanged in the form of P2 protocol data units (P2 PDUs). They carry the application or acknowledgment messages.

Each ASP is permanently associated with one MTP in the local system. The ASP may call its MTP via an application support filter (ASF) program. The ASF can perform additional processing with the P2 PDU.

The Message Transfer Programs: The message transfer programs perform the actual message transfer. The group of programs includes:

- The message transfer service program (EKASP10), which is the interface between an application support program and a sending or receiving message transfer program
- The sending message transfer programs (EKATS10 and EKATPO1), which send messages to their partner programs
- The receiving message transfer programs (EKATR10 and EKATPI1), which receive messages from their partner programs
- The Back-to-Back TP Mirror program (EKATM10), which provides for transferring messages between ASPs in the same MERVA Link node

Each message transfer process (MTP) is associated with one partner MTP in a partner system. It can send messages to and receive messages from only this partner MTP. A specific protocol is used between the partner MTPs: the peer-to-peer protocol at the Message Transfer Layer, called the MERVA Link Message Transfer Protocol (P1).

Messages are exchanged in the form of P1 protocol data units (P1 PDUs). They carry a P2 PDU as their content.

The message transfer is processed synchronously, based on SNA LU 6.2 protocols (advanced program-to-program communication, abbreviated to APPC). One or more messages are transferred to the partner MTP, and a transfer confirmation is returned in the same conversation. The confirmation is requested by the sending MTP from the receiving MTP when a message transmission window is exhausted (that is, when an agreed number of messages has been sent), or when the last message in the send queue cluster has been transferred.

Each ASP and MTP requires an entry in the partner table. The association of an ASP with its MTP, and with the partner ASP and MTP, is defined in the Partner Table. Figure 21 gives an overview of the ASPs and MTPs used for one MERVA Link connection in cooperating MERVA ESA installations.

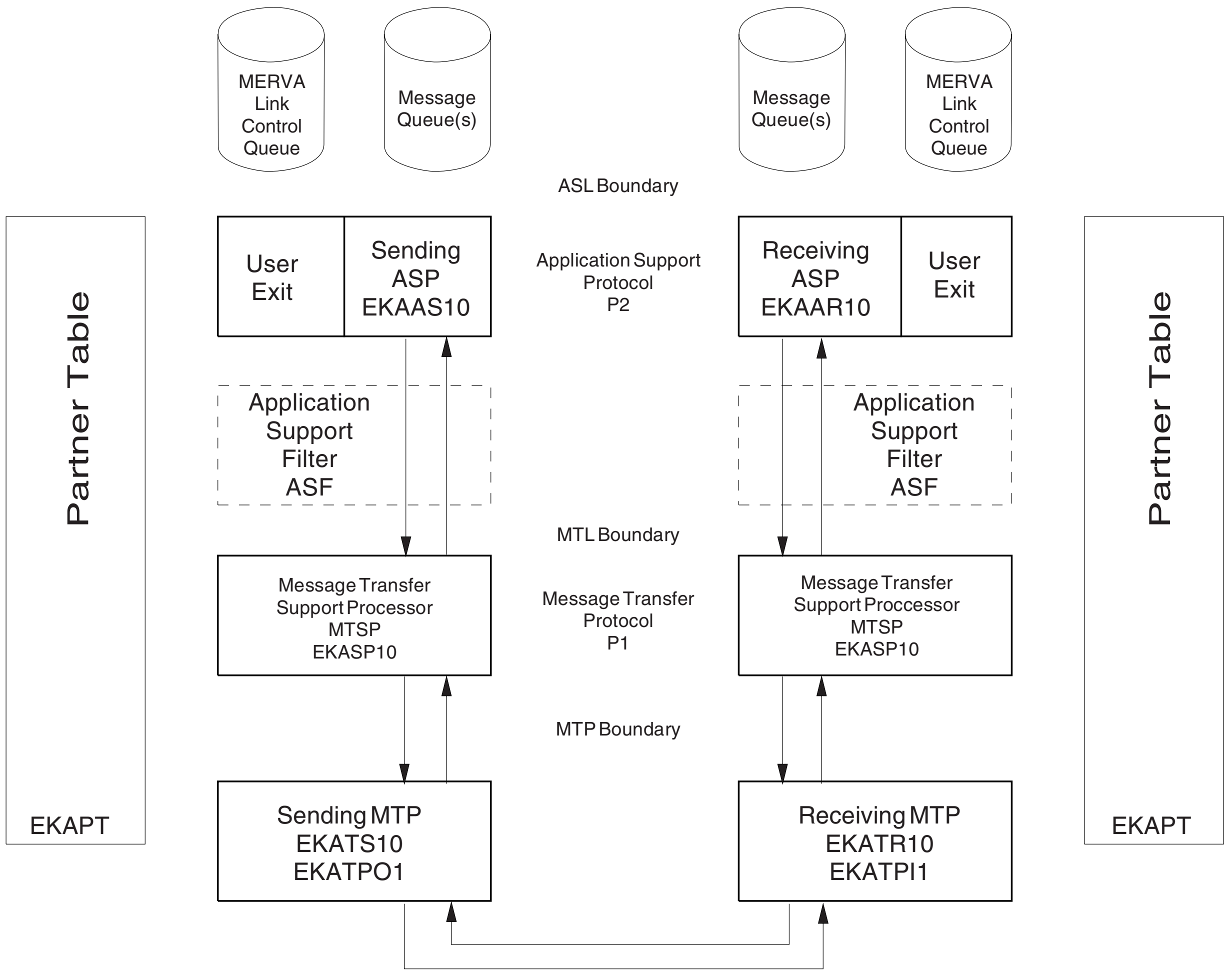

MERVA Link Control Queue
Message Queue(s)
Message Queue(s)
MERVA Link Control Queue
ASL Boundary
Partner Table
EKAPT
User Exit
Sending ASP EKAAS10
Application Support Protocol P2
Receiving ASP EKAAR10
User Exit
Partner Table
EKAPT
Application Support Filter ASF
Application Support Filter ASF
MTL Boundary
Message Transfer Support Processor MTSP EKASP10
Message Transfer Protocol P1
Message Transfer Support Proccessor MTSP EKASP10
MTP Boundary
Sending MTP EKATS10 EKATPO1
Receiving MTP EKATR10 EKATPI1

Figure 21. Overview of a MERVA Link Installation

Detailed information about the MERVA Link protocols and PDUs can be found in the *MERVA for ESA Advanced MERVA Link* manual.

The Partner Table (EKAPT): The partner table, which has the name EKAPT, defines the characteristics of any message transfer application and any partner MERVA system that must be operated at the local MERVA system. For each message transfer application, the following is defined:

1. An ASP, including application information and execution parameters, for example:
   - The name of this ASP
   - The MERVA Link node name of the partner system
   - The name of the partner ASP in the partner system
   - The names of the send queues and the application control queue
   - Routing parameters for sent messages
   - Routing parameters for received messages and reports
   - Security provisions (authentication and encryption)
   - Formats of the messages transferred
   - The module number of an MFS user exit to be called
   - Message transfer journaling requirements
   - Message delivery error handling requirements
   - The names of up to three application support filters (ASF)
   - The name of the MTP associated with this ASP
2. An MTP, including information about the connection with the partner, for example:
   - The name of this MTP
   - The transaction code of the receiving MTP in the partner system
   - The identifier of the connection to the partner system
   - The communication type used (APPC or BTB)
   - The name of the ASP associated with this MTP

For each partner MERVA system, which must be operated at the local MERVA system, a system control process (SCP) defines partner system identifiers and execution parameters, for example:
- A nickname and the MERVA Link node name of the partner MERVA ESA system
- The identifier of the connection to the partner system
- Local operator authorization
- Remote operator authorization

The partner table is generated by coding EKAPT macro statements, which are described in the *MERVA for ESA Macro Reference*. The name of the partner table, EKAPT, cannot be changed.

Application Support Concepts and Functions

Message Integrity: When sending and receiving messages, MERVA Link ensures that any loss of a message is detected and reported, and it ensures that no message is stored twice in the queues of the receiving MERVA Link.

For this purpose, MERVA Link uses application control queues and message sequence numbers. The message sequence numbers are used by the sending and receiving ASPs to verify that no message is missing in a sequence of messages, and to detect duplicate messages. The application control queues are MERVA ESA queues. Each ASP defined in the Partner Table has one application control queue. An application control queue must not be accessed by programs other than MERVA Link programs.

MERVA Link stores the following messages in an application control queue:

- The last confirmed control message (LC control message).

  This message is generated by MERVA Link when an empty application control queue is found, and updated during a sending process. The LC control message keeps track of the status of the sending process. This status can be displayed in the MERVA Link control function MSC.

  The LC control message contains the following data:

  – Last confirmed message sequence number (LC MSN)
  – Active window size (maximum number of unconfirmed messages)
  – Date and time of the status
  – ASP AS status identifier
  – ASP MT status code
  – Diagnostic Code
  – Buckslip, an operator message explaining the ASP status

  There is only one LC control message in an application control queue.
- The in-process messages (IP messages).

  These messages are moved from the send queue to the application control queue when they are prepared for sending, and they stay there until the delivery is confirmed by the partner ASP. The message sequence numbers (MSN) have already been assigned to the IP messages. The maximum number of IP messages in the application control queue is equal to the active window size.

  If the message transfer fails, that is, if the confirmation does not arrive, the IP messages remain in the application control queue. When the ASP is started again, the IP messages are sent to the partner ASP with the same message sequence numbers assigned earlier. Messages from the send queues are only processed after the IP messages have been confirmed and routed.
- The last received control message (LR control message).

  This message is a copy of the last message that was received by an ASP from its partner. The LR control message is used to prevent routing of duplicate messages to the receive queues. When a message is received, its MSN is compared with the MSN of the LR control message:

  – If the received MSN is lower than or equal to the LR MSN, but within the window, the received message is a duplicate. It is not routed but confirmed to the partner ASP. The LR control message is not changed.
  – If the received MSN is one higher than the LR MSN, the received message is the expected new message. It is routed and confirmed to the partner ASP. The LR control message is replaced by the new message with the new MSN.
  – If the received MSN is more than one higher than the LR MSN, at least one message is lost. An error indication is sent to the partner ASP instead of the confirmation. The received message is not routed. The LR control message is not changed.

  The simultaneous routing of a received message and the replacement of the LR control message is ensured by the use of the MERVA ESA queue service ROUTE with automatic delete.

  There is only one LR control message in an application control queue.

Message Sequence Number: The message sequence number (MSN) is a 4-digit number from 0001 to 9999. After 9999, it wraps around to 0001. The MSN is assigned to the messages for sending in ascending order when they are read from one of the send queues and stored in the application control queue. When the transmission of a message fails, all retries are done with the MSN that was originally assigned.

The sending ASP checks the sequence of the MSN in IP messages by means of the MSN in the LC control message and in the IP messages. The receiving ASP checks the sequence of the MSN by means of the LR control message.

Window Size: The communication between two partner ASPs uses a window size. That means, the sending ASP sends as many messages as defined by the window size, and then requests a transfer confirmation of all messages in the window.

A delivery error report may identify a specific IP message. This report implies a delivery confirmation for all IP messages in the window that preceded the reported message. If the delivery error report does not identify a specific IP message, all messages of the window must be considered as undelivered.

Sending Messages: MERVA Link finds the messages for sending in the send queues defined in the partner table entry of an ASP. This set of send queues is called the send queue cluster of that ASP. The sending ASP performs the following steps when preparing a message for sending:

1. Retrieve the message from one of the send queues.
2. Call the user exit specified in the partner table entry with the function code “S” (ready to send message). The user exit can decide if the message should be transferred, or if it should immediately be routed using the routing table defined for the application control queue (close the ASP temporarily for that message).
3. Provide the MERVA Link control information (for example, the message sequence number).
4. Store the message as in-process (IP) message in the application control queue and delete it from the send queue.
5. Call the user exit specified in the partner table entry with the function code “O” (outgoing message). This user exit must check whether an application message or an acknowledgment message must be transferred.
   - For an application message it indicates to the ASP that an application message must be transferred.
   - For an acknowledgment message, for example, a SWIFT or telex acknowledgment, the user exit provides the acknowledgment data, and indicates to the ASP that an acknowledgment message must be transferred. The unique identifier of the reported message is transferred as part of the acknowledgment message.

   The *MERVA for ESA Customization Guide* shows how to code such a user exit.
6. If specified, perform:
   - Authentication to ensure that the message text is not altered during transmission
   - Encryption to make the message contents unreadable during transmission
7. Write the application message or the acknowledgment message to the MERVA ESA journal. The journal record shows the data before encryption.
8. Call the associated MTP.

More than one message may be in process by an ASP. The maximum number of these messages is defined by the window size. When a message window is exhausted, the sending ASP asks for a delivery confirmation for all messages in the window.

When the confirmation has been obtained, the ASP updates the LC control message and routes the IP messages from the application control queue using the routing table defined for the application control queue. Target queues can be:
- Queues for further message processing.
- A queue where the messages wait for acknowledgments (that is, an ACK wait queue). The unique identifier of the application message must be used as a key for the ACK wait queue.

Before a confirmed message is routed, the user exit specified in the partner table entry is called with the function code “C” (confirmed message). The user exit can provide transfer confirmation control information in application-specific terms at this place.

When all IP messages have been routed, more messages are processed from the send queue cluster.

Receiving Application Messages: The receiving ASP performs the following steps when an application message is received:
1. If specified, it performs:
   - Decryption to make the message contents readable
   - Authentication to ensure that the message text was not altered during transmission
2. Writes the decrypted message to the MERVA ESA journal.
3. Checks the message sequence number (MSN). The subsequent steps are only performed if the message is a new message with the correct MSN.
4. Calls the user exit defined in the partner table entry of this ASP with the function code “I” (incoming message). The user exit may check the message or provide additional information for the subsequent routing.
5. Routes the message to the application control queue and to one or several receive queues for further processing using the routing table defined for the application control queue. The LR control message with the new MSN is replaced in the application control queue by the same routing request.

Receiving Acknowledgment Messages: Dependent on the requirements defined in the PT ASP entry, the receiving ASP performs the following steps when an acknowledgment message is received:
1. It writes the message to the MERVA ESA journal
2. It checks the message sequence number (MSN). The subsequent steps are only performed if the acknowledgment message is a new message with the correct MSN.
3. If the receiving ASP must correlate the acknowledgment message with the reported message, it tries to find the reported message in the ACK wait queue defined in the partner table entry of this ASP. For this purpose, the ACK wait queue must be defined with the MERVA Link field EKAAMSID as key 1.
4. If the reported message is not found in the ACK wait queue (this can happen if the applicable message is still in the application control queue of this ASP as an IP message because it is not yet confirmed), the ASP tries to find it in the application control queue.

5. If the applicable message is found in the ACK wait queue or in the application control queue, the user exit defined in the partner table entry of this ASP is called with the function code “R” (incoming report). The user exit can merge report data of the acknowledgment message with the reported message. For a SWIFT acknowledgment, for example, the header, the trailer, and the MSGACK field can be updated.
6. The acknowledged message is routed as specified by the routing table defined for the application control queue. The LR control message in the application control queue is replaced by the updated message in the same routing request. The application message in the ACK wait queue is deleted by a separate request.

   Target queues can be queues for further processing, or the ACK wait queue again if more acknowledgments are expected for the message. If an IP message was acknowledged, the message is routed back to the application control queue because of the pending delivery confirmation.
7. If the reported message is not found in the ACK wait queue nor in the application control queue, the acknowledgment message is formatted as a MERVA Link control message (MCTL) and routed using the routing table defined for the application control queue. The LR control message in the application control queue is replaced by this message in the same routing request.

MERVA Link Message Classes: A message class is assigned to each message that is processed by MERVA Link. The message class consists of 2 characters. It is contained in the field EKACLASS in the MERVA ESA internal message buffer (TOF). The EKACLASS field must be used as key 1 in all application control queues.

The message class can be used for routing decisions. The message class of a message can change depending on its processing status within MERVA Link. The classes LC and LR of the MERVA Link control messages, however, never change.

The following message classes are defined for MERVA Link:

LC Is the last confirmed control message in the application control queue. The LC control message is generated and maintained by a MERVA Link ASP.

LR Is used for:
- The last received control message in the application control queue
- All received application and acknowledgment messages that were routed to receive queues
- All messages from ACK wait queues when they are routed together with the acknowledgment information (acknowledged messages)

IP Are the messages that are being processed by MERVA Link. The in-process messages have been moved from the send queues to the application control queue. They are being transferred to the partner ASP, or waiting for a transfer confirmation.

CF Are confirmed messages. They were IP messages. The transfer was confirmed, and they have been routed to an ACK wait queue or a queue containing transferred messages.

CA Are confirmed messages that have been acknowledged before the transfer confirmation is received. They were IP messages. The transfer was confirmed, and they have been routed to an ACK wait queue or a queue containing transferred messages.

RC Are recovered messages. They have been IP messages. The MERVA Link command **recover** was used to route (copy) them to other queues using the routing table defined for the application control queue.

RC messages are copies of IP messages in the application control queue. They contain a PDM indicator in a MERVA Link control field.

RI Are recovered messages. They have been IP messages, which could not be successfully delivered to the receiving application. Messages of the class RI have either been automatically recovered from a delivery error, or the MERVA Link command **iprecov** was used to route (move) them to other queues using the routing table defined for the application control queue.

RI messages do not contain a PDM indicator in a MERVA Link control field.

RM Are messages that have been immediately removed from an outbound window upon request of an ASF.

RS Are messages of a closed ASP that are routed from the send queues to other queues using the routing table defined for the application control queue.

RR Are messages that are routed from a send queue to other queues upon the request of the user exit defined for the ASP. The routing table defined for the application control queue is used.

The message class RR is an example. A MERVA Link installation can define any other message class instead of RR for this purpose, except those used by MERVA Link.

MERVA Link Control Fields: MERVA Link adds control information to the messages that are sent to a partner or received from a partner. These fields contain, for example, the sender's and receiver's address, a time stamp, and the MERVA Link message identifier.

The control fields become part of a message in the MERVA ESA internal format and in a MERVA ESA queue.

The control fields can be displayed on screen terminals and printers together with the message, and they can be added to a network format of a message using an appropriate MCB.

A list of the MERVA Link control fields is contained in *MERVA for ESA Messages and Codes*.

Journalling: MERVA Link journals all message traffic and some other events. Details about the layout of these journal records can be found in "Appendix A. Journal Record Layouts" on page 185.

Financial Message Transfer/ESA (FMT/ESA)

| MERVA-to-MERVA Financial Message Transfer/ESA (FMT/ESA) uses the
| capabilities of MERVA Link ESA or MERVA-MQI Attachment to transfer SWIFT
| messages between two MERVA ESA systems in a way similar to the way
| MERVA ESA transfers messages via the SWIFT network. FMT/ESA can exploit
| existing intra-enterprise or inter-enterprise networks, and is fully transparent to
| applications.

