LY24-5203-0
File No. S370-30

Program Product

IBM Virtual Machine Facility/370: Remote Spooling Communications Subsystem Networking Logic

Program Number 5748-XP1

IBM

First Edition (March 1979)

This edition applies to Version 1 of the Remote Spooling Communications Subsystem Networking program product (Program Number 5748-XP1) and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters.

Information in this publication is subject to change. Any such changes will be published in new editions or technical newsletters. Before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370 Bibliography of Industry Systems and Application Programs, GC20-0370, and the Technical Newsletters that amend that Bibliography, for the editions and Technical Newsletters that are applicable and current.

The Program Product described in this manual, and all licensed materials available for it, are provided by IBM under the terms of the License Agreement for IBM Program Products. Your local IBM office can advise you regarding the ordering procedures.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Programming Publications, Dept. G60, P.O. Box 6, Endicott, New York, U.S.A., 13760. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Preface

This logic manual describes the internal functioning of the Remote Spooling Communications Subsystem Networking program product. This manual is for IBM program support representatives, and system programmers and analysts responsible for installation, maintenance, and modification of RSCS.

> Note:
> In this manual the term "RSCS" refers to the Remote Spooling Communications Subsystem Networking program product. It does not refer to the Remote Spooling Communications Subsystem component of VM/370. Where these two different programs are discussed together, the difference is made clear.

This manual assists in isolating RSCS module code. It gives:

- An overview of RSCS operations
- Descriptions of RSCS's user functions with reference to the tasks and modules that perform them
- A description of each module's main routines and linkages
- Control flow diagrams of inter-routine inter-task relationships
- Locations and contents of data areas
- An approach to problem determination
- Appendixes with reference material on MULTI-LEAVING and the RSCS Preloader utility under CMS.

These sections document the program logic sufficiently to point to the module listing that the logic manual user needs. Once in a module listing, the user should readily find the logic he is concerned with, using module and subroutine headers (prologues) and the comments in the assembler language statements.

RSCS runs as a virtual machine under the VM/370 Control Program (CP). In extending the VM/370 spooling system capability to include spooling to remote stations, RSCS interacts with the CP spooling system. Therefore, some of the information in this publication requires a knowledge of that area of CP.

Related Publications

IBM Virtual Machine Facility/370: Remote Spooling Communication Subsystem Networking General Information Manual, GH24-5004

IBM Virtual Machine Facility/370: Remote Spooling Communication Subsystem Networking Program Reference and Operations Manual, SH24-5005

IBM Virtual Machine Facility/370: Remote Spooling Communication Subsystem Networking Reference Summary Card, GX24-5119

Virtual Machine Facility/370: System Logic and Problem Determination Guide Volume 1: Control Program (CP), SY20-0886

IBM Data Processing Glossary, GC20-1699

Contents

Illustrations

Section 1: Introduction

RSCS is a software package in the IBM Network Job Interface series of products. Network Job Interface (NJI) is a remote spooling capability comprising several software packages that provide the support for the transmission of files (including jobs and job output data) between processors attached to a telecommunications network. The processors that are nodes in the network can be running the same or different spooling systems, with the common requirement that each runs one of the NJI or NJE (Network Job Entry) support packages.

RSCS is a program product that provides NJI support on those nodes in the network that are VM/370 systems. RSCS is a virtual machine subsystem, operating independently of other virtual machines running under the VM/370 control program, CP. Using the RSCS command language, the RSCS operator manages his node in the network, and can issue some commands to other nodes in the network.

Within the network, there can also be both CPUs and terminals that contain no NJI/NJE support. These do not function as intermediate nodes - they have subhost status - and can, depending upon their configurations, perform operations such as: submit files (jobs); receive files (jobs); or both submit files that are jobs and receive their output.

The processor with an RSCS virtual machine is a relay point, in a network, for data:

- From other virtual machines under its own VM/370 system either to remote processors, or to remote batch stations
- From other nodes to virtual machines on its own processor
- From other nodes to either other nodes or remote batch stations.

The data relay facility of RSCS is provided by its "store and forward" functions, which use the VM/370 CP spooling facility to temporarily hold incoming and outgoing files.

When the data RSCS is transferring is to be forwarded from node to node in the network to the data's ultimate destination, each forwarding node must have the store-and-forward and routing capability provided by one of the NJI or NJE support packages.

When the data that RSCS is transferring is a job to be executed remotely, and if the job output is to be returned to an interactive user or to an indirectly connected system or workstation, the destination system must have the job entry capability provided by one of the NJI or NJE support packages.

The output of a routed job executed at a remote non-VM/370 NJI/NJE system is routed to the real unit record equipment at the submitter's location unless explicitly overridden by the user. The submitter may include in his jobstream NJI control statements to route the remotely executed job output to another destination or destinations.

The output of a routed CMS batch job executed at a remote VM/370 system can be routed to the job's originator by including appropriate CMS commands with the job.

See the RSCS Program Reference and Operations Manual (listed in the Preface) for detailed information about using RSCS.

A given RSCS location (or "node") sees only the nodes adjacent to it in the network. See Figure 1-1, below. An RSCS "link" is defined at an RSCS location as the capacity to communicate with a particular remote location via a direct connection. This includes dialup, leased line, and channel-to-channel adapter (CTCA) connections.

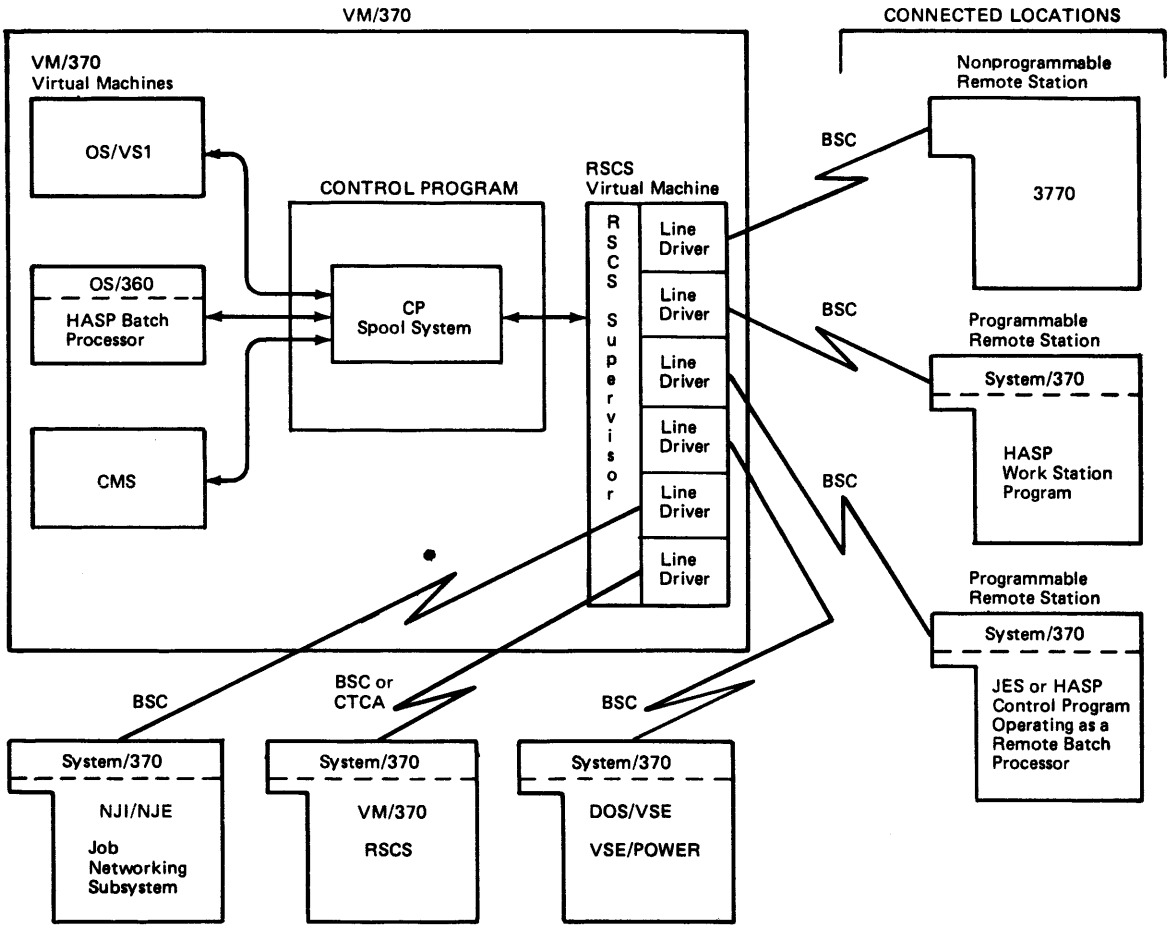


Figure 1-1. RSCS - Sample Network

RSCS Overview

Figure 1-2 is an overview of RSCS operations in a simple network. (Not all network nodes must be VM/370 RSCS nodes; this figure shows the types of operations that RSCS supports.) Refer to Figure 1-2 when reading the following description.

User Archie on the CMS system at VM1 sends a file to user Bob on the CMS system at VM3. Archie issues the commands:

```
SP 00D TO RSCS1
TAG DEV 00D VM3 BOB
PUNCH filename filetype
```

CP spools his virtual punch 00D output to RSCS1. RSCS1 may have any number of links to the other network nodes, but it has locally-specified tables that indicate that files for VM3 are to go to VM2. RSCS1

forwards the spool file, including its destination information, to VM2. RSCS2 on VM2 receives the file, spools it to its own userid, and forwards it to VM3. RSCS3 at VM3 recognizes that the file is for its own location and spools it to userid Bob.

These functions are explained in more detail in Section 2.

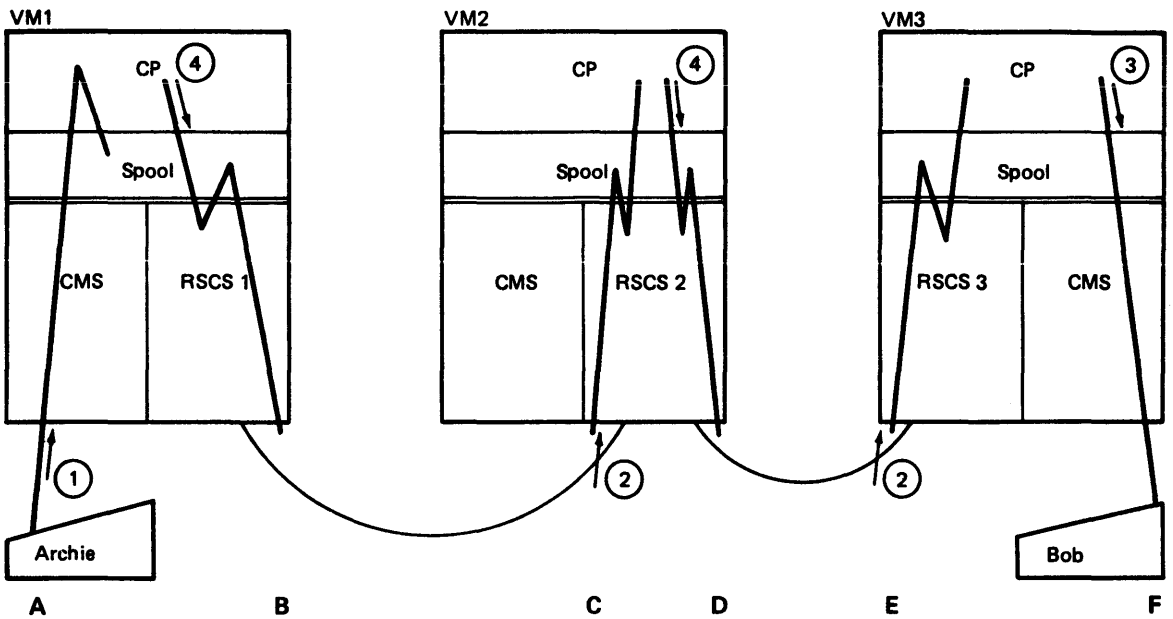


Figure 1-2. RSCS File Handling

The RSCS Virtual Machine and the VM/370 Control Program (CP)

Like other VM/370 virtual machines, the RSCS virtual machine runs under the control of CP. In extending the VM/370 spooling system capability to include spooling to remote stations, RSCS interacts with the CP spooling system. Therefore, some of the information in this publication requires a knowledge of that area of CP.

The RSCS virtual machine consists of the virtual machine operator's console, an RSCS system disk, and attached telecommunications lines. During system initialization, a virtual card reader is defined for the RSCS virtual machine, but this reader does not exist in the CP directory entry for the RSCS virtual machine.

Virtual printers, card punches, and readers are defined dynamically by the program as they are needed. For example, when a file from a remote station is transmitted to RSCS, a virtual punch is defined to accept the file. Similarly, virtual readers are defined when RSCS accepts a spool file to transmit. RSCS virtual storage also dumps onto a virtual printer when abnormal termination of RSCS occurs.

The minimum virtual storage required to run RSCS is 384K, with typical requirements falling in the 512K to 1M range.

Remote System Routes and Links

At a local installation there are a number of transmission paths to remote stations. A unique location identifier (locid) is assigned to each remote station.

For each direct transmission path (nonswitched line) or potential direct transmission path (switched line) to an adjacent network node, a link must be defined at the installation. Each such link is given a name (linkid) that must be the same as the location identifier of the remote system or station to which the direct transmission path leads.

Every successive node along the route connecting nodes that are destinations for data must have prespecified routing information. The routing information at a given node specifies, for each possible destination locid, the link that this node is to use to forward the data.

The links and routes can be defined (a) permanently, in a CMS file on the RSCS system disk called RSCS DIRECT; or (b) temporarily, by the RSCS operator commands DEFINE and ROUTE. Temporary definitions disappear at the next RSCS IPL.

Links and routes can be temporarily removed with the RSCS operator commands DELETE and ROUTE. Any permanently defined links and routes removed by operator commands reappear at the next RSCS IPL.

The alternate path facility of RSCS enables installations to configure their routes and links so as to bypass unavailable links. Refer to Figure 1-3 when reading the following description.

Any node (A) may have both a LINK table entry and a ROUTE table entry for an adjacent node (B). A file that has a final destination of node B, and is being forwarded by node A, will normally be enqueued on the node B link, link B. The ROUTE table entry is used when the final destination of the file at node A is the adjacent node B, but link B to node B is not available. The ROUTE table entry for node B is an alternate path that specifies node C via link C. Node A transmits the file to node C, bypassing the unavailable link. See the RSCS Program Reference and Operations Manual for use of this alternate path facility.

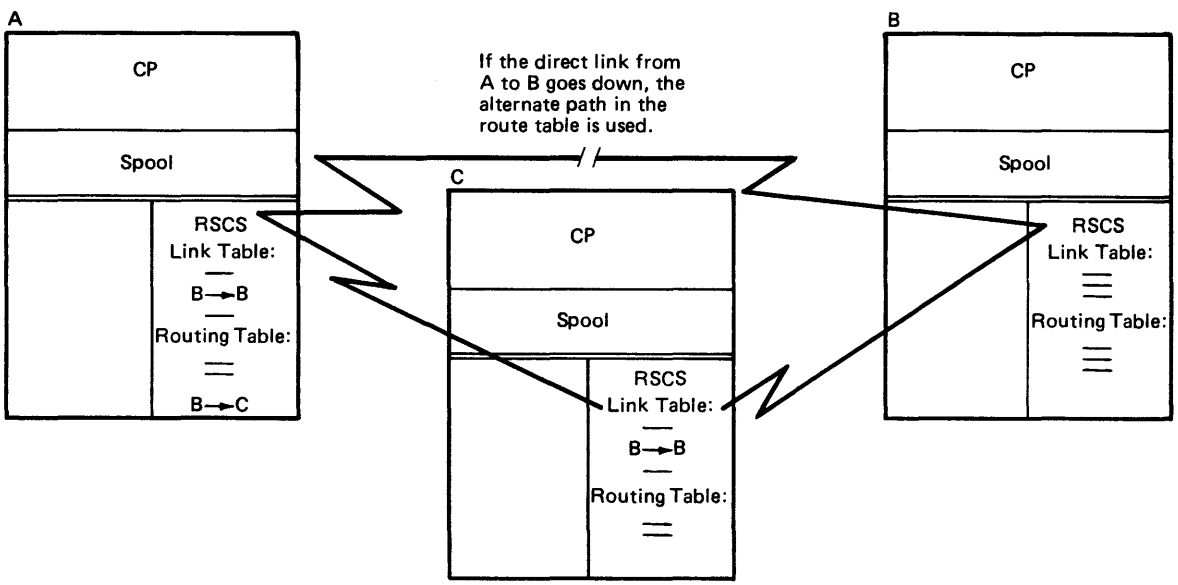


Figure 1-3. Alternate Path Facility

Remote Stations and Systems

Remote stations are configurations of I/O devices attached to a VM/370 system by binary synchronous communications (BSC) switched or nonswitched lines. RSCS supports both programmable and nonprogrammable remote stations.