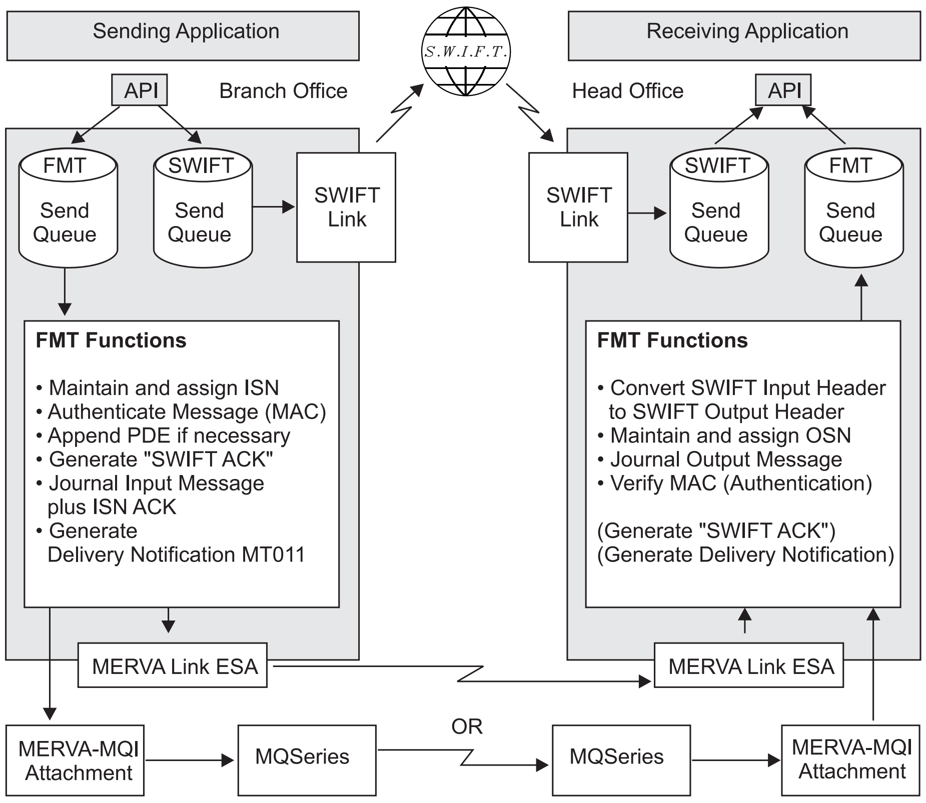


Figure 22. The FMT/ESA

FMT/ESA does the following:

- Prepares SWIFT input messages for a follow-on transfer
- Requests that MERVA Link or MERVA-MQI Attachment transfer the messages from the sending to the receiving MERVA ESA system (messages can be transmitted either in SWIFT format or MERVA ESA queue format)
- Transforms the received SWIFT input messages to SWIFT output messages
- Routes (or lets MERVA Link or MERVA-MQI Attachment route) the output messages to target queues
- Provides a SWIFT acknowledgment for the messages transmitted from the sending MERVA ESA system (the acknowledgment may be generated either in the sending or in the receiving MERVA ESA system)
- Provides a SWIFT delivery notification (message type 011) (if requested by a received SWIFT input message and the customer)
- Journals SWIFT input and output messages and their acknowledgments
- Authenticates SWIFT input and output messages, if requested by the customer
- Checks SWIFT input messages (if using MERVA Link or MERVA-MQI Attachment) and output messages (only if using MERVA-MQI Attachment), if requested by the customer
- When using MERVA Link, and when there is a message recovery for an inoperable MERVA Link ASP using the MERVA Link command RECOVER:
  - Appends a PDE trailer generated by FMT/ESA to a SWIFT input message that can be routed to a SWIFT ready queue
  - Appends a PDE trailer generated by the SWIFT Link to a SWIFT input message before MERVA Link transfers the message to the receiving MERVA ESA

You can also set up FMT/ESA to run on a single MERVA ESA system. This might be of interest for an installation that has one instance of MERVA ESA serving more than one bank, and where these banks currently exchange messages among each other via the SWIFT network. By using FMT/ESA, messages and acknowledgments are routed from the sender to the recipient and back to the sender without ever leaving the MERVA ESA system, even though they look and behave exactly like messages sent and received via the SWIFT network. This is a less expensive way to exchange messages.

Which user exit is called as the FMT/ESA program depends on whether you are using MERVA Link or MERVA-MQI Attachment:

- For MERVA Link, the MFS user exit EKAMU044 represents the FMT/ESA program. MERVA Link calls a user exit at different processing stages before a message is sent and after a message or an acknowledgment is received. Figure 21 on page 137 shows the close connection of a user exit to a MERVA Link sending and receiving ASP, respectively. The functions of the user exit are described in “Sending Messages” on page 140, “Receiving Application Messages” on page 141, and “Receiving Acknowledgment Messages” on page 141.
- For MERVA-MQI Attachment, the MFS user exit DSLKQ044 represents the FMT/ESA program. MERVA-MQI Attachment calls a user exit at different processing stages before a message is sent and after a message or an acknowledgment is received.

FMT/ESA Message Classes: When FMT/ESA is used with MERVA Link, the following additional message classes can be contained in the EKACLASS field:

SE An error was detected before a SWIFT input message was sent to the partner MERVA Link. As a result, a complete SWIFT input message cannot be generated.

Such a message is routed from a send queue to a verification queue and is not transmitted.

IE An error was detected after a SWIFT input message was received in the partner MERVA Link. As a result, a SWIFT output message cannot be generated.

Such a message is routed to a local error queue at the message-receiving side.

LE An error was detected after the MERVA Link confirmation was received indicating that a SWIFT input message was successfully received in the partner MERVA Link. As a result, the SWIFT acknowledgment for the same SWIFT input message contained in the ACK wait queue, cannot be generated.

Such a message is routed to a local error queue at the message-sending side.

PE An error was detected after an acknowledgment message was received. As a result, the SWIFT acknowledgment data cannot be inserted into the appropriate SWIFT input message contained in the ACK wait queue.

Such a message is routed to a local error queue at the message-sending side.

You find more information on the FMT/ESA in the *MERVA for ESA Customization Guide* and in the *MERVA for ESA Operations Guide*.

MERVA Link USS

MERVA Link UNIX System Services (MERVA Link USS) is an implementation of MERVA Link functions in the OS/390 UNIX System Services environment. The MERVA Link USS functions provide for the routing of MERVA Link conversations synchronously from an SNA APPC network to a TCP/IP network, and vice versa (MERVA Link Gateway). These functions also provide for customizing and administering the routing functions.

MERVA Link USS is not associated with a particular MERVA ESA installation. It cannot deliver inbound messages to a MERVA ESA messaging application, nor submit application messages for transmission to a partner node.

The resources provided or used by MERVA Link USS are:

Message Handling Programs

The functionality of the sublayers defined in the MERVA Link Message Handling System model is implemented in a set of MERVA Link USS message handling programs. Each sublayer function implementation is provided in two versions, one for the outbound and one for the inbound tower. Four programs are provided for the lowest MERVA Link sublayer that provides outbound and inbound SNA APPC and TCP/IP services.

The MERVA Link USS message handling programs that are used in the MERVA Link USS gateway scenario are provided with all functions. The other MERVA Link USS message handling programs are provided for installation verification and test purposes only.

Control Programs

MERVA Link USS provides a number of control programs for customization and administration purposes:

- The MERVA Link USS Application Control Daemon (ACD) provides the functions to generate the MERVA Link USS Application Control Table (ACT). The ACT contains all MERVA Link USS customization parameters. A MERVA Link USS Application Control Daemon process and its ACT represent an active MERVA Link USS instance.
- The MERVA Link USS Application Control Command Application (ACC) provides the functions to operate an active MERVA Link USS instance. ACC provides a USS command interface in three modes (single command mode, conversation mode, batch input mode).
- The MERVA Link USS Conversation Security Control Application (ACS) provides the functions to store confidential information that is used for an outbound TCP/IP conversation in a MERVA Link USS security file.
- Some other programs are provided to perform advanced MERVA Link USS tasks. For more information, refer to the *MERVA for ESA Advanced MERVA Link* manual.

Control Files

MERVA Link USS uses a number of control files in the OS/390 USS Hierarchical File System (HFS) for customization and administration purposes:

- The MERVA Link USS customization parameters are provided by the MERVA Link USS administrator in a MERVA Link USS Configuration File (CFG). The ACD uses a configuration file to create the ACT.

- MERVA Link USS conversation security information is provided by the MERVA Link USS administrator in a MERVA Link USS Security File (SEC) using the ACS application. The ACD uses security files to store security information in the ACT.
- MERVA Link USS message handling programs can generate a processing trace, and store the trace information of one conversation in a trace file (TRC). The trace facility can be activated with various trace levels, or be deactivated.
- MERVA Link USS message handling programs write inbound and outbound error reports to a MERVA Link USS error report log file (ERR). The error report log facility is always activated.

MERVA Link USS Programs

MERVA Link USS contains application support programs, message transfer programs, and control programs.

Application Support Programs: Application support functions are not part of a MERVA Link USS Gateway and not part of MERVA Link USS of MERVA ESA V4.1. This is why the application support (AS) programs **ekaaso**, **ekaasi**, **ekap2o**, and **ekap2i** are provided without full functionality for installation verification and test purposes only. The programs **ekaaso** and **ekaasi** can be used only for test purposes. They cannot access a MERVA installation to retrieve messages from a MERVA Link send queue nor deliver an inbound message to a receive queue. However, the two P2 programs **ekap2o** and **ekap2i** provide enough functionality to perform a number of MERVA Link USS control functions that require P2 services.

Message Transfer Programs: Message transfer functions are used by a MERVA Link USS Gateway to route an inbound conversation to a partner MERVA Link system. The necessary functions are contained in the inbound message transfer programs **ekatpi** and **ekatci**, the outbound message transfer programs **ekatpo** and **ekatco**, and in the inbound P1 program **ekap1i**.

The inbound P1 program contains all outbound P1 functions from the program **ekap1o** that are required in a routing process. This is why the outbound P1 program **ekap1o** is not used by a MERVA Link USS routing process. The outbound P1 program is, however, used by the outbound P2 program for some control functions (as described above).

The two programs **ekatpo** and **ekatpi** provide the necessary MERVA Link functions to communicate with a partner system using SNA APPC services. The inbound SNA APPC TP **ekatpi** is started upon request of a MERVA Link sending system directly or indirectly by the APPC/MVS transaction scheduler (ASCH). APPC/MVS must be customized to provide this service.

The two programs **ekatco** and **ekatci** provide the necessary MERVA Link functions to communicate with a partner system using TCP/IP services. The inbound TCP/IP TP **ekatci** is started upon request of a MERVA Link sending system directly or indirectly by the OS/390 USS Internet Daemon (InetD). InetD must be customized to provide this service.

Routing Process Program Call Sequence: Figure 23 on page 148 shows a sample message routing scenario that interconnects a MERVA Link running on a CICS ayatem and a MERVA Link running on an AIX® system. Messages are routed from a sending SNA APPC node via the MERVA Link USS Gateway Node to a receiving

TCP/IP node. The receiving TCP/IP node is assumed to have no SNA services installed.

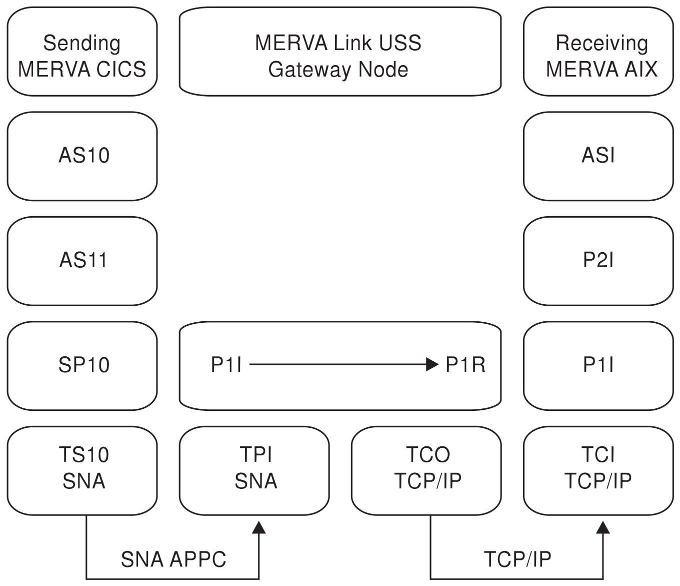


Figure 23. Sample MERVA Link USS Routing Scenario

The program call sequence in a gateway node is:

ekatpi The inbound TP supporting SNA APPC. It receives the data sent by its partner TP using APPC/MVS services, and passes this data to ekap1i. The TP ekatci would be the inbound TP if the originator were a TCP/IP node.

ekap1i
The inbound P1 program. It analyzes and decodes P1 PDUs (message envelopes), and learns that the intended destination node is not the local MERVA Link node. The inbound request is passed to the routing functionality in EKAP1I in this case.

The routing functionality in ekap1i (also called ekap1r) determines the parameters of the intended destination node and passes the inbound request to the applicable outbound TP.

ekatco The outbound TP supporting TCP/IP. It transmits the data passed by the routing functionality in ekap1i to its partner TP using TCP/IP socket services. The program ekatpo would be the outbound TP if the recipient were an SNA APPC node.

Control Programs:

Application Control Daemon (ACD)
A long lasting OS/390 USS process (daemon) that represents a MERVA Link USS instance. The ACD generates and owns the MERVA Link USS Application Control Table (ACT), a USS shared memory region that contains the customization parameters of a MERVA Link USS instance. No other MERVA Link USS process can execute without access to an ACT.

Application Verification Control Daemon (VCD)
A daemon that can be used to verify a MERVA Link configuration in a test environment.

Application Control Command (ACC)
A command program, used in the OS/390 USS interactive shell environment, that lets authorized users display MERVA Link USS resources and modify MERVA Link processing parameters.

Application Verification Control Command (VCC)
A command program that can be used to display data in the Application Control Table (ACT) of a VCD.

Conversation Security Control (ACS)
A program that provides a USS shell command interface to store confidential information in MERVA Link USS security files.

Conversation Security Control Application (VCS)
A program that can be used in a configuration verification environment.

Change Security Information Application (CSI)
A program that provides a USS shell command interface to store confidential conversation security information in a partner MERVA Link system.

MERVA Link USS Gateway Functions

MERVA Link USS Gateway supports all functions necessary to route an inbound MERVA Link SNA APPC conversation to a partner system using the MERVA Link TCP/IP protocol, or vice versa.

Accepting an Inbound Conversation: How MERVA Link USS accepts an inbound conversation and schedules an inbound transaction program depends on the connection type. If the connection type is:

- SNA APPC, the APPC/MVS transaction scheduler ASCH schedules **ekatpi** in an APPC/MVS initiator (an OS/390 region). It accepts the inbound conversation as specified by the CPI-C services provided by APPC/MVS.
- TCP/IP, the OS/390 USS Internet superserver **InetD** schedules **ekatci**. It accepts the inbound conversation as specified by the TCP/IP stream socket services.

Routing an Inbound Conversation: The inbound P1 program (ekap1i) checks whether the local MERVA Link node name and the recipient node name in the first PDU segment of an inbound conversation, a Probe PDU envelope, match. If they match, the local system houses the intended destination node.

If the two MERVA Link node names do not match, the normal situation for a MERVA Link Gateway, the local system acts as a gateway that routes the inbound conversation to the intended destination node.

The gateway function is performed if MERVA Link finds a set of intersystem connection (ISC) parameters in its customization data that describes a connection to the intended destination node. The preferred connection type is the connection type that is different from the inbound connection. This means, the preferred route for an inbound SNA APPC conversation is an outbound TCP/IP connection, and vice versa.

Connecting to a Partner System: The partner system can be connected via SNA APPC or TCP/IP, whatever communication protocol is supported by the partner system. If the partner system supports both communication protocols and the two partner systems are actually connected in those two ways, there is a choice between the two connections. One of these connections is called the **preferred connection**, the other is called the **alternate connection**.

Conversation Security: SNA APPC conversation security is completely handled by APPC/MVS for a MERVA Link USS gateway. MERVA Link USS functions are not concerned with SNA APPC conversation security.

A security function equivalent to that provided by SNA APPC is not provided by TCP/IP. This is why MERVA Link USS provides a conversation security function in the TCP/IP environment. The type of this conversation security function is equivalent to SNA APPC conversation security type PROGRAM. This means that an outbound MERVA Link USS conversation using TCP/IP services must provide a client user ID and password if the server process (the receiving MERVA Link system) asks for it.

The security of an inbound TCP/IP conversation is checked as a mandatory function by the MERVA Link USS TCP/IP server (ekatci). Conversation security information must be contained in the PROBE of an inbound TCP/IP conversation.

Confirming an Inbound Conversation: A request for confirmation received from the partner system is always passed to the inbound P1 program. The P1 program determines whether it must route that request to another partner system (gateway scenario), or whether it can issue the confirmation immediately (local message delivery).

MERVA Link USS Control Functions

Application Control Command Application (ACC): The MERVA Link USS Application Control Command application (ACC) is the means to control the status and the execution of MERVA Link USS. ACC can be used in an OS/390 USS interactive command shell environment, in a USS batch environment (BPXBATCH), and in an MVS batch environment. The command or program **ekaacc** calls ACC in the OS/390 USS environments. Program **EKAACC** calls ACC in the MVS environment.

An ACC interactive command begins with an ACC command name that must be entered in lowercase letters. A subset of the ACC command names is followed by a resource name. Resource names can be, for example, an ASP name, a partner system node name, a keyword, or a MERVA Link diagnostic code. MERVA Link resource names are normally made of uppercase characters, however, they can be entered with lowercase letters. An uppercase translation is performed automatically (where applicable). Help information is displayed as response to an invalid ACC command.

A conversation mode is supported by ACC. The USS command **ekaacc sc** starts the ACC conversation mode in a window. All data entered in a window in ACC conversation mode is interpreted as an ACC command. The command prompt **ekaacc** is a reminder of the fact that the window is in ACC conversation mode.

A batch input mode is supported by ACC. The program calls **ekaacc si** (BPXBATCH) and **EKAACC si** (MVS) start ACC in batch input mode. The ACC commands are retrieved from **stdin**, and the command output is written to **stdout**.

ACC cannot be used when the MERVA Link USS Daemon EKAACD is not active. EKAACD generates and owns the MERVA Link Application Control Table (EKAACT), which is the main resource of the MERVA Link USS Control Facility.

ASP related functions are supported by ACC. ASPs are, however, supported by the MERVA Link USS Gateway for installation verification and test purposes only. All ACC functions related to ASPs are therefore not applicable in a production environment.

Conversation Security Control Applications: Conversation security information is the information that authorizes a client process to access a server process. A server process specifies whether a client process must provide conversation security information, or whether it provides its service without client authentication at the conversation level.

Conversation security information is a client **user ID** (called **user name** in the USS environment), and a client user **password**. Passwords are considered as confidential, and must not be stored in plain text format on any storage media. This is why conversation security information must be handled separately from MERVA Link USS configuration files in MERVA Link USS security files.

The MERVA Link USS Conversation Security Control Application (ACS, VCS) is the means to specify, store, and modify conversation security information in local MERVA Link USS security files. The MERVA Link USS Daemon (ACD) refers to the information in the security files when it creates the MERVA Link USS ACT. MERVA Link USS message processing programs refer only to conversation security information in the ACT.

The MERVA Link USS application Change Security Information in a Partner System (CSI) provides a means to specify and verify conversation security information in the local system, transmit it to a partner system, and store it in the configuration facility of a partner MERVA Link system. The CSI application can be used only for partner MERVA Link systems that support the CSI application as an inbound P2 service. The CSI application is supported by MERVA Link USS. It is not supported by MERVA Link ESA.

Routing Process Trace: The MERVA Link USS Processing Trace Facility provides diagnosis information about MERVA Link USS message handling processes. The trace facility parameters in the ACT determine the HFS directory for the trace files of all routing processes, the trace level (amount of data to be traced), and the trace file naming rules. Parameters of the MERVA Link USS processing trace facility in the ACT can be modified by ACC commands.

If a MERVA Link USS receiving process cannot attach the ACT, it tries to write a trace to the /trc/ subdirectory of the applicable MERVA instance directory. If the name of the latter directory is not available, a MERVA Link USS receiving process tries to write a trace to the /tmp/ directory.

If the request to open a trace file fails, a MERVA Link USS receiving process tries to write a trace to the /tmp/ directory. The unique trace file names are **ekatpi.trace** and **ekatci.trace** in all exceptional situations.

Daemon Activity Trace and Report: The MERVA Link USS Application Control Daemon (ACD) supports an activity trace. This trace can be requested by a command line parameter when the daemon is started. The trace file is written to the directory specified in the trace request parameter (**trc**). The trace file name is ′**ekaacd.t.**MMDDhhmmss′. The last part of the trace file name is a date-time stamp. It identifies the date and time when the daemon was started.

The MERVA Link USS Configuration Verification Daemon (VCD) supports an extended activity trace that is also referred to as the VCD verification report. A VCD report can be requested by a VCD command line parameter when the daemon is started. The report file is written to the directory specified in the report request parameter (**rep**). The report file name is '**ekavcd.r.**MMDDhhmmss'. The last part of the trace file name is a date-time stamp. It identifies the date and time when the daemon was started.

Error Report Log: The MERVA Link USS Error Report Log Facility writes, to a permanent log file, error information received from a partner system, and any error report to be sent to a partner system. The log file is associated with the communication direction (inbound or outbound), and associated with the applicable partner node.

Error reports are always written to the log files, independent of the processing trace level. The information written to an error report log file is, however, the same information that is written to a trace file (as far as intersystem error information is concerned).

Error report log entries are always appended to the applicable existing file. MERVA Link USS does not erase the content of an error report log file, or parts of it.

MERVA-MQI Attachment

MERVA-MQI Attachment enables communication between MERVA ESA and MQSeries running on MVS/ESA or VSE/ESA. MQI stands for Message Queue Interface, a programming interface provided by the MQSeries queue manager that lets application programs access the message queueing services of the MQSeries. Message queueing is a programming technique by which a program communicates with other programs by putting messages into queues.

MERVA-MQI Attachment enables the exchange of messages between MERVA ESA and other applications on platforms where the MQSeries is installed. That is, applications on IBM or non-IBM platforms are possible partners for the communication. Using standard MERVA ESA queues and formats makes this exchange transparent to the customer application.

Functions of MERVA-MQI Attachment

MERVA-MQI Attachment sends and receives three groups of messages:

- SWIFT
- Telex
- User-defined messages

In order to handle these message groups, MERVA-MQI Attachment processes four MQI message types:

Datagram
A message that does not require a reply from the application that receives the message

Request
A message that requires a reply from the application that receives the message

Reply A message that replies to a previous request message

Report
: A message that informs an application about expected and unexpected events

A datagram or a request message is the envelope to transport a SWIFT, telex, or user-defined message. It is possible to add further fields to the envelope. These fields are clearly separated from the SWIFT, telex, or user-defined message. MERVA-MQI Attachment sends and receives datagrams and request messages.

A reply message contains application-specific data previously agreed upon between the sending and the receiving applications. An example of this kind of data is the MSGACK field for a SWIFT message. A reply message without application data also represents a valid reply. MERVA-MQI Attachment sends and receives reply messages.

A report message contains a feedback or reason code that is usually provided by an MQSeries queue manager or message channel agent. One or more report messages can be returned in response to a sent datagram, request message, or reply message. MERVA-MQI Attachment receives report messages.

The following report message types are supported:

Confirm on arrival (COA)
: Generated by the queue manager that owns the destination queue when the message is placed on that queue

Confirm on delivery (COD)
: Generated by the queue manager when an application retrieves the message from the queue in a way that causes the message to be deleted from the queue

Exception
: Generated by a message channel agent of MQSeries for MVS/ESA when a message is sent to another queue manager, but the message cannot be delivered to the specified destination queue

MERVA-MQI Attachment correlates the received reply and report messages with the previously sent message waiting for a response. After correlation of the original request message with the reply message, the application data of the reply message, if available, is accessible in the original message.

After correlation of the original message with one or more report messages, the feedback code is accessible in the original message. When both a COA and a COD report were received and correlated, the feedback code of the last correlated report is available. This is usually the feedback code of the COD report.

Activating MERVA-MQI Attachment

MERVA-MQI Attachment can be activated automatically or by operator commands.

At the message sending side it is started by MERVA ESA when a message is put to a MERVA ESA queue that acts as a send queue for MERVA-MQI Attachment. At the message receiving side it is triggered by the MQSeries queue manager when predetermined conditions on an MQI queue are satisfied that acts as a receive queue for MERVA-MQI Attachment.

The MERVA ESA operator can control MERVA-MQI Attachment by the operator commands CF (change function), SF (start function), and HF (hold function), which can be entered selecting the operator command function CMD or the MERVA system control function MSC, provided that the operator is authorized.

When using MQSeries for MVS/ESA, the SF command can enable triggering for an MQI receive queue for which triggering was disabled. This is only possible if MERVA-MQI Attachment itself disabled triggering due to an error detected during the processing of the messages in the queue. The MQSeries for MVS/ESA command ALTER QLOCAL has to be used outside MERVA ESA to disable triggering for an MQI queue for which triggering is enabled.

Components of MERVA-MQI Attachment

MERVA-MQI Attachment is made up of the following components:

- MERVA-to-MQI send process program
- MQI-to-MERVA receive process program
- Process table for customization
- User exit for send and receive process
- Conversion exit
- Routing table
- Message control blocks
- MERVA ESA resource definitions
- MQSeries resource definitions

MERVA-to-MQI Send Process Program

The send process program DSLKQS is the interface between MERVA ESA and MQSeries at the sending side of MERVA-MQI Attachment. DSLKQS performs the following steps when application messages are to be sent:

- Retrieves the MERVA ESA TUCB from CICS or IMS and determines the name of the function that started the program. The function can have a message queue or a dummy queue.
- Loads MERVA-MQI Attachment process table DSLKPROC and determines the single send process (message queue) or the send processes (dummy queue) the queue belongs to. For a dummy queue, DSLKQS also determines the send queues that are to be processed.
- Connects to the MQSeries queue manager (IMS and CICS/VSE only).
- Opens the MQI send and control queues.
- Retrieves a message from the current MERVA ESA send queue, calls a user exit (if customized), and writes the message to the MQI send queue and MERVA ESA control queue. This is repeated until the messages in the MQI send queue have to be committed. Before committing, MERVA-MQI Attachment writes control data to the MQI control queue.

  After committing, MERVA-MQI Attachment routes the messages from the MERVA ESA control queue to their target queues. The data in the MQI control queue is deleted.
- Retrieves the next message from the MERVA ESA send queue and repeats the message processing until the queue is empty. If there are other send queues or send processes, the retrieval continues until the last message of the last send queue was processed.
- Closes the MQI send and control queues.
- Disconnects from the MQSeries queue manager (IMS and CICS/VSE only).
- Releases the process table DSLKPROC.

MQI-to-MERVA Receive Process Program

The receive process program DSLKQR is the interface between MQSeries and MERVA ESA at the receiving side of MERVA-MQI Attachment. DSLKQR performs the following steps when application messages are received:

- Retrieves the message from CICS or IMS and determines whether it is a MERVA ESA TUCB or an MQSeries trigger message. Then it determines the name of the queue that started the program. The queue can be an MQI receive queue or a MERVA ESA dummy queue.
- Loads MERVA-MQI Attachment process table DSLKPROC and determines the single receive process (MQI receive queue) or the receive processes (MERVA ESA dummy queue) the queue belongs to. For a dummy queue, DSLKQR also determines the MQI receive queues that are to be processed.
- Connects to the MQSeries queue manager (IMS and CICS/VSE only).
- Opens the MQI receive queue.
- Retrieves a message from the current MQI receive queue, calls a user exit (if customized), and writes the message to the MERVA ESA control queue. This is repeated until the retrieved messages from the MQI receive queue have to be committed.

  After committing, MERVA-MQI Attachment retrieves the messages from the MERVA ESA control queue. A reply or report message is correlated with the waiting message of the appropriate send process. The correlated message is routed from the wait queue to its target queue. MERVA-MQI Attachment routes each retrieved message from the MERVA ESA control queue to its target queue.
- Retrieves the next message from the MQI receive queue and repeats the message processing until the queue is empty. If there are other MQI receive queues or receive processes, the retrieval continues until the last message of the last receive queue was processed.
- Closes the MQI receive queue.
- Disconnects from the MQSeries queue manager (IMS and CICS/VSE only).
- Releases the process table DSLKPROC.

Process Table

The process table DSLKPROC contains the definitions of all MERVA-to-MQI send processes and MQI-to-MERVA receive processes:

- A send process describes the queues, the formatting, and other control information required for MERVA-MQI Attachment to send messages from MERVA ESA to the MQSeries. Some examples of the definition parameters of a send process are:
  - The name of the send process
  - The names of the MERVA and MQI send queues
  - The names of the MERVA and MQI control queues
  - The names of various wait queues
  - The message format control information
  - The limit for an MQI commit
  - The MFS user exit number
  - The handling of certain MQI message types in the next processing step
  - The MQI message types written to the MERVA ESA journal
  - The output medium for, and the amount of, operator messages

- A receive process describes the queues, the formatting, and other control information required for MERVA-MQI Attachment to receive messages from the MQSeries and pass them to MERVA ESA. Some examples of the definition parameters of a receive process are:
  – The name of the receive process
  – The names of the MQI receive queues
  – The name of the MQSeries dead-letter queue
  – The name of the MERVA control queue
  – The wait interval for getting a message from an MQI receive queue
  – The message format control information
  – The limit for an MQI commit
  – The MFS user exit number
  – The MQI message types written to the MERVA ESA journal
  – The output medium for, and the amount of, operator messages

The send and receive processes group messages that have common message transfer characteristics.

The process table is defined by the DSLKPROC macro (see the *MERVA for ESA Macro Reference* for details). The name DSLKPROC of the process table cannot be changed. The sample process tables DSLKPSAM (for MVS) and DSLKPSMV (for VSE) show examples of definitions of send and receive processes.

User Exit for Send and Receive Processes

MERVA-MQI Attachment calls a user exit for a send process to do the following:

- Put additional fields to a datagram or request message. These fields are transmitted together with the SWIFT, telex, or user-defined message in the datagram or request message.
- Create the data for a reply message.

MERVA-MQI Attachment calls a user exit for a receive process to do the following:

- Put the additional fields of a received datagram or request message into their target fields in the SWIFT, telex, or user-defined message.
- Put the data of the reply message into their target fields in the SWIFT, telex, or user-defined message. The received reply message was already correlated with the original request message.

The user exit can be written in Assembler or in any of the high-level languages supported for a MERVA ESA MFS user exit:

- C/370
- COBOL
- PL/I

The following language-specific copy books and macros are available to access the data provided by MERVA-MQI Attachment:

DSLKCBLK	User exit control block and TOF field buffer. Available for Assembler, COBOL, and PL/I.
DSLKPROC	Process table entry. Available for Assembler (macro), COBOL, and PL/I.
DSLKEXIC	User exit control block, TOF field buffer, and process table entry. Available for C/370.

For more information about these copy books, see the *MERVA for ESA Customization Guide*.

The sample Assembler user exits DSLKQ001 and DSLKQ002 illustrate the described processing for a send and receive process.

Conversion Exit

MERVA-MQI Attachment itself does not make any conversion of the sent or received messages. Conversion means translating the character and numeric data of a message according to the needs of the message receiver. For example, a conversion from EBCDIC to ASCII or vice versa could become necessary.

MQSeries for MVS/ESA offers the data-conversion exit facility for message conversion. The data-conversion exit only converts messages consisting of characters **and** binary integer numeric data. When the data-conversion exit facility is enabled, MQSeries for MVS/ESA converts messages consisting of character data only. A data-conversion exit supports the conversion from or to double-byte character sets (DBCS).

The MQSeries for MVS/ESA sending message channel agent can invoke the data-conversion exit. The MQSeries for MVS/ESA queue manager invokes the data-conversion exit during the processing of an MQGET call (see the *MERVA for ESA Customization Guide* for details).

When the sending MCA invokes the data-conversion exit, the message data is converted before the message is sent to the recipient. This is required if the recipient is not able to convert the message.

When the MQSeries for MVS/ESA queue manager invokes the data-conversion exit, the message data is converted before the queue manager puts the message to an MQI queue. Usually the conversion is done at the receiving side.

Under CICS/ESA, the sample data-conversion exit DSLKCDCC applies when received messages are to be converted. The equivalent exit DSLKCDCM can be used both under IMS and when the distributed queue management without CICS is installed.

Under CICS/VSE, the sample conversion exit DSLKCVSE is provided that is always invoked by MERVA-MQI Attachment rather than by a component of MQSeries for VSE/ESA (see the *MERVA for ESA Customization Guide* for details).

The exit DSLKCVSE can convert both received messages and messages before they are sent. The exit converts messages consisting of character data only or of characters and binary integer numeric data. For character conversion it uses the sample conversion tables DSLKATOE (ASCII to EBCDIC) and DSLKETOA (EBCDIC to ASCII). The conversion exit only supports the conversion of single-byte character set (SBCS) data.

Routing Table

The sample routing table DSLKQRT shows how the MQI message types datagram, request, reply, and report message can be handled as MERVA messages and distributed among MERVA queues.

MERVA-MQI Attachment adds several new control fields to the message after it sent or received the message. The control fields contain the MQI message type, the name of the send and receive process, the message status, and other information (see the *MERVA for ESA Customization Guide* for details).

The routing table can be used for both the send and the receive process. It is associated with the MERVA control queue of a send or receive process.

Message Control Blocks

MERVA-MQI Attachment uses the following message control blocks (MCBs):

DSLKCOV	Cover MCB. Formats SWIFT, telex, or user-defined messages to the external network format. Displays control fields, additional message data, and reply message data. Operator command: SHOW KCOV or SHOW MQCOVER
DSLKCTL	Control messages. Displays the MQSeries control blocks MQMD (Message Descriptor), MQGMO (Get-Message Options), and MQPMO (Put-Message Options). The MFS editing exits DSLME910 and DSLME911 prepare the external representation of selected control block fields. Operator command: SHOW KCTL or SHOW MQCTL
DSLKDTA	Additional message data. Displays the additional fields for a SWIFT, telex, or user-defined message transmitted as an MQI datagram or request message. Operator command: SHOW KDTA or SHOW MQMORDAT
DSLKNPG	Page heading. Displays the new page heading during the MQSeries control block display. Operator command: SHOW KNPG or SHOW MQNEWPAG
DSLKRPL	Reply message data. Displays the MQI reply message data fields. Operator command: SHOW KRPL or SHOW MQREPLY
DSLKRPT	Report message data. Displays the MQI report message data (the first 100 characters of the message data from the original message). Operator command: SHOW KRPT or SHOW MQREPORT

MERVA ESA Resource Definitions

The required resources are defined in the following MERVA ESA tables:

DSLFDTT	Field definition table, copy book DSLFDTTC. Defines the fields for additional data of a datagram and request message, the reply message, and the MQI control blocks MQMD, MQGMO, and MQPMO.
DSLFNTT	Function table, copy book DSLFNTTC. Defines the send, control, various wait, and dummy queues.
DSLMPTT	MFS program table, copy book DSLMPTTC. Defines the MCBs, the MFS user exits, and the MFS editing exits.
DSLMSGT	Operator message table, copy book DSLKMSTC in DSLMSGTC. Defines the operator messages issued by MERVA-MQI Attachment.
DSLMTTT	Message type table, copy book DSLMTTTC. Defines the message types and message type synonyms of the MCBs.

MQSeries Resource Definitions

When using MQSeries for MVS/ESA, the copy books DSLKCSQC (for CICS) and DSLKCSQI (for IMS) contain the sample definitions of the MQSeries resources:

- Send, control, receive, and reply-to queues
- Processes for receive queues
- Sender and receiver channels

The appropriate copy book can be used as input for the MQSeries utility program CSQUTIL to define the resources. The MERVA ESA program directory contains a sample job for CSQUTIL. For more details see the *MQSeries for MVS/ESA System Management Guide*.

When using MQSeries for VSE/ESA, the master terminal transaction MQMT enables you to define the MQSeries resources:

- Send, control, receive, and reply-to queues
- Sender and receiver channels

The *MERVA for ESA Installation Guide* contains an example how to define local queues using the MQMT transaction. For more information, see the *MQSeries for VSE/ESA System Management Guide*.

Chapter 13. File, System, and Operator Services

This chapter describes additional services available in MERVA ESA for application programs.

Journal Service

The journal program DSLJRNP supervises and controls all the activities concerned with the MERVA ESA journal. This program is started by DSLNUC during the MERVA ESA startup.

The MERVA ESA journal program records:
- Successful commands and their responses
- Unsolicited operator messages
- Accesses to the user file
- Accesses to the authenticator-key file of SWIFT Link
- Program traces when requested
- Routing traces when requested
- Queue traces when requested
- Message traffic of communication network links

The journal record header part starts with the current date and includes the user key data. It is used as the key for the VSAM KSDS journal data set. The journal record header in MERVA ESA has a four-digit year. The format allows writing up to 1000 journal records per second and to create segmented journal records for larger buffers.