PROGRAMMABLE REMOTE STATIONS

Programmable remote stations, such as the IBM System/3 and System/370, are processing systems with attached BSC adapters. These systems must be programmed to provide the MULTI-LEAVING line protocol necessary for their devices to function as remote stations. This programming support is provided by a remote terminal processor (RTP) program generated according to HASP workstation protocol and tailored to the system's hardware configuration. Certain programmable remote stations like the System/3 can only be programmed to function as remote terminals. Others, like the System/360 and System/370, can function either as remote terminals or as host batch systems using RSCS as a remote job entry workstation. Both of these types of remote stations are managed by the spool MULTI-LEAVING (SML) line driver of RSCS.

NONPROGRAMMABLE REMOTE STATIONS

Nonprogrammable remote stations are I/O configurations that cannot be programmed, but are designed to provide the line protocol necessary for them to function as remote stations. They can receive, read, print, punch, and send files. An example of a nonprogrammable remote station is a 3780 Data Transmission Terminal. Nonprogrammable remote stations are managed by the NPT (Nonprogrammable Terminal) RSCS line driver.

REMOTE STATIONS SUPPORTED BY RSCS

The types of devices supported for all types of remote stations, programmable and nonprogrammable, are listed in the RSCS Program Reference and Operations Manual.

REMOTE SYSTEMS SUPPORTED BY RSCS

The programmable remote systems that RSCS can communicate with are:

- Other RSCS systems
- VNET PRPQ systems
- HASP Networking PRPQ systems
- ASP Networking PRPQ systems
- Network Job Entry Facility for JES2
- JES3 Component of OS/VS2
- RES Component of OS/VS1
- VSE/POWER systems

Network Control: RSCS and VM/370 Commands

Both RSCS and VM/370 commands are used to control RSCS. RSCS commands are used by an RSCS operator to control the network; VM/370 CP and CMS commands are used by virtual machine users who use RSCS.

RSCS OPERATOR COMMANDS

To manipulate the file being transmitted across the network and to communicate with the various network users, RSCS provides a command language. Figure 1-4 lists the RSCS commands and the functions they perform. Detailed descriptions of these commands are in the RSCS Reference and Operations Manual.

The operator may enter RSCS commands described in Figure 1-4 at the RSCS virtual machine console. A subset of the RSCS command language may be entered by operators of remote stations or remote systems.

VM/370 CP AND CMS COMMANDS FOR RSCS

The VM/370 CP TAG and SPOOL commands specify a device to be spooled, and they associate a destination location identifier (locid) with that device. SPOOL directs the file to the RSCS virtual machine. The CP CLOSE command or the CMS PRINT or PUNCH commands close the file and transfer it to the RSCS virtual machine.

Data specified by the CP TAG command controls processing of files transmitted across the network. When a VM/370 user creates, on his virtual machine, a file to be transmitted to a remote station via RSCS, the CP TAG command must contain information needed by RSCS.

A virtual machine user may use the CP SMSG (Special Message) command to send messages via RSCS to remote virtual machines or to request status information about a local or remote RSCS or about a remote VM/370 system. The text of an SMSG command can be an RSCS MSG command or an RSCS CMD command; the text of that CMD command can be an RSCS QUERY command or an RSCS CPQUERY command.

For details on how to use the CP SPOOL, TAG, and SMSG commands, see the RSCS Reference and Operations Manual.

Command Name	Function
*	Comment following asterisk prints out on RSCS operator console, and no function is performed. (Useful for CMS EXEC files of RSCS commands, particularly the PROFILE RSCS file.)
BACKSPAC	Restarts or repositions in a backward direction the file currently being transmitted.
CHANGE	Alters one or more attributes of a file owned by RSCS.
CLOSE	Deactivates partially processed files on an inactive link. Discards output (incoming) files. Reenqueues active input files as inactive files.
CMD	Forwards a command line to a remote system for execution.
CP	Executes a command line as a VM/370 Control Program (CP) console function.
CPQUERY	Requests status information from CP, similar to a VM/370 CP QUERY command.
DEFINE	Temporarily adds a new link definition to the RSCS link table, or temporarily alters an existing link definition.
DELETE	Temporarily deletes a link definition from the RSCS link table.
DISCONN	Places RSCS in disconnect mode and optionally directs RSCS operator console output to another virtual machine.
DRAIN	Quiesces file transfer and deactivates an active communication link.

Figure 1-4. RSCS Commands and Their Functions (Part 1 of 2)

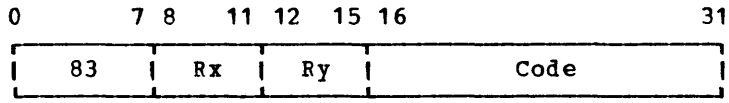
Command Name	Function
EXEC	Executes series of RSCS commands contained in the specified user-built CMS file (filetype: RSCS).
FLUSH	Discontinues processing the currently active file on the specified link.
FORCE	Immediately deactivates an active link, without quiescing file transfer.
FREE	Resumes transmission on a communication link previously in HOLD status.
FWDSPACE	Repositions in a forward direction the file currently being transmitted.
HOLD	Suspends file transmission on an active link without deactivating the link.
HT	Flushes out all messages presently awaiting printing on the RSCS operator console or remote station operator console.
MSG	Sends a console message line to a local or remote operator or user.
ORDER	Reorders files enqueued on a specific link.
PURGE	Removes and discards all or specified inactive files from a link.
QUERY	Requests system information for a link, a file, or for the system in general.
REORDER	Sorts and reorders all files enqueued for all links.
ROUTE	Adds, deletes, or alters an RSCS routing table entry.
START	Activates a specified communication link.
SHUTDOWN	Issues DRAIN to all active links.
TRACE	Monitors line activity on a specified link.
TRANSFER	Changes the destination address for specified files.

Figure 1-4. RSCS Commands and Their Functions (Part 2 of 2)

CP INSTRUCTIONS USED BY THE RSCS CONTROL PROGRAM: DIAGNOSE

When RSCS handles files being transmitted across the network, the RSCS control program and line driver tasks issue DIAGNOSE instructions to obtain CP services.

A DIAGNOSE instruction is used in VM/370 for communication between a virtual machine and CP. The machine-coded format for the VM/370 usage of the DIAGNOSE instruction is:



Content	Explanation
83	DIAGNOSE operation code
Rx	User-specified register number
Ry	User-specified register number
Code	Hexadecimal value that selects a particular CP function.

Figure 1-5 lists the DIAGNOSE function codes used by RSCS, the functions of those codes, and the RSCS modules from which they are issued.

RSCS Preparation and Startup

RSCS PREPARATION

The preparations for running RSCS on a VM/370 system are explained in VM/370: RSCS Networking Program Reference and Operations. In summary, the preparations are:

- Load RSCS modules onto disk.
- If required, perform updates to modules.
- If required, use the Preloader utility (see Appendix B) to combine and link multiple modules.
- Use the CMS editor to build the directory file (RSCS DIRECT) to specify parameters for the RSCS virtual system. These specifications include installation variables and link and route definitions describing communication paths to remote locations.
- If desired, use the CMS editor to build the PROFILE RSCS file to specify installation STARTUP commands.

DYNAMIC DIRECTORY

There is no system generation (requiring assembly) for RSCS.

Each time RSCS is IPLed it dynamically configures itself, referring to the contents of a file (named RSCS DIRECT) on the RSCS system disk. The local system programmer builds and updates this file. This file specifies the RSCS configuration and characteristics desired for this location. The file can be updated to incorporate new link specifications, new routing table entries, etc. as desired even during RSCS operation; the changes become effective on the next RSCS IPL.

During operation, the RSCS operator can enter commands that immediately modify some of the RSCS DIRECT-specified characteristics. But these commands (such as ROUTE, LINK, DEFINE, and DELETE) are in effect only

temporarily; when RSCS is re-IPLed, only the RSCS DIRECT statements remain in effect.

The details of the contents of RSCS DIRECT are in the RSCS Program Reference and Operations Manual.

DIAGNOSE Code	Function	Issued by Module(s)
0000	Gets the userid of the RSCS virtual machine.	DMTIRX
0008	Executes a CP command. Results may be returned in a buffer or sent to the virtual console.	DMTAXM DMTCMX DMTMGX DMTREX
000C	Gets the current time and date.	DMTAXA DMTNCM DMTNPT DMTSML DMTVMB DMTVMC DMTPOW
0010	Frees virtual storage page.	DMTASK DMTCOM
0014	Manipulates input spool files.	DMTAXM DMTNCM DMTNPT DMTSML DMTVMB DMTVMC DMTPOW
0020	Performs DASD I/O without interrupt.	DMTINI
0024	Determines virtual device type information.	DMTIRX DMTLAX DMTNCM DMTREX DMTSML DMTPOW
004C	Generate an RSCS accounting record.	DMTAXA
005C	Edits RSCS messages.	DMTMGX
0068	CP vehicle for sending Special Messages to RSCS.	DMTIRX DMTREX

Figure 1-5. VM/370 DIAGNOSE Instructions Issued by RSCS

Section 2: Method of Operation

The RSCS Control Program

RSCS is a virtual machine subsystem, with a multitasking supervisor (MSUP) that manages multiple independent programs called tasks.

The MSUP supervisor is the heart of the RSCS virtual machine. It is a set of routines and storage areas that coordinate the operation of RSCS.

The tasks are modules or sets of modules that perform RSCS functions. The task modules are executed under control of the MSUP supervisor.

The supervisor provides only those functions that cannot be consistently provided by the tasks themselves; that is, the supervisor provides only the support needed to control and coordinate the execution of the tasks.

The two types of RSCS tasks are system service tasks and line driver tasks. System service tasks provide the system support functions for the supervisor and for other tasks. Line driver tasks manage the transmission paths to remote systems and stations, and interact between the remote stations and the system service tasks and the supervisor. Each line driver task manages transmission to and from one remote station.

The figures in Section 3 show the communication paths between the MSUP supervisor, system service tasks, line driver tasks, remote stations, and VM/370 virtual machines.

MSUP: THE RSCS SUPERVISOP

The MSUP supervisor is a set of service routines that provide functions for the tasks that run under them. These service routines may be called by any task. In general, they provide four kinds of services:

- Task management
- I/O management
- Interrupt handling
- Virtual storage management

TASK MANAGEMENT

The task management service routines provide task initiation and termination, task dispatching, task-to-task communication, and task synchronization.

Task initiation consists of making a task that has been loaded into virtual storage available for execution. This includes:

(a) Building a TASKE entry for the task in the task queue pointed to by the SVECTORS field, TASKQ (X'228'); and

(b) Marking the task dispatchable so that its initialization routine is entered.

In general, the only task to request task initiation and termination is the REX system control task, which is described below.

Task dispatching consists of scanning the task queue entries (TASKE) for tasks that are able to continue executing, and allowing them to execute one at a time.

The two types of task-to-task communications are (1) the DMTSIG routine (ALERT) and (2) the DMTGIV and DMTAKE routines (GIVE/TAKE).

The DMTSIG routine allows a task to immediately invoke another task to pass it information. The interrupted task must have an asynchronous exit routine defined to handle the interruption. Functionally, DMTSIG performs a function analogous to an SVC instruction.

The DMTGIV and DMTAKE routines allow tasks to exchange information buffers with other tasks. The GIVE/TAKE function provides organized enqueuing and delivery of requests for services or information from one task to another. This function is roughly analogous to write-read I/O operation.

Task synchronization involves making tasks ready or not ready for execution under control of the dispatcher. When a task requests the services of another task, the requesting task may suspend its execution while the request is being processed. The synchronization mechanism consists of two routines, DMTWAT and DMTPST. DMTWAT causes the requesting task to temporarily halt execution. DMTPST causes a temporarily-halted task to resume execution. For more information on task synchronization refer to the section "Task Synchronization".

A task that requests supervisor service by branching to a supervisor routine is placed in non-executing mode with a FREEZE SVC, which is issued by the called supervisor routine.

Dispatching in RSCS

The supervisor returns control to the tasks by means of the dispatcher (DMTDSP). The dispatcher scans the queue of tasks to be executed (TASKE in SVECTORS), selects the first dispatchable task element (that is, one that is not marked nondispatchable by DMTWAT), moves this task element to the end of the task queue, and restarts its execution. If no task element is dispatchable, a masked-on wait state PSW is loaded by the dispatcher.

To suspend its execution, the requesting task calls DMTWAT, which inspects the synchronization locks RSCS uses to synchronize task execution. Completion of a service is signalled by means of a synch lock, which is set (or "posted") by DMTPST.

Task-to-Task Communications

Sometimes a task requires the services of another task to complete a function. For example, VMB may require that AXM open a file for input before VMB processing can continue. RSCS tasks communicate with each other to request these kinds of services using two methods: ALERT task-to-task communication, and GIVE/TAKE communication.

Both methods use an element, which is a table of information that describes the nature of the request. In general, these elements are called "request elements" and "alert elements".

The ALERT mechanism is an asynchronous interrupt that causes the requested task to examine the request immediately and uninterruptably. When the requested task processing is complete, control is passed to the MSUP dispatcher. The requestor task remains dispatchable.

The GIVE/TAKE mechanism allows the requested task to service the request in the course of normal task dispatching. The requesting task remains dispatchable unless it requests a WAIT on the GIVE synch lock.

RSCS has a three-level task hierarchy implemented in task programming, but not checked or controlled by the MSUP supervisor. The hierarchy is: (a) the REX task is highest, (b) the AXS and LAX tasks are next, and (c) the line driver tasks are lowest.

This hierarchy is followed in task-to-task communications. A task issues an ALERT to only a lower task. A task issues a GIVE to only a higher task. A task never communicates directly with a task equal to it in the hierarchy.

GIVE/TAKE Synchronous Task-to-Task Communication

The GIVE/TAKE method provides ordered enqueuing of requests for services. Such a request is handled when the servicing task is free to handle it, rather than upon immediate demand. Figure 2-1 and the following descriptions explain the GIVE/TAKE task-to-task communication process.

REQUEST AND RESPONSE ELEMENTS: Generally, request and response elements are tables of information that reside in the storage of both the requesting task and the task providing the service. During task-to-task communication, these elements are passed from one task to another, containing either requests for services or responses to requests.

GIVE TABLES: When a task requests a service of another task via GIVE/TAKE, it builds a GIVE table, a GIVE request buffer, and a GIVE response buffer in its storage. (The request and response buffers may be at the same location in storage.)

The GIVE request buffer contains a GIVE request element (a table of information describing the service requested). After the GIVE request element is built, the requesting task clears the synch lock in its GIVE table to zero (in preparation for a call to DMTWAT) and specifies the address of the GIVE table in a call to DMTGIV.

SUPERVISOR HANDLING OF GIVE REQUESTS: The supervisor routine DMTGIV then builds and enqueues a supervisor GIVE element containing a pointer to the GIVE table, so that the request can be forwarded to the receiving task when that task is ready to accept the request.

TAKING A GIVE REQUEST: When the receiving task is ready to process a GIVE request, the receiving task prepares a TAKE table in its own storage. The TAKE table consists of a field to receive the task name of the requesting task and the addresses and the lengths of a TAKE request buffer and a TAKE response buffer. Functionally, these buffers complement the GIVE request and response buffers and, like the GIVE buffers, may be at the same location in storage.