The complete description of the MERVA ESA journal records is found in “Appendix A. Journal Record Layouts” on page 185.

The MERVA ESA journal program supports two separate physical journal data sets. When the first data sets becomes full or unusable, MERVA ESA switches to the second journal data set and processing continues. If the second journal data set becomes full, MERVA ESA terminates.

It is possible to switch the journal data sets manually by using the operator command JSWITCH. If you specify the RESET parameter with this command, the data set that is switched to is cleared. Automatic switching is controlled by the journal switch status, which can be checked or changed using the JSET command. The switch status can have one of the following values:

ONCE	The journal is switched automatically the first time the switch is necessary; further switches are only possible via operator command.
MANUAL	No automatic switches occur.
CYCLE	The switch is performed back and forth as long as it is needed.

The JSTAT command allows the inspection of a table where all switch events were logged.

During initialization, the journal program opens the journal data set A. If the data set is empty, the journal program writes an initialization record to the journal file. If the journal data set A is full, processing is switched to data set B. An initialization record is written to journal data set B. If this request fails, or the journal data set B is also full, the journal initialization fails. If initialization is successful, a journal statistics message is issued indicating the number of records in the journal file and the percentage of space used in the allocated extents of the data set. If only primary space is allocated for the journal data set, the percentage numbers show how full the journal data set is. If secondary allocation is specified in the cluster definition, the number of allocated extents in the statistics message informs about this value. VSAM expands the space for a data set to a maximum of 123 extents.

The journal program DSLJRNP loads the exit program DSLJR001. The program DSLJR001 is available as a general user exit. If the module cannot be loaded, the customization parameter module DSLPRM specifies whether the journal program terminates or whether processing continues normally.

The following requests are processed by DSLJRNP:

PUT This request puts a new record into the journal file. The journal record identifier, the user key data, and the data itself must be supplied by the calling program. Date and time are supplied by the journal program. With each journal record written within one second, a counter is incremented by 1. It is appended to the time to ensure that each record has a unique VSAM key. The number of digits of this counter depends on the format of the journal record header used. The new format has a 3-digit counter for up to 1000 journal records written within one second.

When the JRNBUF parameter of the customization parameters DSLPRM specifies that segmentation of journal records is allowed, a record may be larger than the maximum record length of the VSAM KSDS. In this case the record is split into segments that will fit into the VSAM KSDS. All segments of a journal record have the same time stamp, which is extended by a 3-digit segment counter to force a unique key for each VSAM record.

GET This request gets an existing record from the journal file. The date, the time, and optionally the user key extension are supplied by the calling program as the search argument. The calling program can specify one of three options:

KEQ The key of the retrieved record must be equal to the key specified in the parameter list.

KGE The journal program returns the record with the next higher key if a record with the specified key is not available.

KGT The journal program returns the record with the next higher key.

The data of the record is passed to the calling program in the buffer. The journal record identifier, the date, the time, and the user key extension are passed to the calling program in the parameter list.

When a journal record is segmented, each get request will supply only one segment in the buffer. The journal record header contains the current segment number and the total number of segments of the record. The application program has to assemble the individual segments to build the complete record in the output buffer.

DSLJRNP indicates the completion status of the request by means of return and reason codes to the journal parameter list. These codes are listed in the *MERVA for ESA Messages and Codes*.

The journal program provides a user exit, DSLJR001, which is called at the end of each request. The input to the user exit is the address of the original journal parameter list in register 1 and the data buffer from the call in register 0. Using this exit program, a duplicate journal data set can be implemented.

DSLJR001 is loaded during the initialization of DSLJRNP. For more information on user exits, refer to the *MERVA for ESA Customization Guide*.

The MERVA ESA journal can be inspected online with the test command **journal** in the operator command function CMD or in the MERVA system control function MSC (see the *MERVA for ESA Installation Guide* for details).

User File Service

The user file contains one record for each MERVA ESA user. Each record determines which functions and message types a user is authorized to use, and any command restrictions that may apply. The record contains control information, such as password and origin ID, and also specifies the user's preferred environment: language, NOPROMPT format, network format, PF keys, and so on. See the *MERVA for ESA User's Guide* for details.

The MERVA ESA user file service is a MERVA ESA central service carried out by DSLNUSR under the control of the nucleus program DSLNUC. The end-user driver DSLEUD requests user file services via the MERVA ESA intertask communication.

The MERVA ESA user file is a keyed VSAM file. The key is the user ID. The data in the file is encrypted. This method is selected by specifying EXUMASK=YES in the MERVA ESA customizing module DSLPRM.

Note: There is a customizable option in MERVA ESA that allows you to apply an additional and more secure encryption algorithm to all modified user file records. When this option is selected, a modified user file record cannot be used by MERVA/ESA V3.1 and earlier.

Records are added to the user file using the online maintenance function of DSLEUD. The records are used to control the user's activities during an end-user driver session.

DSLNUSR provides access to the user file and carries out the following functions:

- Initialization.

  DSLNTS calls DSLNUSR for initialization during the MERVA ESA startup. DSLNUSR opens the user file and allocates the active user table.

  The user exits DSLNU003, DSLNU004, and DSLNU005 are loaded. When loading of one of these programs fails, processing continues, but the user exit is not called during later processing. For more information on user exits refer to the *MERVA for ESA Customization Guide*.
- Processing end-user session control.

  During end-user sessions, DSLEUD requests the following services from DSLNUSR:

  - Signon.

    The user record is read. If the record can be used for signon, the password is checked to see if it is valid. The user exit DSLNU003 is called at this stage to carry out optional additional authorizations. The user file record is returned to the caller. The active user table is updated, and the signon request is journaled. The last sign-on date is updated in the user file record. A unique signon sequence number is returned to the caller. This number is required in all subsequent requests. If the number does not match, it is assumed that the signoff of the user has been forced. For more information on user exits refer to the *MERVA for ESA Customization Guide*.
  - Function Selection.

    The function entered on a panel is checked against the allowed functions to determine whether the request is valid. The function-table entry is returned to the caller. The function selection is journaled. The table of active users is updated.
  - Signoff.

    The active user table is updated. The signoff request is journaled.
  - Password Check or Change.

    The password is checked to see if it is the current password of the user. If a password change is requested, the user’s record is updated. The request is journaled.

    The user exit DSLNU005 is called to check the password of the user. The parameter list containing the entered password, and the user file record containing the current password of the user, are passed to the routine. Both passwords are scrambled and can be compared by the user exit. A routine for unscrambling the password is also provided to allow password checks against external resources. When the user exit is not available, the passwords are compared by DSLNUSR, and processing continues according to the result. For more information on user exits refer to the *MERVA for ESA Customization Guide*.
  - Function Retrieval.

    All functions defined for the user are provided together with function-related information from the function table (for example, the functions’ descriptions) when the Function Selection Panel is displayed. The function retrieval request is journaled.
- Processing user file maintenance.

  The user file maintenance is performed by the end-user function program DSLEUSR. DSLNUSR supports the following requests for online maintenance:
  - Display a user record
  - Add a user record
  - Delete a user record
  - Replace a user record after a change
  - List user records

  The access to one user record is limited to one user at a time. All maintenance requests are journaled.

  DSLNUSR can perform extended authorization checks on all user file maintenance requests except LIST:

  Extended origin ID checking

  The following rules apply:

  - The request is accepted only (1) if the first 8 bytes of the origin ID of the maintenance user match the first 8 bytes of the origin ID in the requested user file record, or (2) if the maintenance user is a master user.
  - Even if the first 8 bytes of the origin IDs match, if an unauthorized user attempts to add or modify the record of an authorized MERVA ESA administrator, the request is rejected. However, an unauthorized user can display this record.

  Extended origin ID checking is performed if the parameter EXUSR=(YES,ORIGINID) is specified in the customizing module DSLPRM.

Extended group ID checking
  The following rules apply:
  - The request is accepted only (1) if the group ID of the maintenance user matches the group ID in the requested user file record, or (2) if the maintenance user is a master user.
  - Even if the group IDs match, if an unauthorized user attempts to add or modify the record of an authorized MERVA ESA administrator, the request is rejected. However, an unauthorized user can display this record.

  Extended group ID checking is performed if the parameter EXUSR=(YES,GROUPID) is specified in the customizing module DSLPRM.

Checking origin and group IDs is useful when user file records for more than one financial institution are maintained in the same MERVA ESA user file. It prevents the deletion or modification of the records of one financial institution's user file by a maintenance user of another financial institution. For each of these requests the user exit DSLNU004 is called, which can carry out an additional authorization. An individual request may be rejected by the user exit depending on the current status or the user profile. For more information on user exits refer to the *MERVA for ESA Customization Guide*.

- Termination.

  DSLNTS calls DSLNUSR for termination during the MERVA ESA termination. DSLNUSR closes the user file and releases the storage of the active user table. The user exit programs are deleted.

For all processing requests, DSLNUSR provides one of the following:
- The user file record and the function table entry
- The data for a list of users

The operator commands **du** and **force** are also serviced by DSLNUSR. The **force** command only sets an indicator into the active user table, but does not remove the entry from the active user table. When this indicator is set, the user is signed off automatically if a user file service request is issued. When a signon request for this user is received, the request is fulfilled and the active user table entry is reused but a new signon sequence number is assigned.

Authentication Service of SWIFT Link

Many SWIFT financial messages require authentication to determine if they have been changed during their transmission from one destination to another, and to show to the receiver that the sender was authorized to send the message. In the customizing parameter module DWSPRM you can specify whether SWIFT Link performs authentication of SWIFT messages.

The authentication service has two components:

- The maintenance of the authenticator-key file, using authenticator keys exchanged on paper via the public mail system, or using the SWIFT bilateral key exchange (BKE) over the SWIFT network
- The authentication of SWIFT messages

These functions are performed by the authenticator-key file program DWSAUTP, and, for the BKE, partly by the SWIFT user security enhancement (USE) functions running on a USE workstation.

The authenticator-key file utility DWSAUTLD calls DWSAUTP directly for the maintenance of the authenticator-key file.

When MERVA ESA is running, DWSAUTP is controlled by DSLNUC.

DWSAUTP can be initialized by the following programs defined in the Nucleus Program Table DSLNPTT:

- The general purpose application program DWSDGPA with the descriptive name SWIFTII
- The special program DWSAUTIN with the descriptive name SWIFTAUT (DWSAUTIN allows for initializing DWSAUTP without starting the connection to the SWIFT network)

The MERVA ESA application programs invoke DWSAUTP as follows:

- The programs link-edited to DSLNUC directly with the DWSAUT macro. For example, the program DWSDGPA calls DWSAUTP directly for authentication of SWIFT messages.
- The programs not link-edited to DSLNUC via the MERVA ESA intertask communication. The task server DSLNTS invokes DWSAUTP as a central service, and the DWSAUT macro is used by the requestor to prepare the parameter list for the request. For example, the SWIFT Link program DWSEAUT calls DWSAUTP for the online maintenance of the authenticator-key file, and for authentication of SWIFT output messages with the user command **authent** (see the *MERVA for ESA User's Guide* for details).

The DWSAUT macro is described in the *MERVA for ESA Macro Reference*.

Maintenance of the Authenticator-Key File

All authenticator keys of a MERVA ESA installation are contained in one authenticator-key file. If the SWIFT bilateral key exchange (BKE) is used, a copy of the authenticator-key file is contained in the USE workstation.

When several financial institutions share one MERVA ESA installation, each of them may have its own sets of authenticator keys, or they may share authenticator keys according to the rules defined by SWIFT.

The authenticator-key file is a VSAM key-sequenced data set (KSDS). The records of this data set are enciphered for security reasons.

The authenticator-key file can be maintained off-line with the utility DWSAUTLD or online with the authenticator-key file maintenance program DWSEAUT.

If BKE is used, the authenticator keys are maintained by the BKE process running on the USE workstation using the SWIFT Link of MERVA ESA. After a BKE is complete or on operator request, an update is sent from the USE workstation to MERVA ESA. The transaction program DWSAUTT receives the update and invokes DWSAUTP for updating the authenticator-key file.

DWSAUTP and DWSEAUT ensure that only authorized users can maintain authenticator-key file records, and one user can only maintain the records for the financial institution indicated by the Origin ID of the user file record.

Online maintenance can be done in two ways:

1. As two steps by two different users, the first one making the changes (add, replace, or delete), and the second one authorizing the changes.
2. One user making these two steps, that is, the changes are immediately authorized.

Refer to the *MERVA for ESA User's Guide* for details.

Authentication of SWIFT Messages

The input to DWSAUTP is the message to be authenticated in the format in which it is transmitted to or received from the SWIFT network, and the authenticator key from the authenticator-key file or, if BKE is not used, supplied in the calling parameter list after the user command **authent**.

Authentication is carried out when:

- The MERVA ESA message type table (containing the specifications for SWIFT message types) shows that a specific message type must be authenticated.
- The message contains an authentication trailer component (MAC). This allows a user to request authentication of a SWIFT input message for which authentication is not specified in the message type table by just adding an authentication trailer with a dummy authenticator to the message.

If authentication is required, the authenticator-key file record relevant for a particular correspondent relationship is retrieved. For better performance, frequently used records are held in main storage.

DWSAUTP decides which key to use:

- The sending key for an input message. The key is used that is current on the sending date.
- The receiving key for an output message. The key is used that was current on the sending date.

The authentication algorithm calculates the authenticator.

Processing continues:

- For a SWIFT input message by providing the authentication trailer with the authenticator.

- For a SWIFT output message by comparing the authenticator in the authentication trailer with the calculated authenticator. If this comparison fails and the authentication was tried with the key from the file, a second or third try is done with the other two keys from the file (if available) that were not current on the sending day.

The success or failure of authentication is indicated by a reason code and a diagnostic message.

General File Service

Product-Sensitive Programming Interface

MERVA ESA can process (for example read from and write to) files that you have defined.

- For MERVA ESA running under CICS, these files must be VSAM key sequenced data sets (KSDS) with fixed record length. No alternate index is allowed. These files must also be defined in the CICS file definitions.
- For MERVA ESA running under IMS, these files must be HISAM databases (DBs) of DL/I with one root segment of fixed length only. A secondary index DB is not allowed. For the database, a VSAM KSDS cluster must be defined whose attributes can be derived from the generation of the database for IMS.

  These files must also be defined in the IMS nucleus. Programs using the file must have the database PCB included in their IMS PSB.

Whenever a file is to use the MERVA ESA general file service, it must be defined to MERVA ESA in the file table DSLFLTT with the DSLFLT macro. Such files can then use the following services of MERVA ESA:

- Initialization of the file using the file utility DSLFLUT
- Listing records that are in the file using the file utility DSLFLUT
- Access to the file in MERVA ESA application programs and user exits using the file service program DSLFLVP
- Online maintenance of the file by the MERVA ESA users
- Expansion function of the MERVA ESA message format services, for example for the expansion of the SWIFT addresses (BICs) to correspondents' names

The MERVA ESA general files can be shared or nonshared files.

- Shared file: This is one physical file that is divided logically into several files of different owners. The record key is preceded by an 8-byte owner prefix.
  - Private records are owned by a specific user. The owner prefix contains the user ID of the owning user. Only this user can process the private records during the online maintenance functions of MERVA ESA.
  - Common records are not owned by a specific user. The owner prefix contains an asterisk (*) indicating the common ownership. Any user who is allowed to perform online file maintenance can process the common records.
- Nonshared file: This is one physical file for just one purpose. Any user who is allowed to perform online file maintenance can process the records of the nonshared file.

A file can be used as a nickname file:that is, the real search argument of another file can be replaced by an "easier" search argument of the nickname file.

For example, the 8 or 11 characters of a SWIFT address can be referred to in the MERVA ESA nicknames file. A nickname can be entered in a message field instead of the SWIFT address, and the SWIFT Link expansion routine DWSMX001 uses the nickname to read the appropriate record from the MERVA ESA nicknames file to get the SWIFT address, which then replaces the nickname in the message field. It then uses the SWIFT address to get the correspondent's name from the SWIFT correspondents file.

When nicknames are to be used for the address expansion of SWIFT Link, the MERVA ESA nicknames file must be filled by the user, and the nickname expansion must be defined in the relevant function table entry.

Nickname files can be nonshared, or shared with private and common records.

For details on defining names expansion in a function, see the description of the EXPAND and EXPNAM parameters of the DSLFNT macro in the *MERVA for ESA Macro Reference*.

MERVA ESA supplies an example of a Nicknames File, which is a shared file. The record layout is described in "Appendix B. Layout of the MERVA ESA Nicknames File" on page 195, and in the copy code DSLCORN.

SWIFT Link supplies the SWIFT Correspondents File, which is a nonshared file.

File Service Program DSLFLVP

The MERVA ESA file service program DSLFLVP can access files, following the MERVA ESA general file concept, from programs running in a MERVA ESA environment. DSLFLVP is invoked by means of the DSLFLV macro. The *MERVA for ESA Macro Reference* explains this macro and its parameters.

The following general file services are available:
- Open a file
- Close a file
- Add a record
- Delete a record
- Replace a record
- Get a record by direct access (get with key)
- Get a record by sequential access

When a MERVA ESA application program uses DSLFLVP to access the general MERVA ESA files, it need not know if it is operating in the environment of a batch program, a CICS environment, or an IMS environment.

Calling the program DSLFLVP requires the use of a parameter list (PL) and a request control block (RCB), which can be generated by the DSLFLV macro.

End of Product-Sensitive Programming Interface

System Services

For many of the services of the operating system or CICS, MERVA ESA programs invoke the service program DSLSRVP. The MERVA ESA program then need not know if it is running under CICS, under IMS, or in a batch environment. The services provided are:

- Obtain and initialize main storage
- Release main storage
- Load a program
- Delete a loaded program
- Request current date and time of day
- Post an event control block
- Wait for an event to occur
- Request exclusive control of a resource
- Release exclusive control of a resource
- Dump main storage

DSLSRVP is invoked by the DSLSRV macro. The *MERVA for ESA Macro Reference* explains this macro and its parameters. The *MERVA for ESA Customization Guide* shows examples of how to use this macro.

Depending on the environment, DSLSRVP uses VSE or MVS macros or CICS commands. CICS commands are used if the field COMEIB in the MERVA ESA communication area (DSLCOM) contains an address.

The parameter list for calling DSLSRVP is contained in DSLCOM.

Operator Command Service

The operator command service performs the execution of MERVA ESA operator commands. The MERVA ESA command server (DSLNCS) is the interface between MERVA ESA operators and the command execution routines. All MERVA ESA operator commands are defined in the MERVA ESA operator Command Table DSLNCMT, and all command responses or their basic structures are defined in the MERVA ESA operator message table (DSLMSGT).

DSLNCS is called only by the programs link-edited to DSLNUC. Therefore the MERVA ESA application programs invoke DSLNCS as follows:
- The programs link-edited to DSLNUC directly.
- The programs not link-edited to DSLNUC via the MERVA ESA intertask communication. The task server DSLNTS invokes DSLNCS as a central service.

The requesting program must provide status information and the command in a MERVA ESA command and response buffer, and DSLNCS adds the command response. This buffer is defined with a DSLNMO MF=BUF macro (see the *MERVA for ESA Macro Reference* for details).

DSLNCS carries out the following steps for command execution:
- The command table DSLNCMT is searched to find the command. If found, the command input is checked to see that it is formally correct.
- The user exit DSLNCU01 is called to check if the operator is authorized to use this command. An error is returned if authorization is not confirmed.
- Control is given to the command execution routine indicated in DSLNCMT. The interface for command execution routines is described in the *MERVA for ESA Customization Guide*.
- After return from the command execution routine, the command response is journaled, or error responses are prepared by DSLNCS depending on the return code from the command execution routine.

Operator and Diagnostic Message Services

Product-Sensitive Programming Interface

The operator and diagnostic messages services handle the messages that are given to the users and operators of MERVA ESA programs:

- Unsolicited messages to operators when running MERVA ESA, its batch programs and utilities
- Response messages for operator and user commands
- Diagnostic and error messages of MFS services, for example, checking

The services for these messages include:

- The definition of the messages or message skeletons in the Message Table DSLMSGT
- The retrieval of the message or the skeleton from the Message Table using the program DSLOMSG
- Issuing the message

The Message Table DSLMSGT

The Message Table DSLMSGT defines all messages that are issued to MERVA ESA operators and users. The DSLMSG macro is used to generate this table. The *MERVA for ESA Macro Reference* explains this macro and its parameters. The *MERVA for ESA Customization Guide* shows examples of how to use this macro.

The DSLMSG macro allows for defining:

- Variable information in the message that is inserted by the message retrieval program DSLOMSG
- Several national languages for the same message

MERVA ESA supplies the sample Message Table DSLMSGT that contains all messages used by MERVA ESA and its components.

The Message Retrieval Program DSLOMSG

The message retrieval program DSLOMSG is used to retrieve operator and diagnostic messages from the Message Table DSLMSGT. During the retrieval process, variable information can be inserted into the message by DSLOMSG.

The DSLOMS macro is used to invoke DSLOMSG. The *MERVA for ESA Macro Reference* explains this macro and its parameters. The *MERVA for ESA Customization Guide* shows examples of how to use this macro.

In the calling parameter list for DSLOMSG, the DSLOMS macro defines the variable fields to insert in the messages. The calling program provides the data for these fields to insert into the messages.

Issuing Operator and Diagnostic Messages

These messages are issued as follows:

- The diagnostic and error messages for display on screen and printer devices are written to the internal message buffer (TOF).
- Unsolicited operator messages are issued using:

- The operator interface program DSLNMOP (refer to “The Operator Interface Program (DSLNMOP)” on page 34 for details). DSLNMOP is invoked using the DSLNMO macro. The *MERVA for ESA Macro Reference* explains this macro and its parameters. The *MERVA for ESA Customization Guide* shows examples of how to use this macro.
- The write-to-operator program DSLWTOP (refer to “The Write-to-Operator Program (DSLWTOP)” on page 35 for details). DSLWTOP is invoked using the DSLWTO macro. The *MERVA for ESA Macro Reference* explains this macro and its parameters. The *MERVA for ESA Customization Guide* shows examples of how to use this macro.

End of Product-Sensitive Programming Interface

Chapter 14. MERVA ESA Data Sets and Utilities

This chapter describes the MERVA ESA data sets and utilities.

Data Sets

The following describes the data sets required for the operation of MERVA ESA and the network links. MERVA ESA uses the following data sets:

- Journal data set
- Queue data set
- Large message cluster
- User file
- Nicknames file
- Message counter log data set
- SPA File (IMS only)
- Authenticator-key file
- SWIFT correspondents file
- SWIFT currency code file
- Telex correspondents file

The SPA file can be a BDAM data set or an HDAM database. All other MERVA ESA data sets are VSAM data sets. They are key-sequenced organized (KSDS) except the queue data set, which is relative-record organized (RRDS).

Journal Data Sets

The journal data sets are controlled by the journal program DSLJRNP (see “Journal Service” on page 161 for details).

The information provided in the journal data sets can be the basis for the off-line preparation of various kinds of overviews and statistics for different administrative and operational purposes. The layout of the journal records is shown in “Appendix A. Journal Record Layouts” on page 185. The journal record header of MERVA ESA contains a 4-digit year field. A 2-digit year field is not supported.

The contents of the journal data sets can be viewed online with the **journal** command in the operator command function CMD or in the MERVA system control function MSC.

Queue Data Sets

All messages in the MERVA ESA queues are stored in the MERVA ESA queue data set (QDS). The data of large messages is stored in the large message cluster described in the next section. For added assurance, a duplicate QDS can be maintained. The queue data sets are controlled by the queue management program DSLQMGT (see “Queue Management Program DSLQMGT” on page 72 for details).

The *MERVA for ESA Installation Guide* gives details about how to calculate the space required for the queue data sets.

The current usage can be seen with the **dq** (display queues) and **dq status** commands in the operator command function (CMD) or the system control function (MSC).

Large Message Cluster

The data of large messages in the MERVA ESA queues is stored in the MERVA ESA large message cluster (LMC). For added assurance, hardware duplication can be used, for example, the Dual Copy of the IBM 3390. The large message cluster is controlled by the program DSLQLRG (see "Large Message Service Program DSLQLRG" on page 84 for details).

The *MERVA for ESA Installation Guide* gives details about how to calculate the space required for the large message cluster.

The current usage can be seen with the **dlmc** (display LMC status) and **dlmct** (display LMC tuning information) commands in the operator command function (CMD) or the system control function (MSC).

User File

The user file contains one record for each MERVA ESA end user. The record defines which functions and message types a user is authorized to use, and any command restrictions that may apply. It contains records of fixed length. The user ID serves as the key for accessing the user file. The user file records are scrambled to prevent reading by unauthorized persons.

The user file is controlled by the user file program DSLNUSR (see "User File Service" on page 163 for details).

The user file is maintained online with the end user functions USRx.

Nicknames File

The Nicknames File is a sample file supplied with MERVA ESA to show the use of the MERVA ESA general file concept. It is defined as a shared file, and it can contain private nickname records and common nickname records. (See "General File Service" on page 168 for details). The records are added and maintained using the general file maintenance of end users in the FLM function.

MERVA ESA provides sample definitions for the Nicknames File for the MERVA ESA File table in the copy code DSLFLTTC.

The record layout of the Nicknames File is described in the copy code DSLCORN and shown in "Appendix B. Layout of the MERVA ESA Nicknames File" on page 195.

The Nicknames File can be printed with the utility DSLFLUT using the message control block DSL0CORN.

Message Counter Log Data Set

| This data set, which is controlled by the program DSLCNTP, is used to record
| statistics regarding the message traffic handled by MERVA ESA on:

- | External network links
- | Internal links using MERVA Link or MERVA-MQI Attachment with FMT/ESA

If the usage exceeds the maximum amount specified in the customization module (DSLPRM), MERVA ESA alerts the system administrator.

Use the **dclog** operator command to view online the total or average usage per month for the previous one to 12 months.

SPA File

The SPA file is used only in IMS. Its purpose is to allow a scratch pad area (SPA) for the MERVA ESA End-User Driver, which is bigger than the 32768 bytes (32KB) and therefore cannot be handled by IMS. For the End-User Driver, the IMS SPA is only 320 bytes, and all other storage is saved in the MERVA ESA SPA file. The scratch pad area saves the permanent storage for a user session between two conversation steps.

The SPA-file initialization program (DSLEBSPA) must be run before starting MERVA ESA for the first time (see "SPA File Initialization Program" on page 180).

The MERVA ESA SPA file can handle up to 96KB of permanent storage plus up to 2MB for processing of large messages for each user session. The file consists of an index record and data records containing the user-session data. The basic direct access method (BDAM) is used. When running the End-User Driver in more than one MPP region, the same SPA file is assigned to each MPP region with DISP=SHR.

For installations that want to implement their own SPA file, the macro DSLEISPA defines the interface between DSLEUD and the online SPA file program. Refer to the *MERVA for ESA Customization Guide* for further information. MERVA ESA V3.2 provides the sample program DSLEOSPB that uses an IMS HDAM database instead of a BDAM data set. This program can be installed instead of DSLEOSPA. Refer to the *MERVA for ESA Customization Guide* and *MERVA for ESA Installation Guide* for further information.

Authenticator-Key File

The authenticator-key file holds the authenticator keys of all financial institutions sharing one MERVA ESA installation. The authenticator keys are used for the authentication of SWIFT messages.

The records in the data set are enciphered. The authenticator keys are stored in these records.

The number of authenticator keys to be stored determines the size of the data set. The *MERVA for ESA Installation Guide* gives details about how to calculate the space for the authenticator-key file.

SWIFT Correspondents File

The SWIFT Correspondents File contains a record for each SWIFT correspondent. The records of the file are created from the SWIFT Bank Identifier Code (BIC) Directory update tape using the utility DWSCORUT. They are used for the expansion of SWIFT addresses (BICs) to full addresses.

The SWIFT Correspondents File is a nonshared file using the MERVA ESA general file concept (see "General File Service" on page 168 for details). The file can be maintained using the general file maintenance of end users in the FLM function.

MERVA ESA provides sample definitions for the SWIFT Correspondents File for the MERVA ESA File Table in the copy code DWSFLTTC.

The record layout of the SWIFT Correspondents File is described in the copy code DWSCOR and shown in “Appendix D. Layout of the SWIFT Correspondents File” on page 199.

The SWIFT Correspondents File can be printed with the utility DSLFLUT using the message control block DWSSCOR.

SWIFT Currency Code File

The SWIFT currency code file contains a record for each currency code. The records of the file are created from the SWIFT Bank Identifier Code (BIC) Directory update tape using the utility DWSCURUT. They can be used for checking the currency codes in a SWIFT message.

The SWIFT currency code file is a nonshared file using the MERVA ESA general file concept (see “General File Service” on page 168 for details). The file can be maintained using the general file maintenance of end users in the FLM function.

MERVA ESA provides sample definitions for the SWIFT currency code file for the MERVA ESA File Table in the copy code DWSFLTTC.

The record layout of the SWIFT currency code file is described in the copy code DWSCUR and shown in “Appendix C. Layout of the Currency Code File” on page 197.

The SWIFT currency code file can be printed with the utility DSLFLUT using the message control block DWSSCUR.

Telex Correspondents File

The Telex Correspondents File contains a record for each telex correspondent. It is used for telex address expansion.

The file is a nonshared file using the MERVA ESA general file concept (see “General File Service” on page 168 for details). Its records are created and maintained using the general file maintenance of end users in the FLM function.

MERVA ESA provides sample definitions for the Telex Correspondents File for the MERVA ESA File table in the copy code ENLFLTTC.

The record layout of the Telex Correspondents File is shown in “Appendix E. Layout of the Telex Correspondents File” on page 201.

The Telex Correspondents File can be printed with the utility DSLFLUT using the message control block ENLTCOR.

Utilities

The utilities provided by MERVA ESA are:

- Queue data set utility
- Large message cluster maintenance utility
- General file utility
- Message counter report utility

- SPA file initialization utility (IMS only)
- Authenticator-key file utility
- SWIFT correspondents file utility
- SWIFT currency code file utility

The *MERVA for ESA Operations Guide* describes in detail how to use all MERVA ESA utilities.

Queue Data Set Utility

The queue data set (QDS) utility DSLQDSUT processes the QDS as follows:

1. Formats the queue data set
2. Copies the queue data set
3. Modifies the queue data set
4. Sets the last unique message reference (UMR)

Formatting removes all messages from the QDS; modifying keeps them, but causes a DSLQMGT restart during the next MERVA ESA startup.

If your MERVA ESA installation uses the unique message reference (UMR) option, the FORMAT and MODIFY functions of DSLQDSUT can be used to set or adjust the last assigned UMR.

The *MERVA for ESA Operations Guide* describes in detail how to use DSLQDSUT. The formula to calculate the space required for the QDS is given in the *MERVA for ESA Installation Guide*.

Formatting a Queue Data Set

Formatting is necessary after the VSAM cluster definition of the QDS, before MERVA ESA is started the first time.

Formatting an existing QDS removes all messages from this QDS. Formatting prepares the QDS as follows:

- The first block is reserved for the QDS log record.
- The second and third blocks are reserved for the QDS byte map for storage allocation in the data blocks.
- The following blocks are for the queue key tables. The number of blocks is determined by the space needed for the queue key tables. This space depends on the NQE parameter of the MERVA ESA customizing parameters, and on the key lengths specified in the function table for the various queues.
- The rest of the QDS is filled with empty data blocks (cleared to binary zeros), each block having a block header indicating the block number, the amount of free space, and the offset to the free space. Data blocks are written to the QDS until VSAM returns the return code for no more space in the data set. Then the first block is updated with the number of available data blocks for use by DSLQMGT.

After successful formatting, the MERVA ESA operator receives a message stating the number of blocks available in the QDS for storing messages.

When duplicate queue data sets are used, both queue data sets must be formatted. After formatting the first queue data set, the DSLQDSUT COPY function or the VSAM Repro function can be used to produce a duplicate of the formatted queue data set.

Copying a Queue Data Set

Copying is recommended for the following reasons:

- To make a backup copy of the QDS
- To make a copy of the first QDS after formatting when using duplicate QDSs
- To make a copy of the error-free QDS after an error in the duplicate one

The output queue data set must have the same space allocation as the input data set. If the output queue data set is smaller, the COPY function cannot complete successfully. If the output queue data set is larger, only as much space is used as in the input queue data set, and the rest of the space cannot be accessed.

Modifying a Queue Data Set

Modifying a queue data set is recommended in the following cases:

- After the NQE parameter of the MERVA ESA customizing parameters has been increased or decreased, the QDS blocks reserved for the queue key tables may not be enough (increase of NQE), or space may be wasted (decrease of NQE). In both cases, the DSLQMGT remains operational, but if the actual queue key table entries do not fit into the system blocks at DSLQMGT termination, a DSLQMGT restart is necessary to build the correct queue key tables during the next DSLQMGT initialization. Modifying the QDS ensures enough system blocks.
- If key lengths in the MERVA ESA function table are changed, the same applies as for NQE.
- If message-processing functions are added to the MERVA ESA function table, and the keys of the new functions are longer than the longest key used before, the same applies as for increasing NQE.
- If message-processing functions are removed from the MERVA ESA function table, and there are still messages in these function queues.

  The EXCLUDE FNT control statement indicates to DSLQDSUT to not copy messages of queues that are not in the function table.
- If large messages that are not in the LMC are referenced in the QDS.

  The EXCLUDE LMC control statement indicates to DSLQDSUT to not copy message references to large messages that are not in the large message cluster.
- If MERVA ESA terminated with error message DSL359A due to corrupted queue element prefix.

  The REPAIR control statement indicates to DSLQDSUT to fix corrupted QE prefixes.
- To adjust the last unique message reference (UMR).

Modifying a queue data set is necessary in the following cases:

- When the old queue data set is full, and you want to allocate a larger one and keep all messages contained in the old queue data set.
- When the old queue data set is too big, and you want to allocate a smaller one to reduce the number of empty blocks, but keep all messages contained in the old queue data set.
- When the old queue data set has one or more corrupted blocks, and you want to keep the messages that are not corrupted.

The MODIFY function copies messages from the old queue data set (input) to the new queue data set (output). If a corrupted QDS block is found, all complete messages are copied. Corrupted QDS blocks can be excluded completely using

EXCLUDE *rbn* control statements. After the input queue data set has been completely processed, the rest of the output queue data set is formatted as in the FORMAT function.

The output queue data set can have a space allocation different from the input data set. If the output queue data set is smaller than the input queue data set, the MODIFY function completes successfully only if all messages (except the ones excluded by EXCLUDE FNT or EXCLUDE *rbn* or EXCLUDE LMC control statements) can be copied into the output queue data set.

When MERVA ESA is started with a queue data set created by the MODIFY function, a DSLQMGT restart is performed to create a new queue key table. This DSLQMGT restart can be shorter than the usual restart as DSLQDSUT tells DSLQMGT which data blocks contain messages.

Large Message Cluster (LMC) Maintenance Utility

The large message cluster (LMC) maintenance utility DSLQMNT can be used to reorganize the input LMC into an output LMC by copying the LMC records in ascending key sequence into the output LMC. To ensure for data integrity, DSLQMNT also verifies that all LMC records are referenced by a queue element in the queue data set. LMC records without a corresponding queue element are not copied.

The reorganization provides a contiguous free space from the end of the last record to the end of the output LMC.

A status and statistics report is provided for the input LMC and the output LMC. It also provides a detailed list of all large messages referenced in the QDS but not found in the LMC.

The utility should be run when the free space can not be used for storing the data of a large message because there is not enough contiguous free space. If this condition happens frequently, it may be necessary to allocate a larger LMC.

General File Utility

MERVA ESA provides the general file utility DSLFLUT to process files that have been defined in the MERVA ESA File Table. Two functions are available:

- Initialize a file
- List the records of a file (that is, the whole file or only selected groups of records).

Control statements determine the processing of DSLFLUT. They are specified in a sequential input data set that may be included in the startup job. At least two control statements are required:one specifying the requested function, and one specifying the file name. The *MERVA for ESA Operations Guide* describes in detail how to use DSLFLUT.

Message Counter Report Utility

| MERVA ESA provides a utility called DSLCNTUT that lets you create monthly
| reports that show the message traffic handled by MERVA ESA on:

- | External network links
- | Internal links using MERVA Link or MERVA-MQI Attachment with FMT/ESA

A report shows the total and average usage per month for the previous one to 12 months. If the usage exceeds the maximum amount specified in the customization module (DSLPRM), MERVA ESA alerts the system administrator. You should print one report each month, and retain copies of these reports for future reference.

SPA File Initialization Program

In IMS, the SPA-file initialization program (DSLEBSPA) is used to initialize the MERVA ESA SPA file before it is used the first time. DSLEBSPA must not be run when MERVA ESA is active.

DSLEBSPA formats the SPA file by writing the index record and empty session records. Details about running the SPA-file initialization program can be found in the *MERVA for ESA Installation Guide* and *MERVA for ESA Operations Guide*.

Authenticator-Key File Utility

The authenticator-key file utility DWSAUTLD maintains the authenticator-key file of SWIFT Link off-line.

If SWIFT bilateral key exchange (BKE) is used, it is recommended to use the maintenance functions of the USE workstation as much as possible, and not DWSAUTLD.