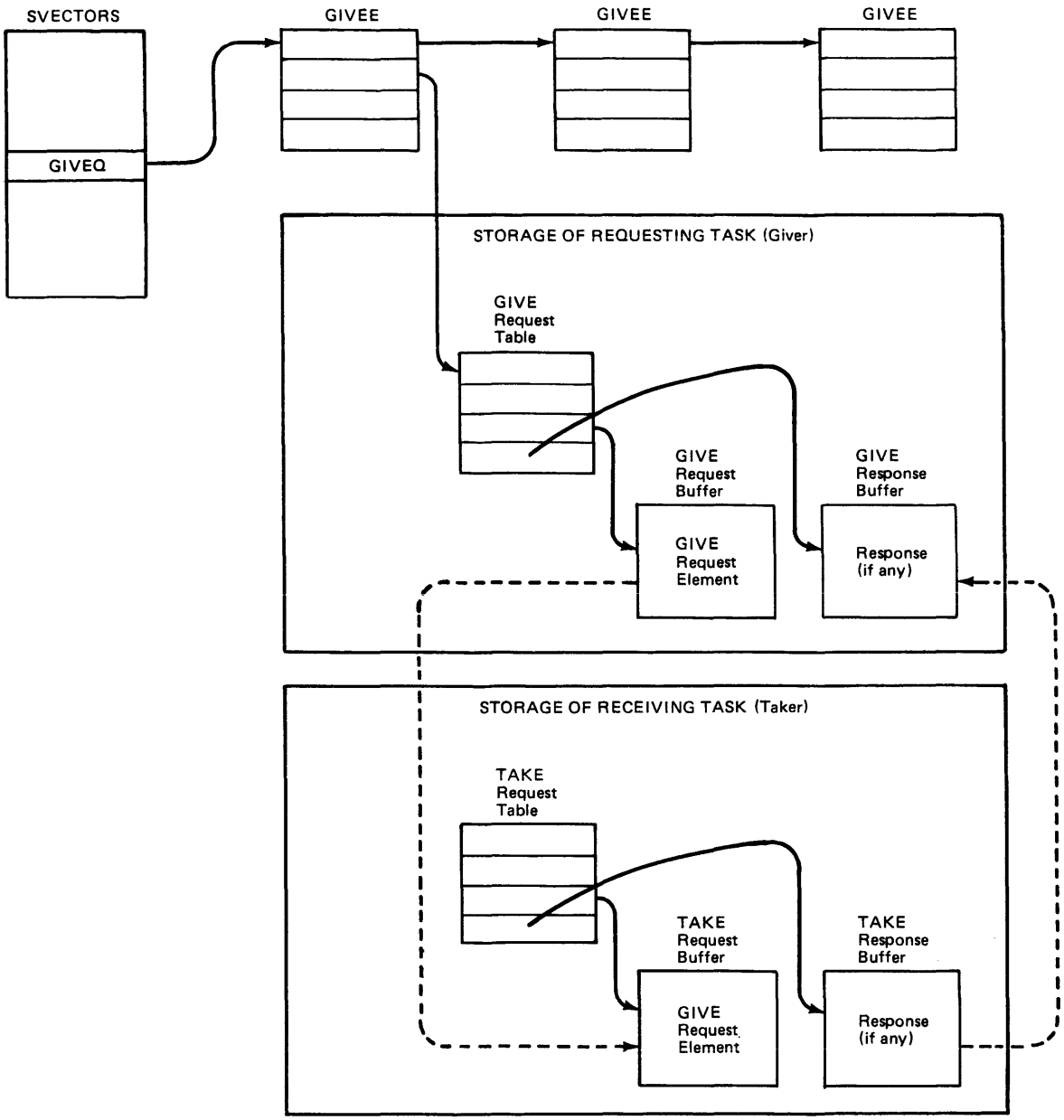


Figure 2-1. Movement of Data During a Typical GIVE/TAKE Transaction

After the TAKE table is built, the receiving task specifies the address of the TAKE table in a call to DMTAKE. The supervisor then moves the GIVE request buffer (containing the GIVE request element) to the receiving task's TAKE request buffer.

RESPONDING TO A GIVE REQUEST: The receiving task performs the requested service, and may update the GIVE request element and place it in its TAKE response buffer. This modified GIVE request element contains information on results of request processing to be returned to the requesting task.

When all requested processing is complete, the receiving task again calls DMTAKE, specifying the address of the TAKE table. The supervisor

responds by immediately moving the contents of the receiving task's TAKE reponse buffer to the requesting task's GIVE response buffer, and posting the synch lock in the requesting task's GIVE table.

MULTIPLE GIVE REQUESTS FOR THE SAME TASK: If another GIVE request addressed to the receiving task has been enqueued, it is given to the receiving task as described above, and dispatched task execution is resumed. On each call to it, DMTAKE first responds to a previously accepted GIVE request (if one exists) and then gives another modified GIVE request element back to the requesting task (if one exists).

WAITING FOR REQUEST COMPLETION: The requesting task waits for request completion by specifying the address of the synch lock in its GIVE table in a call to the WAIT routine (DMTWAT).

The receiving task waits for request availability by calling DMTWAT and specifying the address of its take request synch lock, which is located in its Task Save Area. The take request synch lock is cleared to zero by DMTAKE when no GIVE request address to the calling task remains enqueued. It is posted by DMTGIV when such a request is enqueued as a result of DMTGIV processing for another task.

Synchronization Locks

Synchronization locks (or synch locks) are fullwords in task save areas or control tables (such as TAREA or IOTABLE). Synch locks are also in control areas in function selector routines.

A synch lock is a location in storage where a task that requests a service from another task receives notice of the completion of that service. Task code may contain any number of synch locks, depending on the number and types of service it needs.

The addressability of each synch lock for posting upon completion of the requested service depends upon the type of request. For example, the GIVE/TAKE request synch lock is a fixed location. It is in the Task Save Area, TAREA, at X'48' bytes from the start of the task. The I/O synch lock, however, is at the first of the requesting task's I/O table, whose address is passed with the requesting task's BALR to the I/O request handler (see "Handling I/O Requests").

The synch lock must be set to zero before the request for services is made. Setting the synch lock to zero prepares it for processing by the request servicer and the WAIT routine.

The requesting task (the caller of DMTWAT) may specify the address of a single synch lock (as in the case of a GIVE Table or an IOTABLE), or the address of a list of synch locks, one of which must be posted by DMTPST before dispatching of the requesting task can resume. Figure 2-2 shows the contents of Register 1 on a call to DMTWAT.

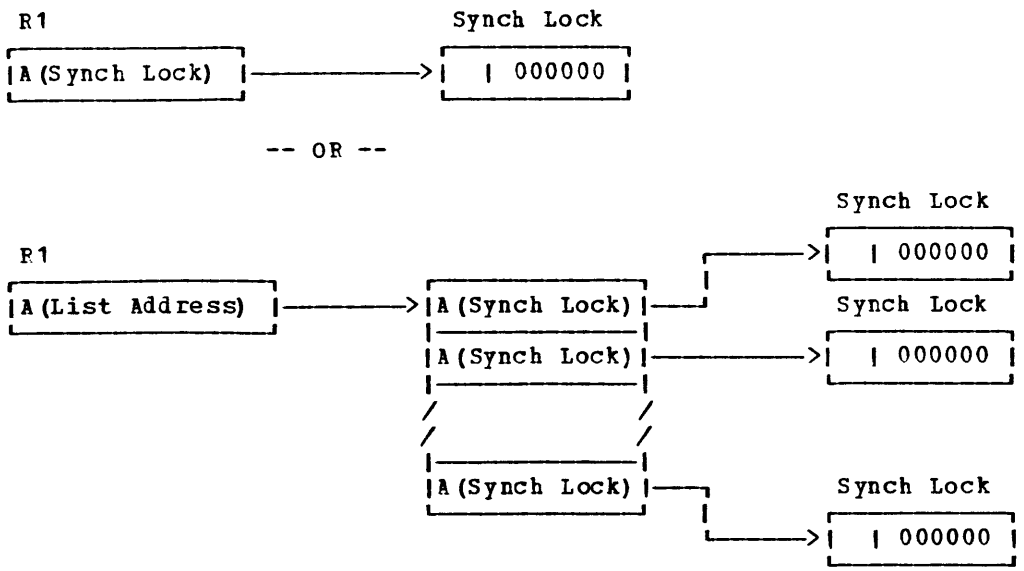


Figure 2-2. Input to the DMTWAT Routine

Posting a Synch Lock

When the requested service is complete, the receiving task signals completion by calling the POST routine (DMTPST), specifying the requesting task's associated synch lock. The POST routine sets the high-order byte of the synch lock to nonzero. This is referred to as "posting" that synch lock, and indicates that the requested service is complete.

If the low-order bytes of the synch lock contain an address indicating that the task issued a WAIT on this synch lock, the POST routine also marks the requesting task dispatchable in its request element TASKE in the TASKQ, and sets the last three bytes of the synch lock to zero.

Waiting For GIVE/TAKE Requested Services: DMTWAT

Before a task can use the results of requested service, it must ensure that the service has been performed. The requesting task signals that it is waiting for completion of a service via a call to the supervisor routine DMTWAT, specifying the synch lock associated with the requested service.

If the high-order byte of the task's synch lock is nonzero when DMTWAT inspects it, control is returned directly to the requesting task. If the high-order byte of the synch lock is zero, DMTWAT marks the requesting task nondispatchable (in the task's request element, TASKE), stores the address of the task's request element in the low-order bytes of the synch lock, and resumes dispatching for other tasks.

Tasks may call DMTWAT specifying multiple synch locks. This is an "OR" condition; the task wants to run if any of the specified synch locks gets posted. Upon such a call, each synch lock is inspected and, if any synch lock is posted, task execution resumes immediately (the wait is

satisfied). If none of the synch locks is posted, the task element for the calling task is marked nondispatchable, its address is stored in each synch lock, and dispatching is resumed for other tasks.

When any synch lock in the list is posted, the task element is marked dispatchable. The dispatcher clears the low-order three bytes of each of the task's synch locks before task execution is resumed.

Asynchronous Interruptions and Exits

Asynchronous interruptions are unpredictable and must be handled as they occur. The three kinds of asynchronous interrupts are I/O, alert, and external.

Any RSCS task that has code (called an asynchronous exit routine) to process asynchronous interrupts of a given type notifies the supervisor, usually during task initialization, by a branch to the supervisor DMTASY module. This routine builds an entry in the appropriate asynchronous interrupt queue that is scanned for "takers" whenever an asynchronous exit condition arises.

Asynchronous exits are provided for external interruptions, for certain I/O interruptions, and for ALERT requests that occur during execution of another task.

Asynchronous exit routines in RSCS tasks perform limited function, often enqueuing requests for further processing at a later time by dispatched tasks. When the asynchronous exit routine completes processing, it returns control to the supervisor, which then resumes dispatching tasks via a call to the dispatcher (DMTDSP).

ALERT Asynchronous Task-to-Task Communication

The ALERT method of task-to-task communication allows a task to immediately invoke a service in another task, bypassing the normal task selection mechanism.

Initially, a task that services alert requests issues an asynchronous exit request for alerts. The request specifies the address of its routine, called an asynchronous exit, that will process alerts to this task. The asynchronous request processor (supervisor module DMTASY) takes control, and records this information in an alert asynchronous exit queue element. The requesting task is made dispatchable to enable it to continue processing beyond its asynchronous exit request.

When a task requires a certain task's alert service, it issues an alert request by branching to DMTSIG, specifying the task to be alerted, and the address of its own alert element, if any, which specifies a request for processing by the alerted task.

The type of request is described in the ALERT element. The contents of the alert element depend on the type(s) of alert request defined by the alerted task. See Section 5 for alert element formats.

The supervisor responds by giving control to the alert asynchronous exit routine of the specified task and by passing to that task the address of the requesting task's ALERT element.

The asynchronous exit routine responds immediately and may copy the

ALERT element into its own storage for further processing. The asynchronous exit routine then returns control to the supervisor, which allows the dispatched task to resume execution. The following summarizes the terms used in the ALERT capability:

The task that will handle ALERT requests via its asynchronous exit initiates this process with an asynchronous interrupt request.

The request for the services of another task's ALERT exit routine is called an ALERT request.

RSCS TASK DESCRIPTIONS

As described previously, the MSUP supervisor is a set of routines that manage RSCS processing. The supervisor is a nucleus that supports the RSCS tasks. (These tasks are analogous to virtual machines under VM/370 CP, and tasks under OS/VS systems.)

The RSCS system service tasks perform less generalized common functions for the system than those functions performed by MSUP. For example, the AXS system service task coordinates common task access to the VM/370 spool file system. Each line driver task manages the communication with the single remote system or station that its link is defined to provide.

The supervisor gives equal priority to all RSCS tasks, and makes no distinction between system service tasks and line driver tasks. Figure 2-3 lists the RSCS tasks and the service each performs.

REX Task Module Functions

CREATE SYSTEM TASKS: DMTCRE The main system service task, REX, is loaded with the supervisor during RSCS initialization. The REX task, in turn, creates other tasks required by the system. DMTCRE reads these other tasks from a CMS disk by means of a CMS read access method, located in DMTCOM. The task is then started as a new active task under RSCS.

PROCESS COMMANDS: DMTCMX DMTCMX receives commands by either GIVE request elements passed by line driver tasks, or from console-entered commands (via DMTREX).

Task Name	Module Name	Function
REX	DMTREX	Handles RSCS operator console I/O; accepts requests for services passed by other system service tasks or line driver tasks; terminates a task; handles program check interruptions.
	DMTCRE	Loads and creates a system service or line driver task.
	DMTCMX	Interprets RSCS command lines, and either executes the commands or forwards command request elements to line drivers for further processing or transmittal.
	DMTMGX	Builds a message line, and distributes the constructed message for delivery or forwarding to the appropriate recipient(s).
	DMTRGX	Handles command and message routing request elements.
	DMTCOM	Comprises a number of independent re-entrant subroutines which may be called by any task.
AXS	DMTAXM	Provides the interface to the VM/370 spool system.
	DMTAXA	Provides accounting interface.
LAX	DMTLAX	Manages communication port allocation.
Line Driver	DMTSML	Manages a telecommunications port for a programmable remote station using RTAM.
	DMTPOW	Manages a telecommunications port for a VSE/POWER system.
	DMTNPT	Manages a telecommunications port for a nonprogrammable remote station terminal.
	DMTVMB	Manages a BSC telecommunications port for a VM/370 to VM/370 link.
	DMTVMC	Manages a channel-to-channel VM/370 to VM/370 link.
	DMTNJI	Comprises the following three modules; manages a BSC telecommunications port for a VM/370 to OS/VS NJI/NJE link.
	DMTNCM	Manages the communications adapter for communication with an NJI/NJE subsystem via BSC or CTCA.
	DMTNHD	Processes network header records.
	DMTNIT	Performs DMTNJI initialization.

Figure 2-3. RSCS Tasks

The commands *, DEFINE, DELETE, DISCONN, HT, EXEC, ROUTE, CP, CPQUERY, FORCE, QUERY, SHUTDOWN, and START (for inactive links) are executed by DMTCMX. Executing these commands generally involves referencing and modifying system status tables (SVECTORS, TTAGQ, TLINKS, etc.).

If the command is not one that DMTCMX executes within its own code, DMTCMX examines the command line for syntax errors and then passes it to the appropriate task for execution. To do this, DMTCMX generates a formatted table called a command alert element to be passed to another active task for execution via an ALERT asynchronous exit.

The commands CHANGE, CLOSE, ORDER, TRANSFER, REORDER, and PURGE are executed by DMTAXS; the commands BACKSPAC, CMD, DRAIN, FLUSH, FREE, FWDSPACE, HOLD, MSG, TRACE, and START (for active links) are executed by the line driver task for the specified link.

PROCESS MESSAGES: DMTMGX DMTMGX manages distribution of all RSCS messages, which may be generated by REX or by any other RSCS task. Each message to be issued is presented to DMTMGX (via GIVE/TAKE for tasks other than REX) along with an internal routing code and an internal severity code.

Messages may be addressed to the local RSCS operator console, to the local VM/370 operator, to a local VM/370 user console, to a remote station operator, or to any combination of these destinations, by the routing code. The severity code is defined for each message, and is an indication of the importance of the message.

Messages for the RSCS local operator console are enqueued for output on the RSCS virtual machine console. Messages for the local VM/370 system operator and for local virtual machine consoles are issued by executing a VM/370 CP MSG command (through the DIAGNOSE 8 interface). Messages for remote RSCS operators are presented to the line drivers for the associated links by the RSCS MSG alert element interface.

TERMINATE SYSTEM TASKS AND HANDLE PROGRAM CHECKS: DMTREX When a line driver task requests termination, a TAKE request is passed to DMTREX specifying that function. DMTREX marks the task as terminated, then searches for active I/O associated with the task. If active I/O is found, it is terminated. To ensure that system integrity is maintained during the termination of the I/O, a mechanism (at label QUIESE) handles situations in which an HIO (Halt I/O instruction) does not take effect immediately.

All RSCS program checks are handled by DMTREX. Program check diagnostic information is dumped, a message to the operator is issued, and the RSCS system status is modified, depending on the nature of the program check. If the program check occurs in a line driver, the line driver execution is terminated and its link is deactivated. Otherwise, RSCS automatically shuts down.

ROUTING OF COMMANDS AND MESSAGES: DMTRGX DMTRGX processes command routing request elements and message routing request elements. Routing request elements received from remote systems are passed to DMTRGX by the receiving line driver. If the routing request element is addressed to the local location, commands are passed to DMTCMX for processing and messages are passed to DMTMGX for distribution to recipients.

Common Supervisor Routines: DMTCOM

DMTCOM contains various routines used by RSCS tasks and other supervisor routines. The address of a vector table, which points to the individual DMTCOM routines, is located at address X'0280' in storage. These routines and their functions are descibed in Figure 2-4.

Communicate with the VM/370 Spool File System: DMTAXS

DMTAXS is RSCS's interface to the VM/370 spool system. When a spool file arrives at the RSCS virtual machine, AXS receives the associated asynchronous interrupt, reads and interprets the file's VM/370 spool file block (SFBLOK) and TAG, enqueues the file for transmission as appropriate, and notifies the appropriate line driver of the new file's availability. AXS provides a GIVE/TAKE request interface to line driver tasks for spool file data input and output, and defines and detaches virtual spool I/O devices as needed. Also, AXS provides an interface to

Routine Name	Vector Table Pointer Name	Function
GETLINK	GLINKREQ	Get Link Table entry
GETROUTE	GROUTREQ	Get Routing Table entry
GETPAGE	GPAGEREQ	Get page of virtual storage
FREEPAGE	FPAGEREQ	Free page of virtual storage
MFI	PMSGREQ	Place entry in message stack
MFO	GMSGREQ	Remove entry from message stack
TODEBCD	GTODEBCD	Convert S/370 TOD clock to EBCDIC
TODS370	GTODS370	Convert EBCDIC clock value to S/370 form
RCMSOPEN	CMSOPEN	Open a CMS file for input
RCMSGET	CMSGET	Read next record of CMS file
GETSUPAG	GPAGESUP	Allocate page of virtual storage for supervisor use

Figure 2-4. DMTCOM Routines

DMTCMX for second-level command execution support.
AXS maintains a queue of a fixed number of virtual storage elements (called tag slots) that describe files currently owned by the RSCS virtual machine. To maintain RSCS integrity in a simple way when a very large number of files is enqueued on the RSCS virtual machine, the virtual storage tag queue is not extended during execution.

When a new file arrives at the RSCS virtual machine, its destination locid is examined, and it is accepted only if there is a matching link or route linkid for which there is a free tag slot available. If the file's destination locid is not defined as a linkid, or as an indirect path through the routing table, the file is returned to the user and the originating user is notified of the action. If there is no free tag slot available for a defined linkid, the file is left "pending", and is accepted when a tag slot becomes free. While a file is pending, it is not recognized by the RSCS command processors, and cannot be referenced through RSCS functions.

To prevent links from being totally locked out by an exhausted (and stagnant) virtual storage tag queue, a minimum number of tag slots is reserved for each link. This guarantees that a minimum number of files are accepted for each associated link. The number of reserved slots is defined during system generation or in the DEFINE command for each link to be defined in RSCS. The appropriate number of slots to be reserved for each link depends on the expected file traffic, the link's line speed, the time the link is to be active, and the desired level of service to be provided to the link. This number for each link may be arrived at through actual operational experience in each location.

Manage Telecommunication Line Allocation: DMTLAX

DMTLAX is responsible for line port resource allocation to line driver tasks. DMTLAX allocates available switched ports (when a link is activated without a specified line address) through an ALERT request interface. When a line port is specifically requested (by virtual address), DMTLAX checks the device for validity as a line port.

Line Driver Tasks: DMTNPT, DMTSML, DMTVMB, DMTVMC, DMTNJI, DMTPOW

As part of the link activation process, REX (module DMTCRE) loads, for each defined link, a line driver module and starts a line driver task to service a port to an adjacent node in the network.

Note that when RSCS is servicing more than one link with the same line driver specified, there is a separate copy of the same line driver module loaded for the different line driver tasks for each port.

The general functions of line driver tasks are:

- Manage I/O on the BSC telecommunications adapter or CTCA.
- Manage transmission of spool file data via GIVE/TAKE requests to the AXS task.
- Execute or forward command alert elements received from the REX task.

The precise functional requirements for communications management vary from line driver to line driver, depending on the type of remote station the line driver supports.

Each line driver maintains its link status and line activity (TRACE) records in the RSCS system status tables.

Six line drivers are provided. DMTNPT supports remote 2770, 2780, 3770 (in 2770 mode), and 3780 terminals. DMTSML supports an RTAM interface to remote HASP-type and ASP-type systems or workstations. DMTPOW supports communication with VSE/POWER. DMTVMB is for VM/370-to-VM/370 communications on BSC lines. DMTVMC is for VM/370-to-VM/370 communications on channel-to-channel adapters. DMTNJI is for communications from VM/370 to an NJI/NJE subsystem on BSC lines or channel-to-channel adapters.

I/O MANAGEMENT

The two kinds of RSCS I/O operations are: I/O to spool files and I/O to RSCS virtual devices. For I/O to input spool files, the AXS task and line driver tasks use CP DIAGNOSE commands. The procedures for I/O to RSCS system devices, telecommunication adapters, and output spool devices are described in this section.

I/O management for tasks consists of the following functions:

- Queuing requested I/O operations
- Starting I/O operations
- Handling I/O interrupts

- Terminating completed I/O requests.

Handling I/O Requests

When a task requires an I/O operation, it builds an I/O request table in its own storage and passes the table's address to the I/O manager. This table contains the following information:

- A synchronization lock, where I/O completion is to be posted
- The address of the device on which the I/O operation is to take place
- The number of sense bytes to be returned, when applicable
- The address of the channel program to be executed.

See Section 5 for the format of the I/O request table.

The task obtains the I/O request entry address from its SVECTORS table and performs a BALR to it, passing the address of the I/O table in register 1.

The BALR is to subroutine DMTIOMRQ in module DMTIOM in the REX task. Here, the calling task execution is suspended with a FREEZE SVC and the request is queued, as described in Figure 2-5.

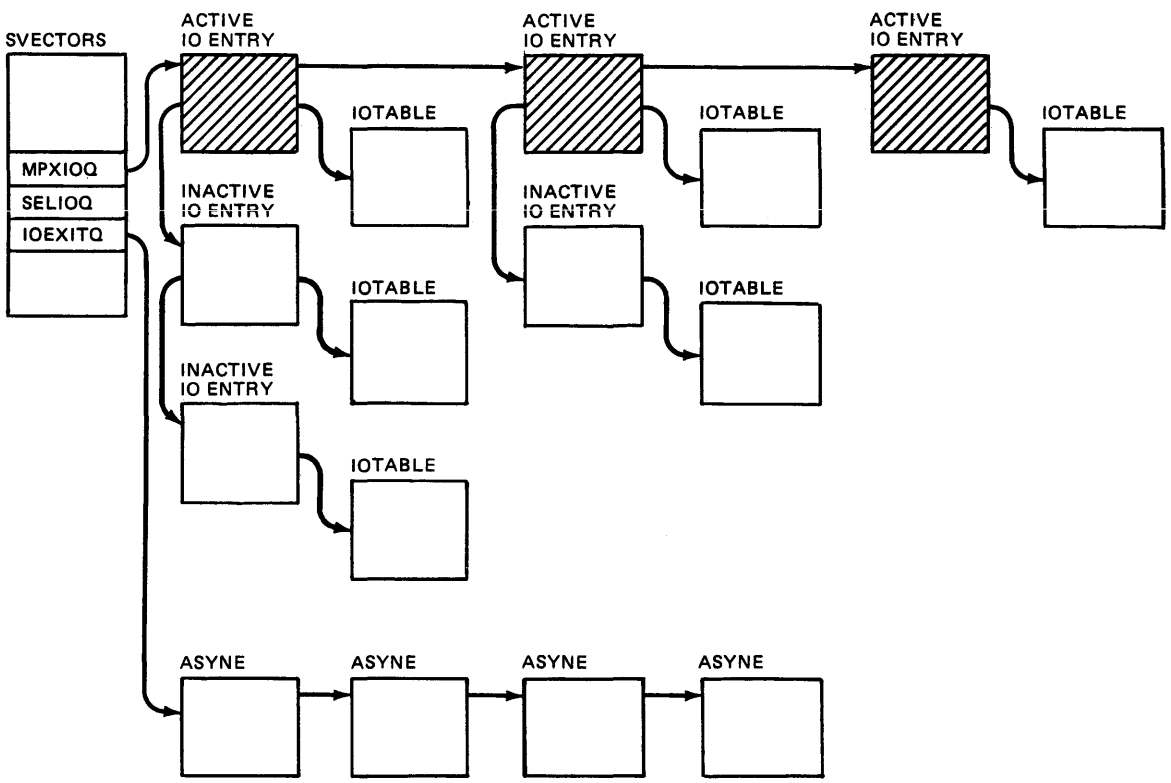


Figure 2-5. I/O Queues and Subqueues

Active and Pending I/O Queues

The supervisor I/O queues (MPXIOQ and SELIOQ) include an active queue and a number of inactive or "pending" subqueues. Each element in the active I/O queue represents an I/O operation which is active on a particular nonshared I/O subchannel. There is only one element in the active I/O queue for a given nonshared I/O subchannel. The active I/O queue is ordered according to ascending numerical I/O subchannel address. Chained to each active I/O queue element there are inactive elements for any operations waiting to be performed on that same nonshared I/O subchannel. The queue element chains are dynamic; the I/O handler adds, inserts and removes elements as required.

Queuing takes place in the subroutine DMTIOMRQ in module DMTIOM in the RSCS supervisor.

QUEUING ACTIVE I/O ELEMENTS When the requested I/O operation is for a presently idle I/O subchannel, an I/O element representing the request is built and chained into the active I/O queue (in its I/O subchannel's numerical address position). The I/O operation is then started by branching to subroutine IOSTART in module DMTIOM.

QUEUING PENDING I/O ELEMENTS When the requested I/O operation is for an I/O subchannel that presently has an I/O element enqueued on the active I/O queue, the nonshared subchannel is busy (active), and the new I/O request cannot be started immediately. In this case, an I/O element representing the request is built and enqueued on the subchannel's inactive I/O subqueue. Then control is passed to the dispatcher module DMTDSP in the supervisor.

Starting an I/O Operation

I/O operations are started by subroutine IOSTART in module DMTIOM in the RSCS supervisor. It attempts to start the I/O for a newly enqueued active queue entry. If it receives a condition code 0, the operation started successfully, and DMTIOM passes control to the dispatcher module DMTDSP in the supervisor. If the I/O cannot be successfully started, it branches to subroutine IODISMIS to terminate the operation with an error.

Handling I/O Interrupts

I/O interrupts are handled by subroutine DMTIOMIN in module DMTIOM in the RSCS supervisor. It locates the active I/O queue element of the device that issued the interrupt, and updates the contents of the element and the status information in the requestor's I/O table. If the interrupting I/O is still running (status incomplete), it passes control to the dispatcher module DMTDSP in the supervisor. If the I/O is complete, it branches to the IODISMIS subroutine to dequeue the interrupting device's I/O request element.

If the DMTIOMIN subroutine finds no active I/O queue element for the interrupting device, it scans the I/O asynchronous exit queue to locate a request for asynchronous I/O exit for the interrupting device. If such a request is found, control is passed to the requestor's exit routine. Otherwise, the interrupt is ignored, and control is passed to the dispatcher.

Dequeuing I/O Requests

The subroutine IODISMIS in module DMTIOM in the RSCS supervisor removes the active I/O request queue element specified by the routine that branches to it. It calls the POST routine to signal I/O completion to the requesting task. It rechains the remaining I/O request queue elements. If there is an inactive I/O queue for the just-detected entry's nonshared I/O subchannel, it makes the first inactive entry active by chaining it into the active queue, and branches to the IOSTART subroutine in DMTIOM.

INTERRUPTION HANDLING

Supervisor service routines handle three kinds of interruptions: external interruptions, SVC interruptions, and I/O interruptions.

In RSCS, supervisor routines use the SVC (SUPERVISOR CALL) to suspend the execution or dispatching of a task when that supervisor routine received control. On an SVC interruption in RSCS, DMTSVC is entered. DMTSVC saves the status of the executing task and passes control to the calling supervisor routine in supervisor execution mode.

RSCS handles external interruptions from tasks by searching for asynchronous exit requests supplied by tasks. When a request with a code matching the external interruption code is found, its asynchronous exit is taken; otherwise, the external interruption is ignored.

I/O interruptions are handled by the RSCS I/O manager. When an active I/O request causes an I/O interruption, the status of the I/O request is updated to reflect the new information. Otherwise, a search is made for an asynchronous exit request for the interrupting device. When one is found, the asynchronous exit is taken. Otherwise, the interruption is ignored.

Special Message Interruption Handling

VMCF (Virtual Machine Communications Facility) external interrupts are used to pass CP Special Messages to RSCS. When one of these interrupts occurs, control is passed to DMTREXCF which disables RSCS for other VMCF interrupts and posts the VMCF synch lock for DMTREX processing. DMTREX moves the pertinent VMCF header information and Special Message text to the RSCS defined buffer (REXREQ). Control is then passed to DMTCMX for Special Message processing which consists of validity checking the Special Message text and performing the indicated RSCS command. DMTCMX then returns control to DMTREX which enables RSCS for VMCF external interrupts.

RSCS SPOOL FILE FORMAT

RSCS uses the CP spool file facilities of VM/370. The spool file records are one page (4096-byte) blocks that contain data records and control records. The data records are punch card or print line images, and appropriate CCWs. The control records are linkage records and tag records, and the special headers and trailers on files from non-RSCS to non-RSCS systems.

CP Spool Data Records

Each spool logical record (card or print line) is stored as one CCW that moves data (READ or WRITE), a TIC to the following CCW, and the full data record. Space is left at the end of each buffer so that a SENSE command can be inserted to force concurrent channel end and device end. For card punch channel programs there is an additional back chain field that points to the card previously punched so that error recovery for punch equipment checks can back up one card. The only exception to the format of READ/WRITE-TIC-Data is in buffers of files directed to the printer. In this case, immediate operation code CCWs (skips and spaces) have their suppress data transfer flag (skip) bit set to one, and are followed by the start of the next record.

CP Spool Buffer Linkage

In addition to the data and CCWs contained in each 4096-byte spool buffer, the first two doublewords contain CP-supported forward and backward links to the next and previous buffers in the file. This two-way linkage allows the file to be backspaced or restarted from any point at any time. Also, it means that if I/O errors are encountered while reading one buffer, the file is put in system hold status. If purged, all buffers except those in error are released. The two-way chain allows this control of the file while preventing fragmentation by allowing pages to be assigned and released individually regardless of their ownership.

CP Spool Tag Record

The first spool buffer of an output spool file contains a special CP-supported data record called the tag record. This record immediately follows the two doublewords containing the forward and backward buffer linkage pointers. The tag record allows VM/370 users to specify information to be associated with spool files that they generate. The information is entered via the CP TAG command, although the tag record is not considered a spool file data record and is not printed or punched as part of the spool file. However, the contents may be interrogated with the CP TAG QUERY operator command.

The format of the tag record is a NOP CCW, followed by a TIC to the next CCW and a 136 byte data field. To differentiate the tag record from an immediate control CCW (no TIC-data sequence) independently of the command code, the skip bit (bit 35) in the CCW has the following convention:

```
Bit 35 = 0  for NOP CCW, TIC, data (tag record)
       = 1  for control CCW (immediate command)
```

RSCS Tag Record Format

The tag record is a user information record, and RSCS as a "user" defines for its own use a special format of data in the tag records.

Whenever a file that is received on a link and spooled is to be

processed (forwarded) by another line driver, RSCS prefaces the tag data with a three character string, C'S&F', called a store and forward flag, that identifies to the RSCS file acceptor function a tag written by RSCS.

A tag with a store and forward flag has more information than the tag written by CP when a virtual machine user spools a file to RSCS. (The CP-written tag contains the operands that the virtual machine user supplies, in RSCS-specified syntax, for each file spooled to RSCS from a local virtual machine).

Non-RSCS Control Records

The basic OS/NJE transmission unit is a job. RSCS handles each job transmitted from an NJE system as an individual file. Within the job are control statements: job header, data set header on each data set in the job, job trailer. To accommodate these control cards as unit record data records within the CP spooling discipline, RSCS converts them to NOP records (command code X'03' in their CCWs) while they are within a VM/370 system, and attaches each job's routing information on a tag. When forwarding them on an RSCS-to-OS link, the line driver removes the tag and reconstructs the NOP control records as OS control statements.

If NJE jobs are punched or printed on real devices within the VM/370 realm, or are read by virtual machine spool readers, the NOP records are discarded and do not appear as garbage data within the real data.

VIRTUAL STORAGE MANAGEMENT

The supervisor virtual storage service routine DMTSTO handles requests by tasks for main storage. When a task requests main storage, DMTSTO reserves page(s) of storage for it. Main storage is freed directly by task programs.

DMTQRQ manages requests for free elements of the supervisor status queue. Supervisor routines call DMTQRQ to reserve and release supervisor status queue elements.

RSCS Basic Functions

This section describes the major functions of RSCS. It is to help you locate the module that contains the code for an operation within an RSCS function. The function descriptions describe the sequence of operations and the modules and tasks they occur in. The descriptions assume an understanding of supervisor services such as task dispatching, task-to-task communication, task synchronization, and I/O management.

RSCS CONFIGURATION AND STARTUP

Loader

The RSCS dynamic loader (DMTCRE) loads text files directly from CMS

disks, using the reentrant CMS read access method in REX module DMTCOM. The loader restricts the program to be loaded to a single CSECT and does not resolve external references. Multiple module RSCS tasks are merged into a single CSECT text file prior to loading, using the RSCS preloader under CMS.

RSCS Initialization

The three initialization modules are DMTINI, DMTMIN, and DMTIRX. DMTINI performs IPL disk load write and read operations. DMTMIN is an MSUP module that performs all initialization functions independent of particular task level programming. DMTIRX is a REX task module that performs all initialization related to specific RSCS functions.