DWSAUTLD is used to:

- Initialize the authenticator-key file with a secure transmission key for enciphering if BKE is used.
- Maintain the authenticator-key file by:
  - Adding records
  - Replacing existing records
  - Exchanging the keys in one record or a set of records depending on the change dates in the records
  - Deleting existing records
  - Listing the contents of records

  DWSAUTLD always creates authorized records in the file. It is not possible to create unauthorized records as during the online maintenance described in the *MERVA for ESA User's Guide*.
- Unload all or part of the authenticator-key file to a sequential data set when saving the file, changing its size, or changing the secure transmission key for enciphering. The sequential file is scrambled.
- Reload the complete authenticator-key file, or a part of it, from a sequential data set created by the unload operation. The records can be reloaded into a new file (especially after space problems in the old file), or the records of the sequential data set can be merged with records already in the file.

If several financial institutions share one MERVA ESA installation, the online maintenance allows one user to process only the authenticator keys of one financial institution. DWSAUTLD can maintain the authenticator keys of all these institutions, or of a single institution depending on the input records that control the processing of DWSAUTLD. These input records are described in detail in the *MERVA for ESA Operations Guide*.

DWSAUTLD uses the authentication program DWSAUTP for accessing the authenticator-key file.

According to the SWIFT specifications, DWSAUTLD supports:

- Small keys consisting of 16 unique hexadecimal characters
- Large keys consisting of 32 hexadecimal characters in any combination

The authenticator-key file can hold both types of keys.

When using BKE, it is recommended to use the unload file of MERVA ESA to create an authenticator-key file in MERVA OS/2 and send the authenticator keys from MERVA OS/2 to MERVA ESA via MERVA Link.

SWIFT Correspondents File Utility

The SWIFT Correspondents File Utility DWSCORUT fills the SWIFT Correspondents File of SWIFT Link with data from the SWIFT Bank Identifier Code (BIC) Directory update tape. This tape is created, maintained, and distributed by S.W.I.F.T. In this book, BICs are also referred to as *SWIFT addresses* or *SWIFT II logical terminals*.

DWSCORUT processes the BIC tape as follows:

- New SWIFT addresses are added to the file.
- Existing SWIFT addresses are replaced.
- SWIFT addresses that are in the file and were added by the DWSCORUT but are not on the tape are deleted from the file.
- SWIFT addresses that are in the file and were added by a USER using general file services but are not on the tape are left on the file.
- SWIFT addresses that are marked for deletion on the BIC tape are deleted from the file.
- If there are records with identical BICs, only the first record found is taken from the tape to the file; the others are ignored.

DWSCORUT writes a report that you can print on a system printer. The report shows all actions taken:

- The contents of the records that have been added
- The contents of the records that have been deleted and ignored
- The old and new contents of the records that have been changed
- Feedback information such as the return code, and error and confirmation messages

The processing of DWSCORUT is determined by control statements. The *MERVA for ESA Operations Guide* shows how to run DWSCORUT.

Report Layout

The report is formatted using MFS services and the system printer sections of an MCB.

Records are mapped from the file buffer to the MERVA ESA internal message buffer (TOF) using the MFS line formatter DSLMLFP. The record layout is described in the net section of an MCB.

You can customize the layouts of the reports by writing a new MCB or by changing the MCB DWSSCOR.

SWIFT Currency Code File Utility

The SWIFT currency code file utility DWSCURUT fills the SWIFT currency code file with data from the SWIFT bank identifier code (BIC) directory update tape. This tape is created, maintained, and distributed by S.W.I.F.T.

DWSCURUT processes the BIC tape as follows:

- New currencies are added to the file.
- Existing currencies are replaced.
- Currencies that are in the file and were added by the DWSCURUT but are not on the tape are deleted from the file.
- Currencies that are in the file and were added by a user using general file services but are not on the tape are left on the file.
- Currencies that are marked for deletion on the BIC tape are deleted from the file.

DWSCURUT writes a report that you can print on a system printer. The report shows all actions taken:

- The contents of the records that have been added
- The contents of the records that have been deleted and ignored
- The old and new contents of the records that have been changed
- Feedback information such as the return code, and error and confirmation messages

The processing of DWSCURUT is determined by control statements. The *MERVA for ESA Operations Guide* shows you how to run DWSCURUT.

Report Layout

The report is formatted using MFS services and the system printer sections of an MCB.

Records are mapped from the file buffer to the MERVA ESA internal message buffer (TOF) using the MFS line formatter DSLMLFP. The record layout is described in the net section of an MCB.

You can customize the layouts of the reports by writing a new MCB or by changing the MCB DWSSCUR.

Part 3. Appendixes

Appendix A. Journal Record Layouts

Each journal record consists of a 50 byte, fixed-length header, which is followed by the data length field and the journal data. The layout of the header is:

Byte 1	Journal record identifier. This identifies the type of the journal record and indicates which program created it. It also determines the layout of the data in the user-key extension, as well as the layout of the journal data. It can have one of the values shown in the list below.
Bytes 2 to 18	A 17-byte time stamp (see Table 2 on page 187 for details).
Bytes 19 to 21	The segment count. This is used only when the journal data is too large to fit into one record, in which case the record is segmented. The segments are numbered, starting with 001, up to a maximum of 999. If the record is not segmented, this field contains blanks.
Bytes 22 to 25	If the journal record is segmented, this field contains a slash followed by the total number of segments belonging to this record (/nnn). If the journal record is not segmented, this field contains blanks.
Bytes 26 to 50	The remaining 25 bytes of the journal header is the user-key extension. The data in the user-key extension depends on the journal record ID.

Bytes 2 through 21 of the journal record header form a 20-byte key that uniquely identifies each record in the journal data set, which is a VSAM KSDS.

The 1–byte journal record identifiers are shown below as 2–digit hexadecimal numbers. The program that creates this type of journal record is shown in parentheses. All values other than those shown are reserved.

ID	Description
00	Journal file initialization record (DSLJRNP)
02	MERVA ESA startup record (DSLNUC)
03	MERVA ESA stop record (DSLNUC)
04	End-user signon record (DSLNUSR)
05	End-user signoff record (DSLNUSR)
06	End-user function selection record (DSLNUSR)
07	Online maintenance record (DSLNUSR)
08	User file initialization (DSLNUSR)
09	User file termination (DSLNUSR)
10	Password check or change (DSLNUSR)
11	Invalid request (DSLNUSR)
12	General File maintenance (DSLEFLM)
13	Function table information (DSLNUSR)
14	MERVA ESA command and response (DSLNCS)

16	Unsolicited MERVA ESA operator messages (DSLNMOP)
17	Queue trace (DSLQMGT)
18	Debugging trace (DSLTRAP)
19	Routing trace (DSLRTNSC)
1A	Queue trace DB2 (DSLQMGD)
40	Received telex message (ENLHCF1)
41	Sent telex message (ENLHCF1)
50	Sent SWIFT message (DWSDGPA)
51	Received SWIFT message (DWSDGPA)
5F	Authenticator-key file maintenance (DWSAUTP)
60	Sent SWIFT message: • EKAMU044 for FMT/ESA with MERVA Link • DSLKQ044 for FMT/ESA with MERVA-MQI Attachment
61	Received SWIFT message: • EKAMU044 for FMT/ESA with MERVA Link • DSLKQ044 for FMT/ESA with MERVA-MQI Attachment
70	Outgoing MERVA Link application message (EKAAS10)
71	Outgoing MERVA Link acknowledgment message (EKAAS10)
72	Incoming MERVA Link application message (EKAAR10)
73	Incoming MERVA Link acknowledgment message (EKAAR10)
74	Incoming MERVA Link delivery report (EKAAR10)
78	Recovered MERVA Link in-process message (EKAAS10)
7F	MERVA Link Control Facility command (EKAEMSC)
80-8F	Journal record identifiers available for user programs
90	Sent MQI datagram (DSLKQS)
91	Sent MQI request message (DSLKQS)
92	Sent MQI reply message (DSLKQS)
97	Received MQI datagram (DSLKQR)
98	Received MQI request message (DSLKQR)
99	Received MQI reply message (DSLKQR)
9C	Received MQI exception report message (DSLKQR)
9D	Received MQI COA report message (DSLKQR)
9E	Received MQI COD report message (DSLKQR)
9F	Received unsupported MQI report message (DSLKQR)
D0-FE	Journal record identifiers available for user programs

Table 2. Journal Record Layouts

Offset in Decimal (in Hex)	Length in Bytes	Description
0 (0)	1	Journal record identifier. It can have one of the hexadecimal values shown in the list on page 185.
		Key field
1 (1)	17	Date and time in the form **YYYYMMDDHHMMSSppp**, where: **YYYY** year (for example 2001). Note: MERVA ESA Version 4 does not support journal record header formats with a 2-digit year. **MM** month (01 to 12). **DD** day (01 to 31). **HH** hour (00 to 23). **MM** minute (00 to 59). **SS** second (00 to 59). **ppp** A fraction used to differentiate among journal records created during the same second. Currently, this starts with 000 and is incremented to 999. In future releases, this field might be changed to support more than 1000 journal records during the same second. Theoretically, a maximum of 2^{24} different values is possible.
18 (12)	3	The segment count for a segmented record. This is used only when the journal data is too large to fit into one record, in which case the record is segmented. The segments are numbered, starting with 001, up to a maximum of 999. If the record is not segmented, this field contains blanks.
21 (15)	4	If the journal record is segmented, this field contains a slash followed by the total number of segments belonging to this record (/nnn). If the journal record is not segmented, this field contains blanks.
25 (19)	25	User-key extension. The data in the user-key extension depends on the journal record ID.
		User-Key Extension of Journal Records with ID 00:

25 (19)	25	`CL25'       vvvvJOURNAL A INIT' or`
		`CL25'       vvvvJOURNAL B INIT'`
		`CL25'SWITCH A TO B    FDBK=xx '`
		
		The placeholder `vvvv` applies to VSE only and represents the available space in bytes.

		User-Key Extension of Journal Records with IDs 04, 05, 06, 07, 08, 09, 10, 11, 13:

28 (1C)	4	`Journal      User-key`
		`Record ID    extension`
		`---------    ---------`
		`   04          'SON '`
		`   05          'SOFF'`
		`   06          'FSEL'`
		`   07          'ONL '`
		`   08          'INIT'`
		`   09          'TERM'`
		`   10          'CHPW'`
		`   11          'INV*'`
		`   13          'FNT '`

Table 2. Journal Record Layouts (continued)

Offset in Decimal (in Hex)	**Length in Bytes**	**Description**

34 (22)	4	For journal records with ID = 07, request:
		`'ADD '`
		`'DSP '`
		`'LST ' or 'LST1'`
		`'CHG '`
		`'DEL '`
		`'UNLK'`

50 (32)	4	Length field showing the length of data that follows from this point: For journal data smaller than 32767 bytes, the length value is contained in the first 2 bytes of this field; the remaining 2 bytes are reserved. For journal data larger than 32767 bytes, the first byte is set to X'80'. The length value is contained in the remaining 3 bytes of this field.
54 (36)	-	Begin of the MERVA ESA journal data. The data that follows depends on the journal record ID. For journal record ID 00, the data part is used only for switch records to fill the remaining space in the last control interval.
		Records with Journal Record IDs 02 and 03:

54 (36)	13	`Journal`
		`Record ID        Value`
		`---------    --------------`
		`   02        'MERVA STARTUP'`
		`   03        'MERVA STOP   '`

		Records with Journal Record IDs 04, 05, 06, 07, 08, 09, 10, 11, 13 show the DSLNUSR Parameter List from Offset 54 as follows:
54 (36)	1	Type code
55 (37)	1	Online Maintenance request code
56 (38)	1	DSLNUSR return code
57 (39)	1	Reserved
58 (3A)	2	Reserved
60 (3C)	2	DSLNUSR reason code
62 (3E)	8	User ID (of updating/requesting user)
70 (46)	8	User ID of update/display (ID = 07)
78 (4E)	8	Function name (ID = 06, Function selection) Logical terminal name (ID = 04, Signon) Next user ID in a list (ID = 07, List) Next function table entry name (ID = 12, FNT)
86 (56)	4	Signon sequence number binary
90 (5A)	4	Reserved
		Records with Journal Record ID 12:
50 (32)	2	Length field showing the length of data that follows from this point
52 (34)	2	Reserved
54 (36)	8	Name of the General File as defined with the DAT parameter of the DSLFLT macro.
62 (3E)	8	Update function ADD, REPLACE or DELETE padded with blanks.

Table 2. Journal Record Layouts (continued)

Offset in Decimal (in Hex)	**Length in Bytes**	**Description**
70 (46)	*	The data of the General File record. This data is described in the message control block defined by the MSGID parameter of the DSLFLT macro. The data does not start with a length field. For the MERVA ESA Nicknames File, the data is described in “Appendix B. Layout of the MERVA ESA Nicknames File” on page 195. For the SWIFT Correspondents File, the data is described in “Appendix D. Layout of the SWIFT Correspondents File” on page 199.
		Records with Journal Record ID 14:
54 (36)	8	User ID of MERVA ESA operator
62 (3E)	34	Origin ID of MERVA ESA operator
96 (60)	2	Length of the command part (CL)
98 (62)	2	Reserved
100 (64)	CL-4	Command with a length of CL - 4 The commands and responses are journaled with their actual lengths. The offsets of the subsequent fields depend on the length of the command.
96 (60) +CL	2	Length of the response part (RL)
96 (64) +CL+2	2	Reserved
96 (64) +CL+4	RL-4	Response with a length of RL - 4
		Records with Journal Record ID 16:
50 (32)	2	Length of the unsolicited operator message (LU)
52 (34)	2	Reserved
54 (36)	LU-4	Unsolicited operator message with a length of LU - 4
		Records with Journal Record ID 17 (Queue Trace):
54 (36)	148	Queue parameter list with the layout shown by a DSLQMG MF=L macro as it is given back to the caller, but without the queue parameter list extension.
202 (CA)	2	Length of queue element even if not shown for QTRACE=SMALL in DSLPRM
204 (CC)	2	Reserved
206 (CE)	2	Length of the queue element prefix as it is also stored in the queue data set with the queue element. This length varies with the number of queues in which a queue element was originally stored.
208 (D0)	2	Reserved
210 (D2)	2	Number of queues (1 to 12) in which the queue element was originally stored.
212 (D4)	2	Number of queues (1 to 12) in which the queue element is still stored.
214 (D6)	2	Length of all unique message references (UMRs) of this queue element. This length is a multiple of 32.
216 (D8)	1	'L' to indicate that the queue element is for a large message, and the message data is stored in the large message cluster (LMC). Binary zeros if the queue element is for a normal message.
217 (D9)	1	Reserved
218 (DA)	8	Name of the original queue for automatic delete and restart.
226 (E2)	4	Queue sequence number of the original queue for automatic delete and restart.

Table 2. Journal Record Layouts (continued)

Offset in Decimal (in Hex)	Length in Bytes	Description
230 (E6)	8	Reserved

238 (EE)	64 to 768	Queue descriptors of the queue element. There can be 1 to 12 queue descriptors as indicated by the original number of queues on offset 210 (D2).
		
		Each queue descriptor has the following format:
		` 8 bytes queue name (function)`
		` 4 bytes queue sequence number`
		` 1 byte  status (x'80' indicates MODIF=WRTBACK was used)`
		` 3 bytes reserved`
		`24 bytes key 1`
		`24 bytes key 2`

nnn	2	Block length of the queue element data including the UMRs. The offset nnn depends on the number of queue descriptors. From here, the data is only available if QTRACE=LARGE was specified in DSLPRM.
nnn+2	2	Reserved
nnn+4	2	Data length of the queue element data including the UMRs. This is the block length minus 4.
nnn+6	2	Reserved
nnn+8	0 to n*32	Unique message references of the queue element in the length indicated by the UMR length field on offset 214 (D6). Each UMR is 32 bytes long and starts with the 4 characters MUMR.
mmm	2	Block length of the queue element data without the UMRs. The offset mmm depends on the offset nnn and the number of UMRs.
mmm+2	2	Reserved
mmm+4	2	Data length of the queue element data without the UMRs. This is the block length minus 4.
mmm+6	2	Reserved
mmm+8	*	The message data in the length (minus 4) indicated at offset mmm+4. The message data can be in the MERVA ESA TOF format, or the status information of a large message, or any other contents.
		Records with Journal Record ID 18 (Debugging Trace):
54 (36)	2	Reserved
56 (38)	8	Time stamp indicating when this trace block was started.
64 (40)	32	CL32' **** MERVA TRACE TABLE **** '
96 (60)	32	Trace table header of the particular program
128 (80)	*	Trace table entries until the end of the journal record The format of the debugging trace table entries is described in the *MERVA for ESA Diagnosis Guide*.
		Records with Journal Record ID 19 (Routing Trace):
54 (36)	10	Reserved
64 (40)	32	CL32' ROUTING TRACE FOR *table* ' *table* is replaced by the name of the routing table which is traced
96 (60)	*	Routing trace table entries of 32 bytes each as described in "Appendix G. Layout of the Routing Trace Entries" on page 209.
		Records with Journal Record ID 1A (Queue Trace DB2):

Table 2. Journal Record Layouts (continued)

Offset in Decimal (in Hex)	Length in Bytes	Description
54 (36)	148	Queue parameter list with the layout shown by a DSLQMG MF=L macro as it is given back to the caller, but without the queue parameter list extension.

202 (CA)	24	DB2 I/O module return information
		
		From here, the data is only available if QTRACE=LARGE was specified in DSLPRM.
		
		The return information has the following format:
		`4 bytes access method (DB2)`
		`8 bytes I/O module name`
		`5 bytes debug information (last statement)`
		`1 byte  I/O module return code`
		`1 byte  I/O module reason code`
		`1 byte  trigger status for backref function`
		`        X'80'  element inserted (ECB)`
		`        X'40'  element inserted (Trx)`
		`4 bytes SQL code`

226 (E2)	984 (12*82)	12 queue descriptors
		
		Each queue descriptor has the following format:
		` 8 bytes function name`
		` 4 bytes QSN`
		` 8 bytes key1 fieldname`
		`24 bytes key1 value`
		` 8 bytes key2 fieldname`
		`24 bytes key2 value`
		` 2 bytes partition ID`
		` 1 byte  queue status byte 1  (corresponds to FNTQUEST)`
		` 1 byte  queue status byte 2  (corresponds to FNTQUST2)`
		` 1 byte  queue status byte 3`
		`         X'04'  extra keys`
		` 1 byte  trigger status`
		`         X'80'  element inserted (ECB)`
		`         X'40'  element inserted (Trx)`

1210 (4BA)	4	Message buffer prefix
1214 (4BE)	2	Message data length including the UMRs and the length field.
1216 (4C0)	2	Reserved
1218 (4C2)	0 to n*32	Unique message references of the queue element. Each UMR is 32 bytes long and starts with the 4 characters MUMR.
mmm	2	Block length of the message data without the UMRs. The offset mmm depends on the number of UMRs.
mmm+2	2	Reserved
mmm+4	2	Data length of the message data without the UMRs + 4.
mmm+6	2	Reserved
mmm+8	*	The message data in the length (minus 4) indicated at offset mmm+4. The message data can be in the MERVA ESA TOF format, or any other contents.
		Records with Journal Record IDs 50, 51:
50 (32)	2	Length of the SWIFT or acknowledgment message (ML)
52 (34)	2	Reserved
54 (36)	ML-4	The SWIFT or acknowledgment message with a length of ML - 4. The basic and application headers must be used to determine the APDU and message type.