Initial loading of RSCS is performed by DMKLD00E LOADER from the virtual card reader. When initial loading is complete, control passes to DMTINI, which functions as a stand-alone routine. The operator is asked for instructions, and RSCS is normally written in IPLable format to the system residence disk. On subsequent IPL from disk, DMTINI reads the disk and reloads RSCS into storage. When DMTINI has completed its function, control passes to DMTMIN.

DMTMIN begins by initializing its own base register and the fixed address fields in low storage. It determines the size of virtual machine storage by referencing each storage key until an addressing exception occurs, and then DMTMIN copies itself to the last page in storage. From its new location at the end of storage, DMTMIN builds the MSUP storage allocation map, MAINMAP, according to storage size, starting at the beginning of the originally loaded DMTMIN module. Immediately following the end of the storage map, the supervisor queue is built and chained into a single free element queue. The minimum number of supervisor queue elements is computed as two for each page of main storage, and is extended through to the end of the last storage page required for the minimum count. All of the storage in use by MSUP at this point is reserved by placing X'FF' in the storage map entry for each page. Finally, the initial task (REX) is created by dequeuing a free supervisor queue element, building the initial task element in it, enqueueing the new task element as the task queue, and reserving a single page of storage at X'10000' for the initial task. Task processing is begun when DMTMIN completes by passing control to the MSUP dispatcher. The storage used by DMTMIN remains unreserved, and is normally used to load the first task dynamically started by DMTIRX (AXS).

DMTIRX is entered immediately the first time REX is dispatched by MSUP as the initial task. Initially, DMTIRX copies itself to the highest possible page boundary address (e.g., X'E000') before the start of REX at X'10000', sets its base registers to the new address, and branches to label IRXGO in the new copy. The task name 'REX' is placed in the initial task element, and RSCS-defined task vectors are set in the MSUP low storage TVECTORS fields. The virtual machine user ID being used is retrieved from CP by a hypervisor call (DIAGNOSE X'00') and is set in its field in DMTCOM. Next, the operator console and system residence DASD device are identified by a similar hypervisor call (DIAGNOSE X'24'), and I/O parameters for these devices are initialized. Next, the CMS Access Control Areas in DMTREX and DMTCRE are initialized as follows:

1. If the RSCS system disk is standard CMS 800-byte format, no special initialization is required and the routine is exited.
2. When the RSCS system disk is in CMS Enhanced Disk format, the GETSUPAG routine in DMTCOM is called to obtain virtual storage for

later use as data buffers. The addresses of the obtained buffers are set in the CMS Access Control areas and the routine exited.

The RSCS location definition, link tables, default start parameters, route table, port table, tag slots, and installation variables are initialized from the RSCS directory by a call to GENVNET. The installation specification tables and the tag slot queue are built beginning at the original starting address of DMTIRX, such that DMTIRX and DMTINI (which are no longer needed) are overwritten.

On return from GENVNET, the total storage from the beginning of the REX task to the end of the installation variable tables is computed and reserved for the REX task in the MSUP storage map. Next, the operator console I/O table in DMTREX is initialized, the console and timer asynchronous interrupt exits are initialized by a call to the ASYNREQ (DMTASY) entry to MSUP. At this point, RSCS prepares for communicating with the CP Special Message Facility by issuing a VMCF AUTHORIZE command and setting bit 31 on in Control Register 0. The AXS and LAX tasks are loaded by calls to DMTCRE, virtual storage is obtained for the DIAG 8 response buffer via a call to GETSUPAG, the initial active link count is set to zero, the maximum number of startable links is computed and set based on the virtual storage available, the program check new PSW is set to enter the program check manager in REX (DMTREXPI), the initial messages are generated and issued by a call to DMTMGX, and control is passed to DMTREXIN. At this point, REX executes the PROFILE RSCS initial command sequence if present and not suppressed by the IPL parameter keyword 'NOPROF'. Finally, AXS is notified via an alert call from REX that initialization is complete, queued spool input files are accepted by AXS, and normal RSCS processing begins.

RSCS SYSTEM DISK ACCESS

The RSCS system disk is a CMS mini disk created, updated, and maintained by installation personnel working under CMS. The RSCS access method supports the RSCS system disk as shown in Figure 2-6.

	Current CMS Disk Format	Enhanced Disk Format*		
Device Type	Block Size: 800	1K	2K	4K
2314	Yes	Yes	Yes	Yes
3330	Yes	Yes	Yes	Yes
3340	Yes	Yes	Yes	Yes
3350	Yes	Yes	Yes	Yes
3310	No	Yes	Yes	Yes
3370	No	Yes	Yes	Yes

* This support may be obtained in VM/370 Basic Systems Extensions Program Product, Release 2; it does not support 800-byte blocks. All of the support for Enhanced Disk Format is new function for RSCS Networking.

Figure 2-6. RSCS System Disk Format

In the algorithms used to access DASDs in blocksizes other than 800 bytes, the RSCS access method permits one level of indirect addressing (one level of pointer blocks). This enables the RSCS user who creates the RSCS system disk under CMS of the VM/370 Basic Systems Extensions Program Product, Release 2, to have the characteristics shown in Figure 2-7.

Block Size	Maximum Number of Files	Maximum Number of Data Blocks	Maximum Size of File
1K Bytes	4,096	256	256K Bytes
2K Bytes	16,384	512	1024K Bytes
4K Bytes	65,536	1024	4096K Bytes

Figure 2-7. RSCS System Disk Characteristics

The operation of this support within RSCS is transparent to the RSCS user. This support is neither invoked nor used by the RSCS user. An installation's options for formatting the RSCS system disk under CMS may be determined from the preceding table.

To locate a file, given the file name and file type, this support usees the addressing scheme shown in Figure 2-8.

RSCS FILE HANDLING FUNCTIONS

Introduction

RSCS file handling is described in the following overview section, followed by descriptions of the link table, link activation, and the input and output of files by the line drivers. Note that there is a description of the MULTI-LEAVING protocol used by several line drivers, in Appendix A of this publication.

Scenario of RSCS File Handling

This section is a high level description of the file handling that RSCS provides. The description uses Figure 2-9 to show RSCS systems in three VM/370 nodes processing a routed file from an originating node, through an intermediate node, to a destination node.

To fulfill its networking file handling capabilities, RSCS has functions to:

1. Receive files spooled to it from virtual machine users on the local VM system where it resides. (See Figure 2-10, path marked with circled 1.)

2. Receive files from adjacent nodes. (See Figure 2-10, paths marked with circled 2.)

3. Analyze and send received files to local virtual users. (See Figure 2-10, path marked with circled 3.)

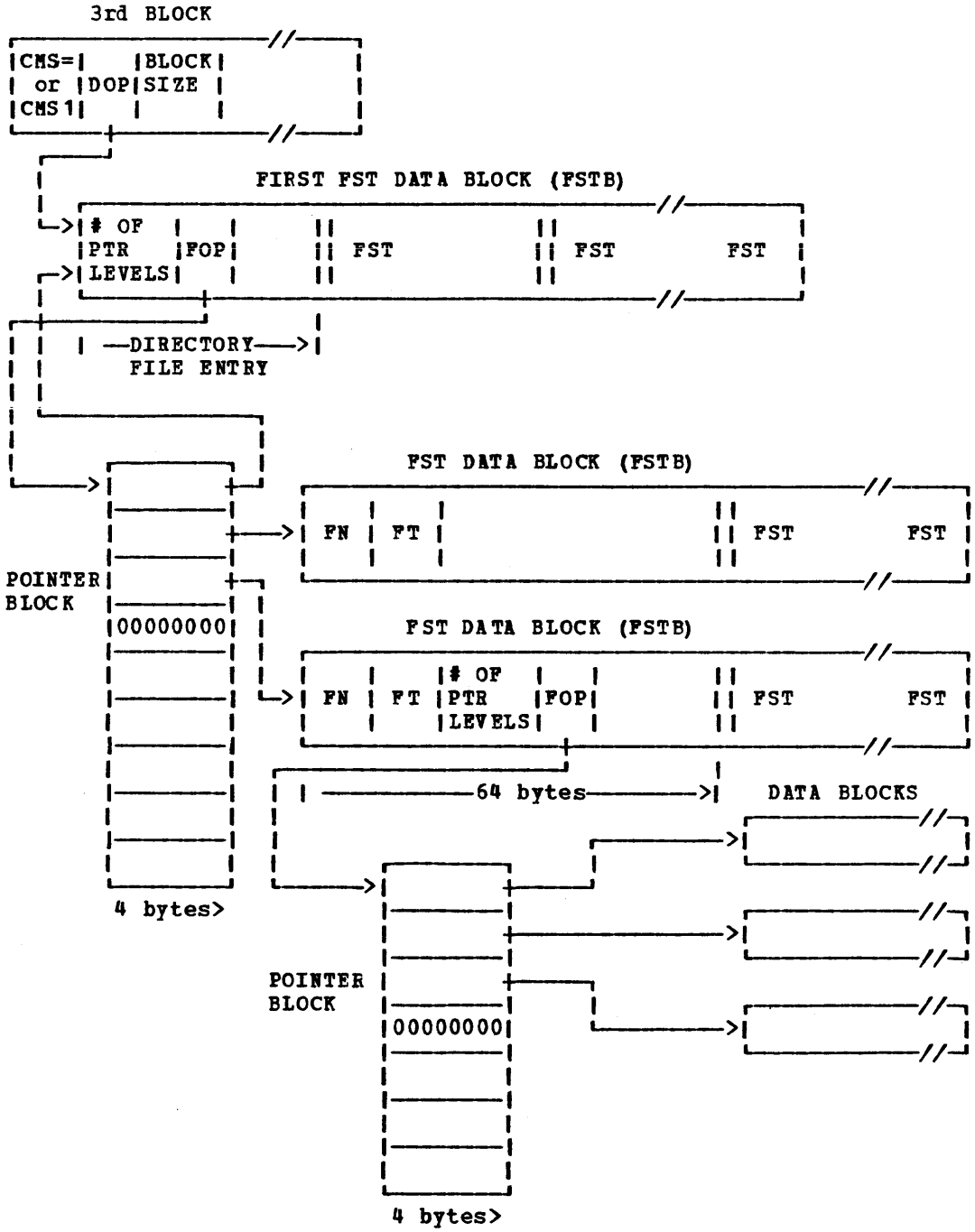


Figure 2-8. Locating a File on the RSCS System Disk (Part 1 of 2)

The procedure for locating a file is as follows:

1. A check is made to determine whether the disk is in EDF format or the current format. If the volume label identifier field is "CMS=", it is in the current format; if the identifier field is "CMS1", it is in the Enhanced Disk Format.

2. If it is in the current format, processing continues as in the VNET PRPQ.

3. If it is in the Enhanced Disk Format, the first File Status Table Block (FSTB) is referenced by the Directory Origin Pointer (DOP).

4. From the first FST in the FSTB, the number of pointer levels is obtained. If the number of pointer levels is 0, the current and only FSTB is searched for the FST containing the given filename and filetype.

5. If the number of pointer levels is 1, the File Origin Pointer (FOP) points to the pointer block. Each entry in the pointer block points to an FSTB. The end of a partially filled pointer block is denoted by zeros.

6. Using the pointer block entries, FSTBs are accessed and scanned until the required FST containing the given filename and filetype is found.

7. If the number of pointer levels is 1, the FOP points to a pointer block. Each entry in the pointer block points to a data block. The end of a partially filled pointer block is denoted by zeros.

8. If the number of pointer levels is 0, the FOP points to the one and only data block for this file.

9. Using the pointer block entries, the data blocks are accessed until the file has been read.

Figure 2-8. Locating a File on the RSCS System Disk (Part 2 of 2)

4. Analyze and forward files to adjacent nodes. (See Figure 2-10, paths marked with circled 4.)

In this scenario, the network is very simple: three VM/370 systems (VM1, VM2, and VM3) each with an RSCS system running, and each with telecommunication links. Systems VM1 and VM3 each have a user virtual system, CMS, running. (This is an example only; CMS does not have to be the user machine.) The scenario has six steps, A through F, described below.

(A) User Archie on the CMS virtual system at location VM1 is sending a file to user Bob on the CMS virtual system at location VM3. To do so, Archie issues the CP commands:

```
SP OOD TO RSCS1
TAG DEV OOD VM3 BOB
PUNCH filename filetype
```

These commands tell CP to spool the virtual punch output from Archie's job, place the specified tag data on the spool file, and give the spool file to the local virtual system called RSCS1. This is an instance of the first type of basic RSCS file handling

function described below.

(B) After CP does this, RSCS1 looks at the tag information on the spool file. RSCS1 determines that the file is destined for VM3, and, to determine this location's specifications for forwarding files to VM3, consults its routing tables and link tables. It transmits the file on the link to VM2. This is an instance of the fourth type of basic RSCS file handling function described below.

(C) On the VM/370 system VM2, the RSCS2 virtual machine receives the file and spools it. RSCS2 sees that the destination information in the file's tag data does not contain this location's (VM2's) locid, and tells VM2 CP to give it to RSCS2. (It spools the file to itself in order to transfer the handling of the file from the RSCS receiving function to the RSCS sending function.) Then it tells RSCS1 that the file is successfully received, allowing RSCS1 to release and effectively erase its copy of the file. This is an instance of the second type of basic RSCS file handling function.

(D) CP informs RSCS2 of the new spool file's presence. RSCS2 performs the same forwarding function (type 4) described for RSCS1's forwarding of the file. In this instance, RSCS2's link table directs it to send the file on the VM3 link.

(E) At system VM3, the RSCS3 virtual machine receives the file and spools it. RSCS3 sees that the file's tag data specifies user Bob, on the CMS system, on the VM3 location (locid) as the destination. RSCS3 tells VM3 CP to spool the file to virtual machine user Bob. RSCS3 issues a console message to Bob, informing him of the file. RSCS3 tells RSCS2 that the file is successfully received, terminating this transmission and allowing RSCS2 to release the VM2 spool copy of the file.

(F) CP passes the RSCS message and its own notification of the file's arrival to Bob.

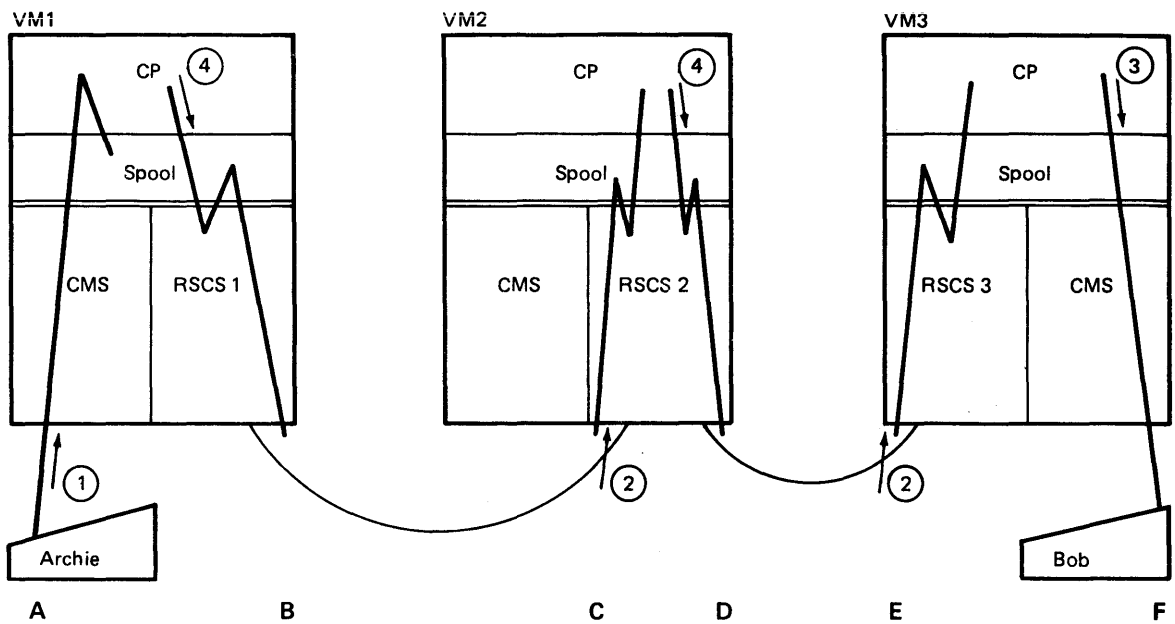


Figure 2-9. Scenario of RSCS File Handling Functions

Receiving a File from a Local Virtual System

This is handled by CP in the same way that it does for any other virtual machine under its control. Spool reader files are queued for input to virtual machines, and are retained by CP until they are read or purged. Files are normally read by virtual machine simulated card readers, but a number of direct hypervisor calls (simulated DIAGNOSE instructions) are provided by CP to allow virtual machine systems to interrogate and manipulate and read the input spool file queue.

Receiving a File from a Remote System

1. The line driver recognizes the start of an arriving file and issues a message to notify the RSCS operator.

2. RSCS spools the file to itself (spool file opened for output).

   a. The line driver issues a GIVE request to DMTAXS requesting for a spool file to be opened for output (request code=11). The "request arrival" synch lock of DMTAXS gets posted.

   b. DMTAXS gets dispatched at AXSCYCLE, tests the "request arrival" synch lock, finds that is on, and then branches to AXSACCPT.

   c. DMTAKE is called to take the GIVE request.

   d. REQXEQ is called to decipher the request code (11). It determines the address of the appropriate subroutine (OPENOUT), and branches to it.

   e. OPENOUT scans the link's queue of active output tag slots to determine if the file is already being processed. If so, the I/O area (TAKE response buffer) is returned to the line driver and the appropriate open code ("old file found") is posted before control is returned to AXSCYCLE.

   f. If not so, OPENOUT: (1) calls GETLINK to check if the line driver has a locally defined link; (2) calls GETSLOT to get a free tag slot, and (3) calls GPAGEREQ to get a page for a new I/O area.

   g. The tag information supplied in the GIVE request is moved into the new tag slot.

   h. DEFINE is called to internally define a virtual punch by means of a call to CP.

   i. OPENOUT moves an asterisk to the virtual machine destination ID field (TAGTOVM field of the tag element) to indicate spooling the punch to RSCS.

   j. OPENOUT calls VSPOOLP to effect a CP SPOOL punch command.

   k. OPENOUT initializes the I/O area (pointed to by register 7) by building the device I/O table. The tag element is then placed at the beginning of the link's active output tag queue.

   l. In response to the line driver's GIVE request, OPENOUT returns the address of a new I/O area and inserts the appropriate open post code before returning control to AXSCYCLE.

3. The line driver gets dispatched and issues subsequent punch commands as the file is received.

4. When EOF is reached, the line driver closes the virtual punch.

   a. The line driver issues a GIVE request to DMTAXS, specifying a closing of the output file (request code = '12').

   b. DMTAXS gets dispatched at AXSCYCLE which eventually calls CLOSEOUT.

   c. CLOSEOUT locates the active output tag slot and dequeues it from the active output queue.

   d. CLOSEOUT updates the tag element with information obtained from the line driver's tag prototype.

   e. CLOSEOUT determines if the file is destined for this location or to a remote location. This is done by comparing the TAGTOLOC field with the LINKID field of the local location's link table.

   f. If the fields are not equal, the file is to be stored and forwarded. VTAGD is called to assemble the appropriate tag command text and to issue the CP TAG command. VTAGD inserts the "S&F" flag into the command text; the command text contains the following:

      S&F TAGTOLOC TAGTOVM TAGPRIOR TAGINLOC TAGINVM TAGINTOD TAGORGID TAGCNTL

   g. VSPOOLP is called to issue CP SPOOL punch OFF to reset the punch to no-spool mode.

   h. VCLOSEO is called to issue CP CLOSE punch.

   i. DETACH is called to issue CP DETACH punch.

   j. FPAGEREQ is called to free the file I/O area's virtual storage page.

   k. FREESLOT is called to free the tag.

   l. The close post code is set and control is returned to AXSCYCLE.

Sending a Received File to a Local Virtual System

Upon receiving a file from a remote node, the line driver enters the file into its local CP spool system, as described previously. However, when the line driver closes the spool file, AXS determines that either the file is to be forwarded to a remote location (in which case the file remains spooled to RSCS), or the file belongs to a local virtual machine (in which case the file is spooled to the local virtual machine's reader). Therefore, the sequence of events is like Steps 1 through 4 of the preceding paragraph with a difference, starting at Step 4-f, as follows:

1. The line driver recognizes the start of an arriving file and issues a message to notify the RSCS console operator.

2. RSCS spools the file to itself (spool file opened for output).

3. The line driver gets dispatched and issues subsequent CP spool punch commands.

4. When EOF is reached, the line driver closes the virtual punch.

   a. The line driver issues a GIVE request to DMTAXS, specifying a closing of the output file (request code = '12').

   b. DMTAXS gets dispatched at AXSCYCLE which eventually calls CLOSEOUT.

   c. CLOSEOUT locates the active output tag element and dequeues it from the active output queue.

   d. CLOSEOUT updates the tag element with information obtained from the line driver's tag prototype.

   e. CLOSEOUT determines if the file is destined for this location or to a remote location. This is done by comparing the TAGTOLOC field with the LINKID field of the local location's link table.

   f. If the file is destined for this node (local locid same as TAGTOLOC field), a further check is made as to whether the file is to go to a local virtual machine or to a remote workstation.

      With the TAGTOVM field as a search argument, GLINKREQ is called to see if a link with a link ID that matches TAGTOVM is defined at the location. If so, the file is spooled to RSCS and enqueued for transmission to the workstation. Otherwise, the file is spooled to the specified local virtual machine.

5. The file is punched to the local virtual machine user.

   a. CLOSEOUT calls VTAGMSG to put user notification message into the spooled file's tag area.

   b. VSPOOLP is called to point the reader of the virtual machine (whose ID is stored in TAGTOVM) to the spooled file.

   c. VCLOSEO is called to issue CP CLOSE punch.

   d. DETACH is called to issue CP DETACH punch.

   e. FPAGEREQ is called to free the file I/O area's virtual storage page.

   f. FREESLOT is called to free the tag.

   g. The close post code is set and control is returned to AXSCYCLE.

Sending a Received File to a Remote Node

1. CP notifies RSCS of the spool file on RSCS's virtual reader.

   a. A simulated asynchronous device end I/O interrupt is given to the lowest virtual device address reader that is both idle and eligible to read the file, RSCS's device address X'001'.

   b. The I/O interrupt handler (DMTIOM/IOASYNCH) selects the

address of the asynchronous exit routine for RSCS and passes control to it (DMTAXM/AXSASYIO).

c. AXSASYIO calls DMTPST to post the "file arrival" synch lock, and then returns control to the dispatcher.

d. When DMTAXS gets dispatched, it executes at AXSCYCLE, which detects that the "file arrival" synch lock has been posted, and causes a branch to ACCEPT.

2. RSCS accepts the file.

a. DMTAXM/ACCEPT internally issues a "CLOSE RDR HOLD" to CP.

b. ACCEPT calls GSUCCESS to read a spool file block and tag information. (GSUCCESS uses the VM/370 DIAGNOSE X'14' instruction, subcode X'FF', to do this). The address of the spool file block and tag information is returned to the buffer specified in the DIAGNOSE X'14' instruction.)

c. ACCEPT calls TAGFIND and scans the tag queue of every link defined in the link table to look for a match with the tag just read.

If there is a match it indicates that the spool file block and tag just read was already in process. TAGFIND returns to ACCEPT with condition code 0, and ACCEPT reads the next tag and spool file block.

d. If there is no match, the spool file is eligible for processing; TAGFIND returns to ACCEPT with condition code 3. ACCEPT then checks the validity of the data in the tag read.

e. ACCEPT extracts the destination information from the tag and calls the module GROUTREQ to determine the link on which to enqueue the file.

If GROUTREQ fails to find a link that has been defined for the destination, ACCEPT checks if the file has been marked "S&F" (store-and-forward) and if it originated from RSCS.

If the file has been flagged S&F (previously done when the file was closed after RSCS received it and spooled it) and if there is no link defined for it: (1) the spool file is purged from VM/370, (2) AXSM103, which issues a message request to DMTREX, is called to notify the spool file's originator, and (3) the next spool file is read from RSCS's reader.

If the file was not flagged "S&F" (that is, the file originated locally), ACCEPURG examines the location ID of the file's originator. If the originating location ID (TAGTOLOC) is the same as the local LOCID (described by the LINKID field of the first link table entry in the link table chain), the file must have originated from VM/370's real card reader if the file origin userid is equal to the userid of the RSCS virtual machine. In this case the file is purged and a message issued to the originator.

If the origin location ID is not the local LOCID, the file is transferred to the originator by internally issuing the CP TRANSFER command, and a message is sent to the originator.

f. If a link is found, GETSLOT is called to get a free tag slot.

If no free tag slot is available: (1) the file is left alone

in the spool, (2) the count of the number of pending files is incremented, (3) AXSM102 is called to issue the "file pending" message, and (4) the next spool file block and tag is read.

g. TAGGEN is called to move the contents of the tag and spool file block into the free tag slot to generate a new tag element.

h. TAGPLACE is called to enqueue the tag slot, by transmission priority and file size, to the tag queue of the link selected by GROUTREQ.

i. The LFLAG bits in the link's link table are checked to see if the link is already processing a file. If so, ACCEPT proceeds to read the next spool file. If not so, the ALERT bit is reset and an ALERT is issued to the link's line driver and control is returned to the dispatcher.

3. The line driver reads the spool file for transmission to the next node. (Spool file is opened.)

a. The line driver issues a GIVE request to DMTAXS, requesting to open a file for input (request code = X'01'). The "request arrival" synch lock for DMTAXS gets posted.

b. When DMTAXS gets dispatched, AXSCYCLE finds that its "request arrival" synch lock has been posted and thus branches to AXSACCPT, which calls DMTAKE to take the GIVE request. REQXEQ is then called. REQXEQ deciphers the request code (X'01'), determines the appropriate processing routine (OPENIN), and branches to it.

c. OPENIN searches for active files for the link specified in the request by scanning its active input tag queue. If such a file is already in process, that file is returned to the caller.

d. OPENILNK finds the link table for the link that sent the open request. This is done by scanning for a link table that contains a link ID that matches the requesting link's ID.

e. Call UNPEND to enqueue as many pending files for the line driver as possible: (1) Call TAGFIND to search for a pending file for this link, (2) call TAGGEN to generate a tag slot, (3) call TAGPLACE to enqueue the new tag slot on the link's queue, (4) update the pending file count, and (5) issue a message if a pending file is missing.

f. Call FILSELEC to select a file to be read from the link's tag queue. FILSELEC scans the tag queue, pointed to by LINPUTQ in the line driver's link table. It dequeues the first tag found for a file that is not in hold status and that matches the file class setting in the caller's link table. A message is issued if the file is missing from CP's spool system.

g. Call GPAGEREQ to get a page of storage for the I/O area. The address of this storage is stored in the TAGBLOCK field of the selected tag slot.

h. Call DEFINE to internally define a virtual reader by means of a call to CP.

i. Call VSPOOLR to internally issue the "SPOOL READER CLASS *" command.

j. OPENIN issues the DIAGNOSE X'14' command to make the file described in the tag slot the next in the reader just defined. Then it opens the file by issuing another DIAGNOSE X'14' command to read the first spool page buffer of the file into the I/O area.

k. OPENIN enqueues the tag slot at the beginning of the active input tag queue.

l. As a response to the original open request by the line driver to DMTAXS, OPENIN returns to the line driver the address of the I/O area (buffer) containing the spool file records. The line driver reads the rest of the file with successive DIAGNOSE X'14' calls.

m. A branch is made to OPENEXIT which inserts the appropriate open post code before giving control back to AXSCYCLE. The return from TAKE posts the GIVE table in the line driver storage.

4. The line driver is dispatched and starts transmitting spool file records to the remote location. EOF is recognized when a DIAGNOSE X'14' returns EOF on the input spool file.

5. The line driver receives positive acknowledgement from the receiving location after the file is completely transmitted. The file is purged or held depending upon the close options. (Spool file is closed.)

a. The line driver issues a GIVE request to DMTAXS, requesting input file closing (request code = X'02'). The "request arrival" synch lock for DMTAXS gets posted.

b. DMTAXS gets dispatched at AXSCYCLE and finds that its "request arrival" synch lock has been posted, and branches to AXSACCPT.

c. DMTAKE is called to take the GIVE request.

d. REQXEQ is called to decipher the request code (X'02'), determines the name of the appropriate subroutine (CLOSIN), and branches to it.

e. CLOSIN scans the active input tag queue for the tag of the file to be closed. When the tag is found, it is removed from the active input tag queue.

f. CLOSIN examines the AXSREQ field of the AXS monitor control area and decides whether the file is to be held or purged from the spool system. (AXSREQ contains a copy of the line driver's request element.)

g. CLOSIN frees the I/O areas used, sets the appropriate close post code, and returns control to AXSCYCLE.

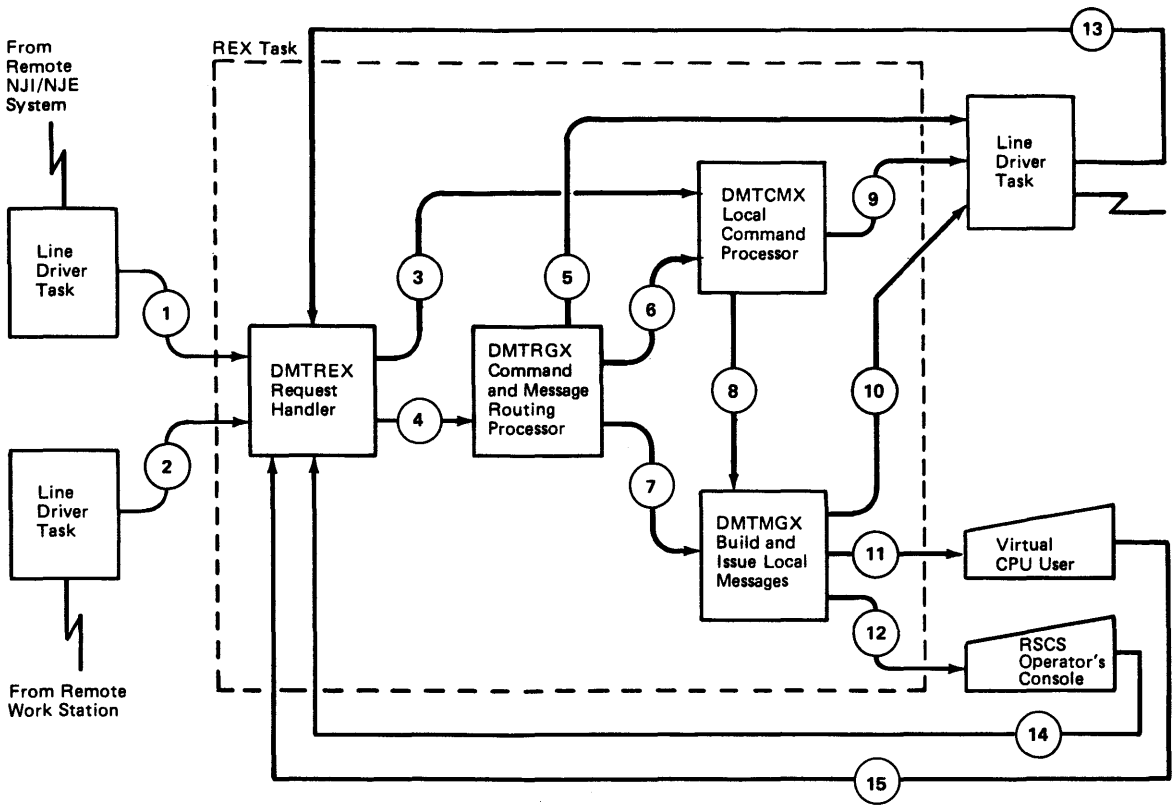
COMMAND AND MESSAGE HANDLING FUNCTIONS

Command and message flow within RSCS is handled by a store and forward mechanism with differences from file store and forwarding. Some differences are:

- Commands and messages are not stored on spool files, and are lost if the RSCS node handling them goes down.

- Routed commands and messages are not forwarded if the necessary link is undefined, inactive, or not connected.

Figure 2-10 shows the steps used in handling RSCS commands and messages. The circled numbers in the diagram correspond to the following descriptions.



Summary of Communication Used on Each Numbered Path:

1. Routing Request Element (GIVE/TAKE)
2. Command Request Element (GIVE/TAKE)
3. Command Request Element (BALR)
4. Routing Request Element (BALR)
5. CMD/MSG Alert Element (ALERT)
6. Command Request Element (BALR)
7. Message Request Element (BALR) - remote origin
8. Message Request Element (BALR) - local origin
9. Command Alert Element (ALERT)
10. Message Alert Element (ALERT)
11. Message to Local User (CP MESSAGE command)
12. Message to Local RSCS Operator (SIO)
13. Message Request Element (GIVE/TAKE)
14. RSCS Local Operator Command (SIO)
15. Message from Local User (CP SMSG command)

Figure 2-10. RSCS Command and Message Handling

Remote System Command and Message Input to RSCS (Path 1)

When a line driver receives a routed command or message from a remote system, it builds a Routing Request Element and passes it in a GIVE request to supervisor module DMTGIV, which builds a Give Queue Element, posts the DMTREX synch lock (at the end of the task save area), and marks the REX task dispatchable.

When REX is dispatched and finds its GIVE/TAKE synch lock posted, it issues a TAKE, examines the request, and because it is a Routing Request Element, passes it to the routing processor module DMTRGX.

Remote Workstation Command Input to RSCS (Path 2)

When a command is from a remote workstation (SMI or NPT), the line driver builds a Command Request Element and passes it in a give request to supervisor module DMTGIV, which builds a Give Queue Element for it, posts the DMTREX Give/Take synch lock (at the end of its task save area), and marks the REX task dispatchable.