Table 2. Journal Record Layouts (continued)

Offset in Decimal (in Hex)	Length in Bytes	Description
		Records with Journal Record ID 5F:
50 (32)	2	Length field showing the length of data that follows from this point
52 (34)	2	Reserved
54 (36)	8	User ID of updating user
62 (3E)	34	Origin ID of updating user
96 (60)	1	Update function for DWSAUTP. These are the same values as defined for the field AUTPUPDF in the DWSAUT MF=L macro expansion.
97 (61)	8	Home destination for update
105 (69)	3	Blanks
108 (6C)	8	Corresponding destination for update
116 (74)	3	Blanks
119 (77)	*	Repetition of the sequence of fields from offset 97 (61) to offset 119 (77) for generic delete and exchange as indicated by the length field on offset 50 (34).
		Records with Journal Record IDs 70, 71, 72, 73, 74, 78: These records of MERVA Link use the user-key extension on offset 25 as follows:
25 (19)	2	Message class (ID 78 only)
27 (1B)	1	Reserved
28 (1C)	4	Message sequence number
32 (20)	8	Application support process (ASP) name
40 (28)	10	Reserved
50 (32)	4	Journal record data length (JRDL in the format LL00 or LLLL)
		Records with Journal Record IDs 70 and 72 (outgoing or incoming application message):
54 (36)	2	Length of the MERVA Link header (HL)
56 (38)	2	Header ID 0120
58 (3A)	HL-4	Header data
54 (36) +HL	4	Body part ID 00008122 or 00008123
54 (36) +HL +4	4	Body part data length (DL)
54 (36) +HL +8	DL-4	Message text (body part data)
		Records with Journal Record IDs 71 and 73 (outgoing or incoming acknowledgment message):
54 (36)	2	Length of the MERVA Link acknowledgment message (SL)
56 (38)	2	Status report ID 0122
58 (3A)	SL-4	Acknowledgment message data
		Records with Journal Record ID 74 (incoming delivery report):
54 (36)	2	Length of the MERVA Link delivery report (RL)
56 (38)	2	Delivery report ID 0111
58 (3A)	RL-4	Delivery report data

Table 2. Journal Record Layouts (continued)

Offset in Decimal (in Hex)	Length in Bytes	Description
		Records with Journal Record ID 78 (recovered in-process (IP) message in the MERVA ESA queue format):
54 (36)	JRDL-4	Data of the queue buffer starting with *DSLTOF$
		Records with Journal Record ID 7F (MERVA Link control command and response):
25 (19)	25	Reserved
54 (36)	8	User ID of the MERVA ESA operator
62 (3E)	34	Origin ID of the MERVA ESA operator
96 (60)	2	MERVA Link control command length (CL = X'001C')
98 (62)	2	Reserved
100 (64)	24	MERVA Link control command
124 (7C)	2	Length of response (RL)
126 (7E)	2	Reserved
128 (80)	RL-4	Response data
		User-Key Extension of Journal Records with IDs 90, 91, 92, 97, 98:
25 (19)	24	Message identifier field *MsgId* from the MQI message descriptor control block MQMD.
49 (31)	1	Reserved
		User-Key Extension of Journal Records with IDs 99, 9C, 9D, 9E, 9F:

25 (19)	4	`Journal`
		 `Record ID      Value`
		 `---------    ---------`
		 `   99         'XACK'`
		 `   9C         'XEXC'`
		 `   9D         'XCOA'`
		 `   9E         'XCOD'`
		 `   9F         'XUNS'`

29 (1D)	20	Bytes 5 to 24 of the message identifier field *MsgId* from the MQI message descriptor control block MQMD.
49 (31)	1	Reserved
		Records with Journal Record IDs 90, 91, 92, 97, 98, 99, 9C, 9D, 9E, 9F:
50 (32)	4	Journal record data length (JRDL in the format LL00 or LLLL)
54 (36)	JRDL-4	The application message data of the MQI message type

Appendix B. Layout of the MERVA ESA Nicknames File

The layout of the MERVA ESA Nicknames File is defined in the message control block (MCB) DSL0CORN.The MCB also describes how a record is displayed during online maintenance using the MERVA ESA General File Maintenance, and how it is printed when the file is listed using the MERVA ESA utility program DSLFLUT.

The records of the MERVA ESA Nicknames File have a fixed length of 304 bytes. Table 3 summarizes the format of the Nicknames File.

Table 3. Format of the MERVA ESA Nicknames File

Offset Decimal (Hex)	Length (Bytes)	Field Name	Description
0 (0)	2		Record length field
2 (2)	2		Reserved
4 (4)	24	DSLCORID	Correspondents Identifier
28 (1C)	124	DSLBLANK	Reserved (blanks)

152 (98)	26	DSLFLCUP	Creation Stamp of the format:
			`UUUUUUUU  User ID   8 bytes`
			`00        Reserved  2 bytes`
			`YY/MM/DD  Date      8 bytes`
			`HH:MM:SS  Time      8 bytes`

178 (B2)	26	DSLFLUP	Update Stamp of the format:
			`UUUUUUUU  User ID   8 bytes`
			`00        Reserved  2 bytes`
			`YY/MM/DD  Date      8 bytes`
			`HH:MM:SS  Time      8 bytes`

204 (CC)	6		Reserved
210 (D2)	32	DSLCORN	Key of record (search field)
210 (D2)	8	DSLCORNO	Owner prefix of key
218 (DA)	24	DSLCORNN	Nickname of key
242 (F2)	62	DSLBLANK	Reserved (blanks)

The first 4 bytes (record length field and 2 bytes reserved) are not defined in the MCB but added by the MERVA ESA message format services (DSLMMFS).

Appendix C. Layout of the Currency Code File

The layout of the Currency Code File is defined in the message control block (MCB) DWSSCUR.The MCB also describes how a record is displayed during online maintenance using the MERVA ESA General File Maintenance, and how it is printed when the file is listed using the MERVA ESAutility program DSLFLUT. In addition, the MCB defines the layout of the report listing when loading the file using the SWIFT Link utility program DWSCURUT.

The records of the Currency Code File have a fixed length of 1622 bytes.

Table 4 summarizes the format of the Currency Code File.

Table 4. Format of the Currency Code File

Offset Decimal (Hex)	Length (Bytes)	Field Name	Description
0 (0)	2		Record length field
2 (2)	2		Reserved
4 (4)	3	DWSCURID	Key: Currency's Identifier
7 (7)	70	DWSCURNM	Key: Currency Name
77 (4D)	1	DWSCURF	Key: Maximum Length of Fraction
78 (4E)	2	DWSCURC01	Key: Country Code of 1st country using the currency
80 (50)	70	DWSCURN01	Key: Country Name of 1st country using the currency
150 (96)	2	DWSCURC02	Key: Country Code of 2nd country using the currency
152 (98)	70	DWSCURN02	Key: Country Name of 2nd country using the currency
222 (DE)	2	DWSCURC03	Key: Country Code of 3rd country using the currency
224 (E0)	70	DWSCURN03	Key: Country Name of 3rd country using the currency
294 (126)	2	DWSCURC04	Key: Country Code of 4th country using the currency
296 (128)	70	DWSCURN04	Key: Country Name of 4th country using the currency
366 (16E)	2	DWSCURC05	Key: Country Code of 5th country using the currency
368 (170)	70	DWSCURN05	Key: Country Name of 5th country using the currency
438 (1B6)	2	DWSCURC06	Key: Country Code of 6th country using the currency
440 (1B8)	70	DWSCURN06	Key: Country Name of 6th country using the currency
510 (1FE)	2	DWSCURC07	Key: Country Code of 7th country using the currency
512 (200)	70	DWSCURN07	Key: Country Name of 7th country using the currency
582 (246)	2	DWSCURC08	Key: Country Code of 8th country using the currency
584 (248)	70	DWSCURN08	Key: Country Name of 8th country using the currency
654 (28E)	2	DWSCURC09	Key: Country Code of 9th country using the currency
656 (290)	70	DWSCURN09	Key: Country Name of 9th country using the currency
726 (2D6)	2	DWSCURC10	Key: Country Code of 10th country using the currency
728 (2D8)	70	DWSCURN10	Key: Country Name of 10th country using the currency
798 (31E)	2	DWSCURC11	Key: Country Code of 11th country using the currency
800 (320)	70	DWSCURN11	Key: Country Name of 11th country using the currency
870 (366)	2	DWSCURC12	Key: Country Code of 12th country using the currency
872 (368)	70	DWSCURN12	Key: Country Name of 12th country using the currency
942 (3AE)	2	DWSCURC13	Key: Country Code of 13th country using the currency
944 (3B0)	70	DWSCURN13	Key: Country Name of 13th country using the currency
1014 (3F6)	2	DWSCURC14	Key: Country Code of 14th country using the currency

Table 4. Format of the Currency Code File (continued)

Offset Decimal (Hex)	Length (Bytes)	Field Name	Description
1016 (3F8)	70	DWSCURN14	Key: Country Name of 14th country using the currency
1086 (43E)	2	DWSCURC15	Key: Country Code of 15th country using the currency
1088 (440)	70	DWSCURN15	Key: Country Name of 15th country using the currency
1158 (486)	2	DWSCURC16	Key: Country Code of 16th country using the currency
1160 (488)	70	DWSCURN16	Key: Country Name of 16th country using the currency
1230 (4CE)	2	DWSCURC17	Key: Country Code of 17th country using the currency
1232 (4D0)	70	DWSCURN17	Key: Country Name of 17th country using the currency
1302 (516)	2	DWSCURC18	Key: Country Code of 18th country using the currency
1304 (518)	70	DWSCURN18	Key: Country Name of 18th country using the currency
1374 (55E)	2	DWSCURC19	Key: Country Code of 19th country using the currency
1376 (560)	70	DWSCURN19	Key: Country Name of 19th country using the currency
1446 (5A6)	2	DWSCURC20	Key: Country Code of 20th country using the currency
1448 (5A8)	70	DWSCURN14	Key: Country Name of 20th country using the currency

1518 (5EE)	26	DSLFLCUP	Creation Stamp of the format:
			`UUUUUUUU  User ID   8 bytes`
			`00        Reserved  2 bytes`
			`YY/MM/DD  Date      8 bytes`
			`HH:MM:SS: Time      8 bytes`

1544 (608)	26	DSLFLUP	Update Stamp of the format:
			`UUUUUUUU  User ID   8 bytes`
			`00        Reserved  2 bytes`
			`YY/MM/DD  Date      8 bytes`
			`HH:MM:SS  Time      8 bytes`

1570 (622)	26	DWSCURIM	First import stamp of the format:
			`DWSCURUT  User ID   8 bytes`
			`00        Reserved  2 bytes`
			`YY/MM/DD  Date      8 bytes`
			`HH:MM:SS  Time      8 bytes`

1596 (63C)	26	DWSCURST	Last import stamp of the format:
			`DWSCURST  User ID   8 bytes`
			`00        Reserved  2 bytes`
			`YY/MM/DD  Date      8 bytes`
			`HH:MM:SS  Time      8 bytes`

The first 4 bytes (record length field and 2 bytes reserved) are not defined in the MCB but added by the message format services (DSLMMFS).

Appendix D. Layout of the SWIFT Correspondents File

The layout of the SWIFT Correspondents File is defined in the message control block (MCB) DWSSCOR, and is shown in Table 5.The MCB also describes how a record is displayed during online maintenance using the MERVA ESA General File Maintenance, and how it is printed when the file is listed using the MERVA ESAutility program DSLFLUT. In addition, the MCB defines the layout of the report listing when loading the file using the SWIFT Link utility program DWSCORUT.

The records of the SWIFT Correspondents File have a fixed length of 1738 bytes. The first 4 bytes (record length field and 2 bytes reserved) are not defined in the MCB but added by the message format services (DSLMMFS).

Table 5. Format of the SWIFT Correspondents File

Offset Decimal (Hex)	Length (Bytes)	Field Name	Description
0 (0)	2		Record length field
2 (2)	2		Reserved
4 (4)	24	DSLCORID	Key: Correspondent's Identifier

28 (1C)	26	DSLFLCUP	Creation Stamp of the format:
			`UUUUUUUU  User ID   8 bytes`
			`00        Reserved  2 bytes`
			`YY/MM/DD  Date      8 bytes`
			`HH:MM:SS  Time      8 bytes`

54 (36)	26	DSLFLUP	Update Stamp of the format:
			`UUUUUUUUU  User ID   8 bytes`
			`00         Reserved  2 bytes`
			`YY/MM/DD   Date      8 bytes`
			`HH:MM:SS   Time      8 bytes`

80 (50)	6		Reserved
86 (56)	140	DWSCORBK	Correspondent's name
86 (56)	35	DWSCORB1	Correspondent's name line 1
121 (79)	35	DWSCORB2	Correspondent's name line 2
156 (9C)	35	DWSCORB3	Correspondent's name line 3
191 (BF)	35	DWSCORB4	Correspondent's name line 4
226 (E2)	200	DWSCORAD	Correspondent's address
226 (E2)	40	DWSCORA1	Correspondent's address line 1
266 (10A)	40	DWSCORA2	Correspondent's address line 2
306 (132)	40	DWSCORA3	Correspondent's address line 3
346 (15A)	40	DWSCORA4	Correspondent's address line 4
386 (182)	40	DWSCORA5	Correspondent's address line 5
426 (1AA)	10	DWSCORZP	ZIP code
436 (1B4)	16	DSLBLANK	Reserved (blanks)

452 (1C4)	26	DWSCORIM	First import stamp of the format:
			`DWSCORUT  User ID   8 bytes`
			`00        Reserved  2 bytes`
			`YY/MM/DD  Date      8 bytes`
			`HH:MM:SS  Time      8 bytes`

478 (1DE)	26	DWSCORST	Last import stamp of the format:
			`DWSCORUT  User ID   8 bytes`
			`00        Reserved  2 bytes`
			`YY/MM/DD  Date      8 bytes`
			`HH:MM:SS  Time      8 bytes`

Table 5. Format of the SWIFT Correspondents File (continued)

Offset Decimal (Hex)	Length (Bytes)	Field Name	Description
504 (1F8)	1234	DWSCORBE	Unedited data from BIC Directory Update Tape
504 (1F8)	35	DWSCORI1	Institution Name (first part)
539 (21B)	35	DWSCORI2	Institution Name (second part)
574 (23E)	35	DWSCORI3	Institution Name (third part)
609 (261)	35	DWSCORR1	Branch Information (first part)
644 (284)	35	DWSCORR2	Branch Information (second part)
679 (2A7)	35	DWSCORC1	City Heading
714 (2CA)	4	DWSCORS1	Subtype indication
718 (2CE)	30	DWSCORV1	Value Added Services (first 10 fields)
748 (2EC)	30	DWSCORV2	Value Added Services (second 10 fields)
778 (30A)	35	DWSCORE1	Extra Information
813 (32D)	35	DWSCORP1	Physical address (first part)
848 (350)	35	DWSCORP2	Physical address (second part)
883 (373)	35	DWSCORP3	Physical address (third part)
918 (396)	35	DWSCORP4	Physical address (fourth part)
953 (3B9)	35	DWSCORL1	Location (first part)
988 (3DC)	35	DWSCORL2	Location (second part)
1023 (3FF)	35	DWSCORL3	Location (third part)
1058 (422)	35	DWSCORN1	Country Name (first part)
1093 (445)	35	DWSCORN2	Country Name (second part)
1128 (468)	35	DWSCORW1	POB Number
1163 (48B)	35	DWSCORX1	POB Location (first part)
1198 (4AE)	35	DWSCORX2	POB Location (second part)
1233 (4D1)	35	DWSCORX3	POB Location (third part)
1268 (4F4)	35	DWSCORY1	POB Country Name (first part)
1303 (517)	35	DWSCORY2	POB Country Name (second part)
1338 (53A)	400	DWSCORZ1	BIC+ fields
1338 (53A)	2	DWSCORSC	Source of Info
1340 (53C)	6	DWSCORSP	Special Code
1346 (542)	1	DWSCORNM	Non-merged Code
1347 (543)	6	DWSCORUP	Update Date
1353 (549)	4	DWSCOROR	Origin of Key
1357 (54D)	8	DWSCORKY	Bic+ Key
1365 (555)	6	DWSCORCH	Chips UID
1371 (55B)	15	DWSCORNI	National ID
1386 (56A)	6	DWSCORAC	Activation Date
1392 (570)	6	DWSCORDE	Deactivation Date
1398 (576)	8	DWSCORUN	Undo Merge BIC+
1406 (57E)	8	DWSCORNW	New BIC+ Key
1414 (586)	128	DWSCORUS	User Field:
1414 (586)	43	DWSCORU1	User Field 1
1457 (5B1)	43	DWSCORU2	User Field 2
1500 (5DC)	42	DWSCORU3	User Field 3
1542 (606)	17	DWSCOR1X	Cross Border 1
1559 (617)	17	DWSCOR2X	Cross Border 2
1576 (628)	17	DWSCOR3X	Cross Border 3
1593 (639)	17	DWSCOR4X	Cross Border 4
1610 (64A)	29	DWSCOR1W	X with Dates 1
1639 (667)	29	DWSCOR2W	X with Dates 2
1668 (684)	70	DWSCORTR	Trailer
1668 (684)	35	DWSCORT1	Trailer (first part)
1703 (6A7)	35	DWSCORT2	Trailer (second part)

Appendix E. Layout of the Telex Correspondents File

The layout of the Telex Correspondents File is defined in the message control block (MCB) ENLTCORthat also describes how:

- A record is displayed during online maintenance using the general file maintenance
- It is printed when the file is listed using the MERVA ESA utility program DSLFLUT.

The records of the Telex Link Telex Correspondents File have a fixed length of 504 bytes. Table 6 summarizes the format of the Telex Correspondents File.