When REX is dispatched and finds its Give/Take synch lock posted, it issues a Take request to supervisor module DMTAKE. The TAKE processor locates the GIVE Queue Element for this task, and passes the Command Request Element to the taking task, REX. The task examines the request and because it is a Command Request Element passes it to the command executor module DMTCMX.

Commands for Local Execution (Path 3)

DMTREX passes command request elements for local execution to DMTCMX. These elements may originate from a remote workstation line driver, a command sent to RSCS through the Special Message Facility, or built by DMTREX from a command entered at the RSCS operator console.

Routing Request Element for This Location (Path 4)

DMTREX passes routing request elements from remote NJI/NJE systems to DMTRGX.

If the destination locid in the Routing Request Element contains the local location's locid and it is a command request, DMTRGX forwards it to command executor processor module DMTCMX. If it is a message for the local location's locid and no destination userid is specified, DMTRGX constructs a DMTRGX170 console message and forwards it to the message processor module, DMTMGX. But, if it is a message for this locid and there is a userid specified in the routing information, the following happens:

a. If there is a linkid defined matching the specified userid (remote workstation), RGX moves the destination userid into the destination locid field and constructs a DMTRGX170 message element, which it passes to DMTMGX to issue.

b. If there is no linkid matching the specified userid, the userid must be for a local virtual machine or the RSCS

operator. DMTRGX formats a DMTRGX171 message element and forwards it to DMTMGX to issue to the local virtual machine specified in the destination userid.

Routing Request Element for Another System (Path 5)

If the destination locid in the Routing Request Element is not the local location's locid, DMTRGX calls GROUTREQ in DMTCOM to determine the path to the destination locid. If the link is defined, active, and connected, RGX passes the element to it with an alert request to the link's line driver task. If the link is undefined, inactive, or not connected, RGX either (a) for a message type Routing Request Element, discards it, or (b) for a command type Routing Request Element, issues a message to the originator via DMTMGX, giving the reason for inability to process the command any further.

Local Commands Originating from NJI/NJE Systems (Path 6)

DMTREX passes command request elements for local execution to DMTCMX. These elements originate from a remote NJI/NJE system.

Message Request Elements from NJI/NJE Messages (Path 7)

DMTRGX passes to DMTMGX message request elements that are constructed from message type routing request elements received from remote NJI/NJE systems, as described for Path 4.

Messages Arising from Command Execution (Path 8)

The message routine builds messages, using text supplied in module DMTMSG and variables supplied in the message buffer, and forwards them to the appropriate recipients. Messages issued as part of locally-executed commands go to DMTMGX.

The contents of each message built at this location are edited into one of four formats. Fditing is under control of the EMSG setting in CP. The DIAGNOSE command X'5C' invokes this editing. Depending upon the EMSG setting, the message is one of:

Message header and message text
Message header only
Message text only
Neither header nor text

Line Driver Handling of Command Alert Elements (Path 9)

Line drivers receive alert elements, containing line driver Command Alert Elements. The two general classes of line driver alert elements are: those that specify internal line driver commands that one of the line driver's processors executes, and those that specify commands or

messages for the line driver to transmit on its link.

The second byte of a line driver command alert element, called the Function Code, identifies the element type. All X'Bx' requests are forwarding (transmitting) commands containing data for remote locations:

X'B0' is from a locally-entered or remote workstation-entered CMD command;

X'B1' is from a locally or workstation-entered MSG command;

X'B2' is an RSCS-issued routed message built by this location's DMTMGX module and directed to a remote location.

When DMTCMX issues an alert (Path 9 in Figure 2-7) for an internally-executed line driver function (such as backspace), the function code in the line driver command alert element is one of: X'80' through X'84', X'90', X'91', or X'A0'.

When a line driver is able, it accepts any alert immediately. It posts its appropriate synch lock, and stacks all X'B1' and X'B2' requests by calling the REX task, module DMTCOM, routine PMSGREQ. All line drivers except DMTSML and DMTPOW do the same with X'B0' requests.

When a line driver is busy (already processing a command), it rejects any internal line driver commands. DMTSML and DMTPOW also reject X'B0' requests when they are busy.

The line driver mainline program sees the synch lock that indicates one or more stacked requests, and handles the requests by calling the REX task, module DMTCOM, routine GMSGREQ.

Forwarding Locally-Generated Messages on Links (Path 10)

When a locally-originated message, formatted and edited by DMTMGX, is to be forwarded to a remote location, DMTMGX passes it to the appropriate line driver in a Message Alert Element.

Issuing Messages to Local Virtual System Users (Path 11)

DMTMGX issues messages to local virtual system users by means of the DIAGNOSE X'08' instruction, which executes a CP MSG command. If the user that the message is directed to is not logged on, the message is purged.

Issuing Messages to the Local RSCS Operator's Console (Path 12)

DMTMGX issues messages to the local RSCS operator by a BALR to the PMSGREQ routine in module DMTCOM, where the message is queued for issuing by the DMTREX module's console message routine.

Line-Driver-Issued Messages (Path 13)

When the line driver issues a message as part of executing a request, it does so by a GIVE request to DMTREX, passing a Message Request Element.

Local RSCS Operator Command Input (Path 14)

A command issued at the RSCS operator's console may be a request for a function on the local system or at a remote system, or it may specify a message to a local virtual machine user or to a remote operator console or virtual machine user.

The asynchronous interrupt generated by the RSCS console ATTENTION key causes control to pass to the console I/O asynchronous exit routine, DMTREXCI, in the REX task, module DMTREX. The message or command issued by the operator is passed to DMTCMX.

Receiving Messages from Local Virtual System Users (Path 15)

A command issued by any virtual machine to an RSCS virtual machine must be sent through the Special Message Facility. It may be a request for a function at a local or remote RSCS system, or it may specify a message to a remote virtual machine console.

The asynchronous interrupt generated by the Virtual Machine Communication Facility (VMCF) causes control to be passed to the asynchronous VMCF external interrupt routine, DMTREXCF, in the REX task module DMTREX. The message (MSG) or command (CMD) issued by the Special Message sender is then passed to DMTCMX for processing.

RSCS ACCOUNTING

An accounting record is generated by module DMTAXA, when called from CLOSEIN or CLOSEOUT in module DMTAXM, each time an input or output spool file is closed. Accounting records are generated for input files only if the spool file originated locally. Accounting data is obtained from the spool file's tag block. Time and date information is obtained from VM/370 CP through the DIAGNOSE instruction. The DIAGNOSE X'4C' instruction is used to punch the assembled accounting record to VM/370.

Line Driver Functions

THE LINK TABLE

A line driver task can only exist for an entry in the link table. During RSCS initialization, a link table entry is built for each link defined in a LINK statement in the RSCS directory. (The first entry in the link table is reserved as a container of the local locid.) The parameters in the LINK statement are recorded in its link table entry.

During RSCS operation, the RSCS operator can make the following changes to the contents of the link table in RSCS storage:

- Use the DEFINE command to alter any of the parameters in a link table entry that is not active (does not have a line driver task running for it).
- Use the DEFINE command to add new link definitions. (There are sixteen empty link table entries after RSCS initialization.)
- Use the DELETE command to remove a link definition, leaving its previous link table entry empty.

The link table has two functions: it contains the parameters of defined links, as described above, and when the link has been started, it contains running data for its line driver task. See Section 5 of this manual for details of the Link Table Entry contents.

LINK ACTIVATION: LOADING AND STARTING A LINE DRIVER TASK

The objective of line driver task startup is to provide a task to perform the operations of one of the links specified in the link table. This is done by obtaining the storage required by the line driver module, loading the line driver module into that storage, and causing its initialization code to execute. The line driver's initialization, in turn, configures the line driver to the characteristics specified for its link.

The START command specifies a link to be started, and includes any optional parameters to override parameters in the link table entry. (The overrides are implemented only for this activation of the link; they do not replace the permanent "default" parameters in the link table or in the RSCS DIRECT entry.)

Loading the Line Driver, Function Description

1. Process the START command in REX module DMTCMX.

   a. DMTCMX validates the command parameters. It builds the link table "active" parameter entries, using the parameter default values in the link table, unless an override value is specified in the START command.

   b. Issue an ALERT to the LAX task to validate the line address specified for this link. If it is a switchable port, it must have an entry in the switchable port table (built at initialization from RSCS DIRECT data set PORT statement specifications). It must be a supported device type for the line driver to be loaded. If it is a leased line, it must not be already in use by another active link.

   c. LAX task completes the ALERT request for line validation by passing to DMTCMX a return code that specifies the result of the line validation.

   d. When the REX task is next dispatched, DMTCMX evaluates the validation return code and executes appropriate routines for the conditions in the code. If validation found the line to be satisfactory, DMTCMX calls REX module DMTCRE to create the

line driver task.

2. Load the line driver module specified for this link, using REX module DMTCRE.

   a. Call REX module DMTCOM, routine RCMSGET, to read the first text record of the module. On return, locate in that record the module's size.

   b. Find in MAINMAP (the byte-map of free and in-use virtual storage pages) the location of the number of contiguous free storage pages required for the module.

   c. Pass control to the supervisor storage manager module, DMTSTO, to reserve the desired pages in MAINMAP.

   d. When the REX task is redispatched, DMTCMX constructs the line driver module in the reserved storage with a series of requests for the module text records, performed by the RCMSGET routine in the DMTCOM module in the REX task.

   e. Initiate startup of the line driver as a task by passing control to the supervisor module DMTASK, giving the task name in register 0 and the address of the task save area in register 1. (The task name default is the first four characters of the linkid if there was no override value in the START command for the task.) If there is no task by this name already running, DMTASK proceeds to start this one. It builds a task queue entry for it, marking it dispatchable. Control passes to the REX task module DMTCMX, which issues an operator message that the link is started, and turns on the ACTIVE and CONNECT flags in the line driver task's link table.

   f. When the newly started task's line driver task gets dispatched, its initialization code executes.

Line Driver Task Initialization, Function Description

A line driver module is loaded and started as a task as part of starting a link, as described above.

Whenever a task is dispatched, the register contents and the PSW in the task's save area are loaded, and control passes to the instruction address that was in the PSW.

When the line driver task is dispatched after being loaded, the PSW assembled in its save area contains the address of its initialization routine. Register 0 is preinitialized during loading with a count of the parameter information, register 1 with the address of the parameter information, and register 2 with the task's link table address.

Every RSCS line driver contains, in its initialization routine code, logic for the following, with exceptions as noted:

   a. Save the address of its link table in its own storage.

   b. Copy its linkid into its own storage.

   c. Copy its local RSCS location's locid (contained in the linkid field of the first link table entry) into its own storage.

   d. Scan the parameter information, decode the parameters, and

configure itself to the link's requirements and specifications.

e. Build the file request element that this link will pass in its GIVE request element to AXS when it initiates or terminates processing of an input or output file.

f. Build tag prototype for this task, containing initial constant values (such as linkid) for any tag queue entry AXS will build for this task's files.

g. Issue asynchronous exit requests for interrupts (usually alert interrupts) that this link should receive through DMTASY.

h. Some line drivers need teleprocessing buffers. (The VMB line driver contains its own buffer). They call the REX module DMTCOM, routine GPAGEREQ to find the required number of pages in MAINMAP and get the storage manager module, DMTSTO, to reserve them.

In the NJI line driver, the DMTNIT module issues the buffer storage requests.

i. If the link has a BSC line, disable the BSC adapter, set its mode, and enable it. If the link has a channel-to-channel adapter, test its status by a SENSE command to determine the state of the remote system.

SML LINE DRIVER FUNCTION DESCRIPTIONS

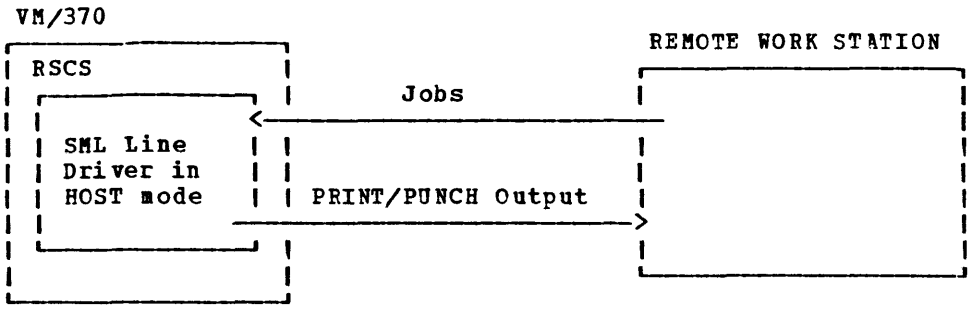
The SML line driver (DMTSML) provides binary synchronous communication (BSC) line protocol for programmable remote stations. DMTSML contains four types of routines:

- Processors (routines) that execute the functions required by the processing modes.
- An input/output routine that accepts and transmits data on the BSC line.
- A function selector routine that dispatches one of the processors when a request for services is received.
- Buffer blocking and deblocking routines.

Figure 3-6, "Program Organization for the SML Line Driver Task," shows the functional relationships among these routines.

With programmable remote stations, the SML line driver operates as a host (HOST mode) or as a remote job entry station (RJE mode). In HOST mode a remote station may submit jobs to VM/370 and receive print and punch output from VM/370. In RJE mode, VM/370 may send jobs to a remote batch system for processing and receive print and punch output from the remote batch system. Figure 2-11 shows the types of data flowing to and from RSCS via the SML Line Driver.

HOST Mode:



RJE Mode

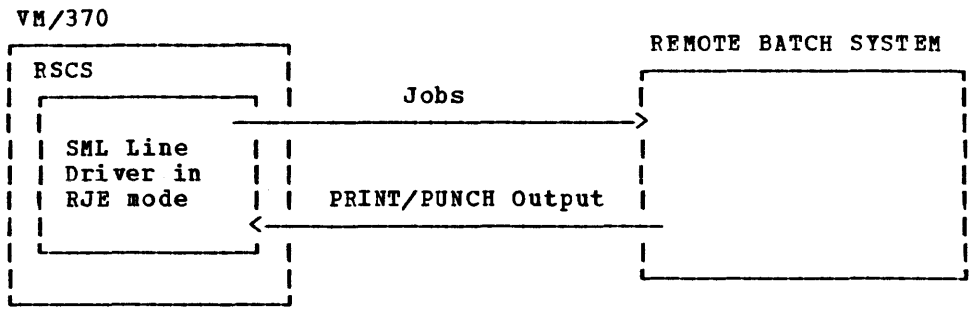


Figure 2-11. SML Line Driver Data Flow to Remote Stations and Systems

SML Processors

The SML program provides eight "processors," or routines, designed to handle the eight functions required to support the processing modes. Figure 2-12 is a list of the SML processors, the processing modes they support, and a brief statement of their functions. The DMTSML program determines its mode of operation by testing a byte, SMLSYS, which is set during SML line driver initialization. SMLSYS set to one of the following values:

X'80' - RJE mode, HASP
X'40' - RJE mode, ASP
X'20' - RJE mode, VS1/JES
X'10' - HOST mode

Processor	Mode	Function
$CRTN1	Both	Processes MULTI-LEAVING control records: permission to transmit, request to transmit, and SIGNON control records.
$PRTN1	RJE	Processes print file records from remote stations and passes them to the VM/370 spool system.
$URTN1	RJE	Processes punch file records from remote stations and passes them to the VM/370 spool system.
$JRTN1	HOST	Processes job file records from the remote station and passes them to the VM/370 spool system.
$WRTN1	Both	In HOST mode, passes command request elements via DMTMGX to DMTCMX. In RJE mode, passes message request elements to the RSCS operator's console.
$RRTN1	Both	Receives records from the VM/370 spool system for transmission to remote stations.
CMDPROC	Both	Executes local commands passed by DMTCMX; passes messages and commands to remote stations.
MSGPROC	Both	Sends messages to remote stations.

Figure 2-12. SML Function Processors

SML Command Processor: $WRTN1

When a command is transmitted from a remote station to RSCS, SML receives the command and coordinates processing of the command with Supervisor routines and the REX task command module DMTCMX.

The SML command processor, $WRTN1, processes a command request from a remote station by passing a command request element to the REX task (module DMTCMX) via a GIVE request. DMTCMX then determines whether the command should be executed by DMTCMX, DMTAXS, or the line driver. If the command is to be executed by the line driver, it is passed back to SML via an alert request. SML routine CMDPROC then executes the command.

SML Line I/O Manager: COMSUP

The SML line I/O handler routine, COMSUP, controls communications on the BSC line for SML. This routine receives data from the BSC line and passes it to the deblocker routine ($TPGET). COMSUP also sends data (which has been blocked by the blocker routine, $TPPUT) to a remote station. COMSUP also acknowledges receipt of data over the line using the standard BSC line control characters.

SML Function Selector Routine: $START

The $START routine is entered when SML is required (by either a remote station or a virtual machine) to perform a function. This routine selects a function to execute by using a "Commutator Table", a list of synch locks, and "Task Control Tables".

Each processor except MSGPROC and CMDPROC has a TCT (task control table) which contains necessary control information. Also, contained within the TCT is a branch instruction to the appropriate processor.