Table 6. Format of the Telex Correspondents File

Offset Decimal (Hex)	Length (Bytes)	Field Name	Description
0 (0)	2		Record length field
2 (2)	2		Reserved
4 (4)	24	DSLCORID	Key: Correspondent's Identifier
28 (1C)	124	DSLZERO	Reserved (zeros)

152 (98)	26	DSLFLCUP	Creation Stamp with the format:
			`UUUUUUUU  User ID   8 bytes`
			`00        Reserved  2 bytes`
			`YY/MM/DD  Date      8 bytes`
			`HH:MM:SS  Time      8 bytes`

178 (B2)	26	DSLFLUP	Update Stamp with the format:
			`UUUUUUUU  User ID   8 bytes`
			`00        Reserved  2 bytes`
			`YY/MM/DD  Date      8 bytes`
			`HH:MM:SS  Time      8 bytes`

204 (CC)	6		Reserved
210 (D2)	35	ENLFLCO1	Correspondent's address line 1
245 (F5)	35	ENLFLCO2	Correspondent's address line 2
280 (118)	22	ENLFLNR1	First telex number
302 (12E)	22	ENLFLNR2	Second telex number
324 (144)	120	ENLFLTKY	Test-key requirements of 6 bytes each; a 4-byte message type, "Y" or "N", and 1 reserved byte
444 (1BC)	20	ENLFLAB1	Long Answerback 1
464 (1D0)	20	ENLFLAB2	Long Answerback 2
484 (1E4)	20	DSLZERO	Reserved (zeros)

The first 4 bytes (record length field and 2 reserved bytes) are not defined in the MCB but added by the message format services (DSLMMFS).

Appendix F. Layout of the DB2 Tables

If you use queue management using DB2, the messages are stored in the following DB2 tables:

DSLTQUEL	Queue element table
DSLTQXDEF	Extra-key definition table
DSLTQXKEY	Queue extra-key table
DSLTQBUSY	Busy table
DSLTQMSG	Message table
DSLTQFUN	Function control table (for example, the highest QSN used so far)
DSLTQSTAT	MERVA status table

Note: The clause "NOT NULL WITH DEFAULT" applies only to DB2 for OS/390.

Table DSLTQUEL (Queue Element Table)

This table is used to store queue element data. It contains the following columns (the names and descriptions of columns that comprise the primary key are shown in **bold**):

Column Name	Description	Data Type and Additional Information
QUEUE	**queue name**	CHAR(8), NOT NULL, PRIMARY KEY
QSN	**queue sequence number**	INTEGER, NOT NULL, PRIMARY KEY Allowed values: 1 to +2 147 483 647
KEY1FIELD	key 1 field name	CHAR(8), NOT NULL This is the TOF field name, for example, SW20.
KEY1VALUE	key 1 value	CHAR(25), NOT NULL, NON-UNIQUE INDEX
KEY2FIELD	key 2 field name	CHAR(8), NOT NULL This is the TOF field name, for example, SWBHSN.
KEY2VALUE	key 2 value	CHAR(25), NOT NULL, NON-UNIQUE INDEX
UMR_MERVA	UMR MERVA name	CHAR(8), NOT NULL This is the name as specified for the NAME parameter of the MERVA customization module DSLPRM.
UMR_SEQNO	UMR sequence number	INTEGER, NOT NULL Allowed values: 1 to +99 999 999 Value is 0 for messages written with the modifier NOUMR, for example, messages in the queue SLSLTT.
UMR_DATE	UMR date	DATE, NOT NULL WITH DEFAULT YYYY-MM-DD (int. 4 bytes, ext. 10 characters)
UMR_TIME	UMR time	TIME, NOT NULL WITH DEFAULT HH.MM.SS (internally 3 bytes, externally 8 characters)

PUTOPERATION	PUT operation method	CHAR(1), NOT NULL WITH DEFAULT 'S'
		M = multiple PUT
		S = single PUT

Column Name	Description	Data Type and Additional Information

WRTBACK_IND	write-back indicator	CHAR(1), NOT NULL WITH DEFAULT 'N'
		N = 'not written back'
		Y = 'written back'

PARTID	part ID	CHAR(2), NOT NULL Part ID of this message in DSLTQMSG.
MSGTABLENO	message number	DECIMAL(13,0), NOT NULL Number of this message in DSLTQMSG.
DEPTH	highest sequence number	SMALLINT, NOT NULL This is the highest sequence number of this message in DSLTQMSG. If set to 1, the message is not split, and the message length must be less than or equal to 4000 characters.

Note: The key values are stored with trailing hexadecimal zeros and the 25th (last) character is always hexadecimal zero.

Table DSLTQXDEF (Extra-Key Definition Table)

This table is used to define the extra keys for a queue. It contains the following columns (the names and descriptions of columns that comprise the primary key are shown in **bold**):

Column Name	Description	Data Type and Additional Information
QUEUE	**queue name**	CHAR(8), NOT NULL, PRIMARY KEY
KEYNO	**key number**	SMALLINT, NOT NULL, PRIMARY KEY Because MERVA already uses key numbers 1 and 2, it is recommended that you start with key number 3.

ACTIVE	active indicator	CHAR(1), NOT NULL WITH DEFAULT 'Y'
		N = not active
		Y = active

KEYFIELD	key field name	CHAR(8), NOT NULL The TOF field name, for example SW20.
STARTPOS	start position in the TOF field	SMALLINT, NOT NULL
LENGTH	key length	SMALLINT, NOT NULL Allowed values: 1 to 48
KEYDESC	description	VARCHAR(24), NOT NULL WITH DEFAULT Description of this key field, for example "currency code".
CREATOR	creator of the extra-key definition	CHAR(8), NOT NULL WITH DEFAULT USER
CREATEDATE	creation date	DATE, NOT NULL WITH DEFAULT Date the extra-key definition was created.
CREATETIME	creation time	TIME, NOT NULL WITH DEFAULT Time the extra-key definition was created.
CREATECOMMENT	creator's comment	VARCHAR(200), NOT NULL WITH DEFAULT

Note: To make QUEUE and KEYFIELD the primary key is not possible, because you might want to define the same KEYFIELD more than once as extra key, for example, one with STARTPOS 1, and one with STARTPOS 5.

Table DSLTQXKEY (Queue Extra-Key Table)

This table is used to store the extra-key data. It contains the following columns (the names and descriptions of columns that comprise the primary key are shown in **bold**):

Column Name	Description	Data Type and Additional Information
QUEUE	**queue name**	CHAR(8), NOT NULL, PRIMARY KEY
QSN	**queue sequence number**	INTEGER, NOT NULL, PRIMARY KEY
KEYNO	**key number**	SMALLINT, NOT NULL, PRIMARY KEY
KEYFIELD	key field name	CHAR(8), NOT NULL, NON-UNIQUE INDEX The TOF field name, for example SW20.
STARTPOS	Start position in the TOF field	SMALLINT, NOT NULL
LENGTH	Length of the key	SMALLINT, NOT NULL
KEYDESC	description	VARCHAR(24), NOT NULL WITH DEFAULT Description of this key field.
KEYVALUE	key value	VARCHAR(48), NOT NULL, NON-UNIQUE INDEX
CREATEDATE	creation date	DATE, NOT NULL WITH DEFAULT Date the extra-key definition was created.
CREATETIME	creation time	TIME, NOT NULL WITH DEFAULT Time the extra-key definition was created.
CREATECOMMENT	creator’s comment	VARCHAR(200), NOT NULL WITH DEFAULT

Table DSLTQBUSY (Busy Table)

This table contains one entry for each message currently in use; that is, each message that is busy. It contains the following columns (the names and descriptions of columns that comprise the primary key are shown in **bold**):

Column Name	Description	Data Type and Additional Information
QUEUE	**queue name**	CHAR(8), NOT NULL, PRIMARY KEY
QSN	**queue sequence number**	INTEGER, NOT NULL, PRIMARY KEY Allowed values: 1 to +2 147 483 647

SERVICE	Service Indicator	CHAR(1), NOT NULL
		C = central service
		D = direct service

PROGNAME	Program Name (DSLQMDIO)	CHAR(8), NOT NULL
BUSY_TIMESTAMP	BUSY Timestamp	TIMESTAMP, NOT NULL WITH DEFAULT YYYY-MM-DD-HH.MM.SS.TTTTTT (internally 10 bytes, externally 26 characters)

Note: Every time MERVA is started, all central entries are cleared with a `DELETE FROM DSLTQBUSY WHERE SERVICE = 'C'`.

Table DSLTQMSG (Message Table)

This table is used to store the message data. It contains the following columns (the names and descriptions of columns that comprise the primary key are shown in **bold**):

Column Name	Description	Data Type and Additional Information
PARTID	**Partition ID**	CHAR(2), NOT NULL, PRIMARY KEY
MSGTABLENO	**message number**	DECIMAL(13,0), NOT NULL, PRIMARY KEY Note: The MSGTABLENO is UNIQUE.
SEQNO	**message sequence number**	SMALLINT, NOT NULL, PRIMARY KEY Allowed values: 1 to +32767
MESSAGE	message	VARCHAR(4000), NOT NULL

Table DSLTQFUN (Function Control Table)

This table is used to store control information about queues. It contains the following columns (the names and descriptions of columns that comprise the primary key are shown in **bold**):

Column Name	Description	Data Type and Additional Information
QUEUE	**queue name**	CHAR(8), NOT NULL, PRIMARY KEY
MAXQSN	Highest QSN used so far in this queue	INTEGER, NOT NULL Allowed values: 1 to +2 147 483 647
STATUS	Queue Status	CHAR(6), NOT NULL HOLD or NOHOLD
TRIGGERECB	trigger insert (ECB)	CHAR(1), NOT NULL Y = set flag when message is inserted in queue N = do not set flag when message is inserted in queue
TRIGGERINS	trigger insert (TRX)	CHAR(1), NOT NULL Y = set flag when message is inserted in queue N = do not set flag when message is inserted in queue

Table DSLTQSTAT (MERVA Status Table)

This table is used to store message number (MSGTABLENO) and UMR of the message that was last added to the queue. It contains the following columns:

Column Name	Description	Data Type and Additional Information
MSGTABLENO	message number	DECIMAL(13,0), NOT NULL Message number of the last message added to the queue. This will be the highest message number in the queue.
UMR_MERVA	MERVA name	CHAR(8), NOT NULL Identifier portion (MERVA name) of the UMR of the last message added to the queue.
UMR_SEQNO	sequence number	INTEGER, NOT NULL Sequence number portion of the UMR of the last message added to the queue.
UMR_DATE	date	DATE, NOT NULL WITH DEFAULT YYYY-MM-DD (internally 4 bytes, externally 10 characters) Date portion of the UMR of the last message added to the queue.

Column Name	Description	Data Type and Additional Information
UMR_TIME	time	TIME, NOT NULL WITH DEFAULT HH.MM.SS (internally 3 bytes, externally 8 characters) Time portion of the UMR of the last message added to the queue.

Appendix G. Layout of the Routing Trace Entries

This appendix describes the layout of each particular routing trace entry. Uppercase letters appear in the routing trace entries as shown; italics are variables and are explained for each individual routing trace entry.

Routing Trace Header

The routing trace header is always the first entry of a routing trace.

Format:

```
ROUTING TRACE FOR rt-name
```

Explanation:

rt-name is the name of the routing table as coded in its first DSLROUTE macro. It shows which routing table is processed.

Routing Trace DEFINE Entry

Format:

```
nnn DEF var-name
tof-name result
```

Explanations:

nnn
Is the number of the routing table entry processed.

DEF
Shows that a DSLROUTE TYPE=DEFINE is processed.

var-name
Is the name of the variable field to be defined.

tof-name
Is the name of the TOF field from which the variable field is to be defined. If a literal value is used instead of a TOF field name, then *LITERAL* is shown instead of a TOF field name.

result
Is the result of the definition. It can have the following values:

FOUND	The TOF field was found and contained enough data about the DISP and LENGTH parameters to have at least 1 byte of data for the variable field. In this case, the DEF trace entry is followed by another routing trace entry showing the data found and padded to 32 bytes with binary zeros if necessary.
EMPTY	The TOF field was found but was either empty or did not contain enough data about the DISP and LENGTH parameters to have at least 1 byte of data for the variable field.
TNOTFD	The TOF field was not found, that is, the MERVA ESA TOF supervisor returned reason code 3.

RC=*nnn*	The TOF field was not found for the reason indicated by *nnn*. See the *MERVA for ESA Messages and Codes* for the explanation of the reason codes of DSLTOFSV.
VNOTFD	The TOF field was found, but this was the definition of the 21st variable field name, and only 20 field names are possible.

Routing Trace TEST Entry

Format:

```
nnn TST operand1
operand2 result
```

Explanations:

nnn
Is the number of the routing table entry processed.

TST
Shows that a DSLROUTE TYPE=TEST is processed.

operand1
Is the first operand of the test. It is either the name of a variable field or LITERAL when a literal is the first operand.

operand2
Is the second operand of the test. It is either the name of a variable field or LITERAL when a literal is the second operand.

result
Is the result of the test. It can have the following values:

TRUE	Depending on the test operator and the modifier (not shown in the routing trace entry), all variable fields have been found, compared, and the condition is true; that is, the TRUE label is taken for further processing of the routing table. Two additional routing trace entries follow showing the data compared. For the AMOUNT modifier, the operands are shown after adjustment to the decimal comma.
FALSE	Depending on the test operator and the modifier, all variable fields have been found, compared, and the condition is false (that is, the FALSE label is taken for further processing of the routing table). Two additional routing trace entries follow, showing the data compared. For the AMOUNT modifier, the operands are shown after adjustment to the decimal comma.
VNOTFD	A variable field name was specified but it was not defined successfully before the test, or it was dropped before the test. Ensure that the FOUND, NOTFND, and EMPTY labels are always used correctly during DEFINE.
OVERFL	The AMOUNT modifier was specified, but the two operands could not be adjusted with their significant digits before and after the decimal comma within 32 bytes. The same processing takes place as for FALSE.

Routing Trace SET Entry

Format:

```
nnn SET functn-name-parts
result
```

Explanations:

nnn
Is the number of the routing table entry processed.

SET
Shows that a DSLROUTE TYPE=SET was processed.

functn-name-parts
Show the parts (up to 8) of which a target function can be composed. See the source program or the assembler listing for which parts are to be composed. The SET entry shows each part separated from the other by a blank. The separating blanks are omitted when a target function is given to DSLQMGT.

result
Is the result of the set. It can have the following values:

(blank)	Set was successful with no error.
TRUNC	Set was successful, but the concatenated variables and literals used were longer than 8 characters. Only 8 characters in the set will be used; characters in excess of this are ignored.
FNOTFD	Set was successful but the target set was not found in the function table. None of the targets found until then is used, and processing continues with the FINAL statement.
VNOTFD	A variable field name was specified but it was not defined successfully before the set, or has been dropped before the set. Ensure that always the FOUND, NOTFND, and EMPTY labels are used correctly during DEFINE.
TOMANY	An attempt was made to define a fourth target function. The target function is ignored, and processing proceeds with the FINAL statement.

If a routing error occurs during the evaluation of the routing table, MERVA routes the message to the queue specified in the NEXT parameter of the function table entry. If no NEXT parameter is specified, it routes the message to the target queue specified in the TARGET parameter of the TYPE=FINAL statement of the DSLROUTE macro.

Routing Trace DROP Entry

Format:

```
nnn DRP var-name
var-name
```

Explanations:

nnn
Is the number of the routing table entry processed.

DRP
Shows that a DSLROUTE TYPE=DROP was processed.

var-name
Shows the variable field names that are found and dropped. For DROP ALL, it shows which variable names have been defined successfully. For DROP with a list of field names, it shows only those that have been found. When more than two variable field names are dropped, as many continuation trace entries follow as are necessary to show all field names, with the sequence number and the DRP indication omitted.

Routing Trace FINAL Entries

Each routing trace has three final entries:

1. The FIN entry

   Format:

   ```
   nnn FIN orign function
   ```

   Explanations:

   nnn
   Is the number of the FINAL routing table entry.

   FIN
   Shows that a DSLROUTE TYPE=FINAL was processed.

   orign
   Is used only when an error occurs that leads to the use of the NEXT or FINAL function, and *orign* shows this with the words NEXT or FINAL. *Origin* is blank when no errors have been detected.

   function
   Is the target function used from NEXT or FINAL if *orign* is not blank.

2. The summary of target functions

   Format:

   ```
   TARG= target1  target2  target3
         target4  target5  target6
         target7  target8
    target9
         target10 target11 target12
   ```

   Explanations:

   TARG=
   Shows the final summary of routing target functions evaluated during this routing process. All target functions shown here have been shown earlier with SET routing trace entries. Depending on the routing result, none, or one to twelve target functions can be shown indicated by *target1* to *target12*.

3. The return and reason code

   Format:

   ```
   RETCODE = r REASON = rc
   ```

Explanations:

r	Shows the return code of DSLRTNSC. See the *MERVA for ESA Messages and Codes* for the explanation of the return codes.
rc	Shows the reason code of DSLRTNSC. See the *MERVA for ESA Messages and Codes* for the explanation of the reason codes.

Appendix H. 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

Programming Interface Information

This book is intended to help the customer to understand MERVA. This book primarily documents Product-Sensitive Programming Interface and Associated Guidance Information provided by MERVA.

General-Use Programming Interface allow the customer to write programs that obtain the services of MERVA.

However, this book also documents Product-Sensitive Programming Interface and Associated Guidance Information.

Product-Sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-Sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-Sensitive Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section by the following marking:

Product-Sensitive Programming Interface

Product-Sensitive Programming Interface and Associated Guidance Information...

End of Product-Sensitive Programming Interface

Trademarks

The following terms are trademarks of the IBM Corporation in the United States, other countries, or both:

- Advanced Peer-to-Peer Networking
- AIX
- APPN
- C/370
- CICS
- CICS/ESA
- CICS/MVS
- CICS/VSE
- DB2
- Distributed Relational Database Architecture
- DRDA
- eNetwork
- IBM
- IMS/ESA
- Language Environment
- MQSeries
- MVS
- MVS/ESA
- MVS/XA
- OS/2
- OS/390
- RACF
- VSE/ESA
- VTAM

Workstation (AWS) and Directory Services Application (DSA) are trademarks of S.W.I.F.T., La Hulpe in Belgium.

Pentium is a trademark of Intel Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

C-bus is a trademark of Corollary, Inc.

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

- *MERVA ESA Components Client User's Guide*, SH12-6282
- *MQSeries for MVS/ESA System Management Guide*, SC33-0806
- *MQSeries for VSE/ESA System Management Guide*, GC34-5364
- *MVS/ESA SP V5 Writing TPs for APPC/MVS*, GC28-1471
- *MVS/ESA SP V5 Writing Servers for APPC/MVS*, GC28-1472
- *MVS/ESA SP V5 Initialization and Tuning Reference*, SC28-1452.

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
Concepts and Components
Version 4 Release 1

Publication No. SH12-6381-01

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
SH12-6381-01

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

SH12-6381-01

Cut or Fold
Along Line

Program Number: 5648-B29

SH12-6381-01

Spine information:

IBM MERVA for ESA Concepts and Components Version 4 Release 1