The SML commutator table is a branch table consisting of branch (BC) and no-operation (NOP) instructions. The targets of the branch instructions are the TCTs for the eight processor routines, COMSUP, and $TPGET, each of which performs a specific function. When the service of a routine is not required, the commutator table entry for that routine is a NOP instruction. When the function of the routine is required, the NOP instruction in the commutator table entry for that routine is replaced with a Branch instruction, thereby "opening a gate" in the commutator table.

The $START routine cycles through the commutator table, falling through any NOP instructions and taking any branches. Control is passed in this way to any routine whose gate in the commutator table is open.

When the routine completes the function requested, it "closes" its gate in the commutator table by replacing the Branch instruction with a NOP instruction. $START continues cycling through the commutator table, taking any open branches.

Initially after line driver startup, the commutator table branch instruction for reading a card, $RCOM1, is not a NOP. This allows the line driver to immediately read an available file from CP spool as soon as the line driver is started. In this case, processor $RRTN1 gives an OPEN request to DMTAXS to start reading a file from CP spool. If there is no file to be opened, $RRTN1 "closes" the commutator table gate for $RCOM1.

When the bottom of the commutator table is reached, $START tests a series of synch locks to see if any have been posted, signifying a request for an SML function. If any synch lock is posted, $START opens the commutator table gate for the requested routine and goes to the top of the commutator table to start cycling through it again.

If the bottom of the commutator table is reached and there are no posted synch locks, SML discontinues processing by issuing a wait request via a call to the Supervisor module DMTWAT, waiting on a list of the synch locks. When any of the synch locks is posted, $START receives control, opens the appropriate gate, and starts cycling through the commutator.

Block and Deblock SML Teleprocessing Buffers: $TPPUT and $TPGET

Data received over the BSC line is placed in one of four teleprocessing (TP) buffers. The size of TP buffers is specified by a START command parameter and can be up to 1017 bytes.

Data contained in TP buffers is deblocked into "tanks," which are unit buffers of a specific size used to deblock the larger TP buffers. There are 15 tanks; these are allocated as they are needed by processors. The size of tanks is determined by MULTI-LEAVING control bytes.

When an SML function has been requested, the data must be either blocked for transmission (if it is data for a remote station) or deblocked for processing (if it has been received from a remote station).

$TPGET receives data from a BSC line (via the COMSUP routine) and allocates tanks to output processors as they are needed.

$TPPUT receives tanks from input processors, blocks the data in these tanks into TP buffers, and gives control to COMSUP to transmit the buffers over the line.

SML File Send (Transmit Input Spool File on Link) Function

Assume a file to be sent to a remote node has been spooled to RSCS. The events that follow are:

1. RSCS is informed of the presence of the file. DMTAXS accepts it by issuing an alert to the appropriate link's line driver task (DMTSML).

2. DMTSML's exit routine (ASYNEXIT) posts DMTSML's reader device synch lock (RDEVSYNC), and control is returned to the dispatcher.

3. DMTSML is dispatched and the function selector routine, $START, is entered. $START finds that the synch lock RDEVSYNC has been posted and opens the gate for card reading ($RCOM1) in the commutator table (see description of $START, above).

4. With the gate opened, $RRTN1 reads the VM/370 spool file, deblocks (reconstructs) it into original record sizes, reblocks them into tanks for transmission, and gets them transmitted:

   a. $RRTN1 issues a GIVE to DMTAXS, requesting a spool file be opened and requesting a virtual reader be defined for the line driver itself.

   b. On successful return from DMTAXS, $RRTN1 extracts tag information from the spool file block (SFB) to determine the required transmission type (e.g., print or punch), and the transmission mode (HOST or RJE). This information is used to prepare the transmission record control byte (RCB), transmitted with each block of data.

      If the file was not successfully opened, the gate in the commutator table is closed and control returns to $START.

   c. $TPOPEN is called, requesting permission to transmit ($TPOPEN does this by transmitting a Request Control Record). If permission is granted, the message DMTSML146I is issued to indicate that the file is being transmitted.

   d. VMDEBLOK is called. It reads VM/370 spool file blocks or pages into page buffers, and issues subsequent reads as records in the buffers are used up. The records are deblocked according to the CCW information embedded with the data during spooling. VMDEBLOK returns to the caller a deblocked record that is placed in a 136-byte field (RCTTDTA1) within the tank.

   e. The record given by VMDEBLOK is combined with the proper MULTI-LEAVING transmission control bytes in preparation for transmission.

f. The record and its transmission control bytes are passed to $TPPUT via a call. $TPPUT compresses the record into a "line" that complies with the MULTI-LEAVING transmission protocol. The compressed line is packed into a teleprocessing buffer. When the buffer is filled, it is queued to the $OUTBUF chain for processing by COMSUP.

g. COMSUP initiates the actual I/O activity to transmit TP buffers. COMSUP dequeues buffers from $OUTBUF for transmission and calls DMTIOMRQ to initiate the I/O operation. COMSUP's gate is opened when the adapter synch lock, ADAECB gets posted by an I/O interrupt.

h. The EOF condition is eventually reached as VMDEBLOK reads the file from CP spool, in response to requests from $RRTN1 for more records to send. As soon as EOF is reached: (1) Message DMTSML147I is issued, (2) a GIVE is issued to DMTAXS to close and purge the file, (3) $TPPUT is called to transmit a null record to inform the receiver of the EOF condition, (4) the gate in the commutator table is closed, and (5) control is returned to the start of $RRTN1 to process another file.

SML File Receive (Spool Output File Incoming on Link) Function

This process starts with COMSUP's gate being opened when its synch lock ADAECB gets posted by an adapter asynchronous I/O interrupt. COMSUP examines the BSC control characters on the TP buffers received. The three kinds of control characters received are:

1. DLE-ACK, which specifies there is no block transmitted with this buffer.

2. DLE-STX, which specifies start of text. This acknowledges that RSCS received a buffer containing data, and that RSCS, in response, can send an output buffer.

3. NAK, which specifies that the buffer sent by RSCS to a remote location was not correctly received.

The control character significant to COMSUP when receiving a file is the DLE-STX. Upon receipt of this character from the TP line, COMSUP starts the process of receiving a file by chaining the received buffer to the input buffer chain, and opening the gate for $TPGET in the commutator table.

The general sequence of events that occur is as follows:

1. $TPGET is entered; it deblocks received telecommunication buffers into tanks, selects the appropriate processor, queues the tank to the selected processor's TCTTANK queue. $TPGET operates as follows:

   a. Get a buffer from the $INBUF queue and look for a matching TCT to attach the buffer to by comparing RCBs.

   b. Get a tank to decompress a buffer into.

   c. Decompress the buffer into the tank.

   d. Chain the tank to the TCTTANK queue for the processor selected.

e. Open the commutator gate for that processor.

2. The selected processor is entered when $TPGET returns to the commutator. The processor selected is one of the following:

$PRTN1: This routine processes print files. It dequeues tanks queued to itself by COMSUP, obtains an output spool device, and outputs the tanks to a virtual printer as follows:

a. Call $GETTNK to obtain a tank.

b. Call DMTAXS to request for an output spool file to be opened and a printer device defined.

c. Set up the carriage control using information contained in the SRCB.

d. Call DMTIOMRQ to print a record to the VM/370 spool file.

e. Call DMTAXS to close the spool file when EOF is reached.

f. Return to the commutator ($START).

$URTN1: This routine processes punch files. It dequeues tanks queued to itself by COMSUP, obtains an output spool device, and outputs the tank to a virtual punch as follows:

a. Call $GETTNK to obtain a tank.

b. Call DMTAXS, requesting an output spool file be opened and a punch device defined.

c. Call DMTIOMRQ to punch a record to the VM/370 spool file.

d. Call DMTAXS to close the spool file when EOF is reached.

e. Return to the commutator ($START).

$JRTN1: This routine processes job files. It dequeues tanks queued to it by COMSUP, obtains an output spool device, and outputs the tank to a virtual punch as follows:

a. Obtain a tank from $GETTNK.

b. Call DMTAXS to have a spool file opened and a punch device defined.

c. Call $USREXIT to validate the information on the job file's ID card and to save any other text information on the ID card. This text information represents user commands, which will be passed to the REX task for processing. $USREXIT also fills in the JCTTOVM field of the processor's TCT.

d. Call DMTIOMRQ to punch the job file records to the spool file.

e. Call DMTAXS to close the spool file when EOF is reached.

f. Return to the commutator ($START).

POW LINE DRIVER FUNCTION DESCRIPTIONS

The POW line driver (DMTPOW) provides binary synchronous communication (BSC) line protocol for remote VSE/POWER systems. DMTPOW contains four types of routines:

- Processors (routines) that execute the functions required by the processing modes.
- An input/output routine that accepts and transmits data on the BSC line.
- A function selector routine that dispatches one of the processors when a request for services is received.
- Buffer blocking and deblocking routines.

Figure 3-6, "Program Organization for the POW Line Driver Task," shows the functional relationships among these routines.

Figure 2-13 shows data flowing to and from RSCS via the POW Line Driver.

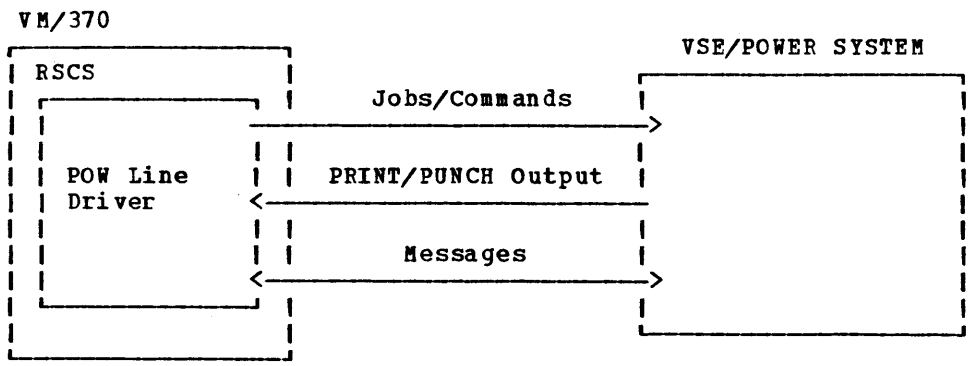


Figure 2-13. POW Line Driver Data Flow to a VSE/POWER System

POW Processors

The POW program provides eight "processors," or routines, designed to handle the eight functions required. Figure 2-14 is a list of the POW processors and a brief statement of their functions.

Processor	Function
$CRTN1	Processes MULTI-LEAVING control records: permission to transmit, request to transmit, and SIGNON control records.
$PRTN1	Processes print file records from the remote VSE/POWER system and passes them to the VM/370 spool.
$URTN1	Processes punch file records from the remote VSE/POWER system and passes them to the VM/370 spool.
$WRTN1	Writes received VSE/POWER commands to the RSCS operator's console.
$RRTN1	Receives records from the VM/370 spool system for transmission to the remote VSE/POWER system.
$MRTN1	Writes received VSE/POWER messages to the RSCS operator.
CMDPROC	Executes local commands passed by DMTCMX; passes commands to the remote VSE/POWER system.
MSGPROC	Sends operator and system messages to the remote VSE/POWER system.

Figure 2-14. POW Function Processors

POW Line I/O Manager: COMSUP

The POW line I/O handler routine, COMSUP, controls communications on the BSC line for POW. This routine receives data from the BSC line and passes it to the deblocker routine ($TPGET). COMSUP also sends data (which has been blocked by the blocker routine, $TPPUT) to the remote system. COMSUP acknowledges receipt of data over the line using the standard BSC line control characters.

POW Function Selector Routine: $START

The $START routine is entered when POW is required by a remote VSE/POWER system to perform a function. This routine selects a function to execute by using a "Commutator Table", a list of synch locks, and "Task Control Tables".

Each processor except MSGPROC and CMDPROC has a TCT (task control table) which contains necessary control information. Also, contained within the TCT is a branch instruction to the appropriate processor.

The POW commutator table is a branch table consisting of branch (BC) and no-operation (NOP) instructions. The targets of the branch instructions are the TCTs for the eight processor routines, COMSUP, and $TPGET, each

of which performs a specific function. When the service of a routine is not required, the commutator table entry for that routine is a NOP instruction. When the function of the routine is required, the NOP instruction in the commutator table entry for that routine is replaced with a Branch instruction, thereby "opening a gate" in the commutator table.

The $START routine cycles through the commutator table, falling through any NOP instructions and taking any branches. Control is passed in this way to any routine whose gate in the commutator table is open.

When the routine completes the function requested, it "closes" its gate in the commutator table by replacing the Branch instruction with a NOP instruction. $START continues cycling through the commutator table, taking any open branches.

When the bottom of the commutator table is reached, $START tests a series of synch locks to see if any have been posted, signifying a request for a POW function. If any synch lock is posted, $START opens the commutator table gate for the requested routine and goes to the top of the commutator table to start cycling through it again.

If the bottom of the commutator table is reached and there are no posted synch locks, POW discontinues processing by issuing a wait request via a call to the Supervisor module DMTWAT, waiting on a list of the synch locks. When any of the synch locks is posted, $START receives control, opens the appropriate gate, and starts cycling through the commutator.

POW Asynchronous Alert Exit Routine: ASYNEXIT

This is an alert exit entered by DMTSIG when a message or command has been entered for the POW line driver to process, or, a reader file has arrived for transmission to the remote system. It operates as follows:

1. Test if the alerting task is DMTAXS or DMTREX.
2. If the alert is from DMTAXS, call DMTPST to post the RDEVSYNC synch lock and exit.
3. If the alert is from DMTREX, determine if it is for a command or message.
4. If it is a command and a previous command is not being processed, accept the command, post CMDECB, and exit. Otherwise, indicate to DMTREX that the command is refused, and exit.
5. If it is a message, call MFI in DMTCOM to stack the message for later processing, post MSGECB, and exit.

Block and Deblock POW Teleprocessing Buffers: $TPPUT and $TPGET

Data received over the BSC line is placed in one of four 512-byte teleprocessing (TP) buffers.

Data contained in TP buffers is deblocked into "tanks," which are unit buffers of a specific size used to deblock the larger TP buffers. There are 15 tanks; these are allocated as they are needed by processors.

When a POW function has been requested, the data must be either blocked

for transmission (if it is data for a remote VSE/POWER system), or deblocked for processing (if it has been received from a remote VSE/POWER system).

$TPGET receives data from a BSC line (via the COMSUP routine) and allocates tanks to output processors as they are needed.

$TPPUT receives tanks from input processors, blocks the data in these tanks into TP buffers, and gives control to COMSUP to transmit the buffers over the line.

POW Control Record Processor: $CRTN1

This routine performs the processing required for MULTI-LEAVING control records. It functions as follows:

1. Get a control tank from the queue; if none are available, close the $CRTN1 gate in the commutator and exit to the next commutator entry.

2. If a tank is available, remove it from the queue and open the gate for $TPGET to get another tank.

3. Based on the bits contained in the RCB, branch to the appropriate sub-processor. The following table lists the routines that can be entered.

RCB	Routine	Reason for Entry
80	MC0	Not supported; exit
90	MC1	Start function request
A0	MC2	Start function permission
B0	MC3	Not supported; exit
C0	MC4	Not supported; exit
D0	MC5	Not supported; exit
E0	MC6	Not supported; exit
F0	MC7	General control record

MC1 - This sub-processor is entered when VSE/POWER transmits a "Request to Initiate Function" record. The function to be started is contained in the SRCB. A search is made of all the TCTs to find a match on the function code. If none is found, exit. If a matching TCT is found, the RCB in the received record is changed to X'A0' (Permission Granted) and $TPPUT is called to send the record to VSE/POWER.

MC2 - This sub-processor is entered when VSE/POWER transmits "Permission Granted" to a "Request to Initiate Function" from RSCS. A search is made of all the TCTs to find a match on the function code contained in the SRCB. If none is found, exit. If a matching TCT is found, its gate in the commutator is opened to allow it to begin processing.

MC7 - This sub-processor is entered when a general control record is received by RSCS. This type of control record contains a sub-function code in the SRCB. This code is an identification character between A and Z (EBCDIC). Only A (initial signon) is supported. DMTPOW checks the record for "PSIGNONxxx" or "PCOMPLETE". If PSIGNON, $TPPUT is called to transmit "PREADYnnn". If PCOMPLETE, message DMTPOW905I is

issued and a switch is set to indicate signon complete. Otherwise, error message DMTPOW902E is issued and the line driver is deactivated.

POW File Send (Transmit Input Spool File on Link) Function

Assume a file to be sent to a remote node has been spooled to RSCS. The events that follow are:

1. RSCS is informed of the presence of the file. DMTAXS accepts it by issuing an alert to the appropriate link's line driver task (DMTPOW).

2. DMTPOW's exit routine (ASYNEXIT) posts DMTPOW's reader device synch lock (RDEVSYNC), and control is returned to the dispatcher.

3. DMTPOW is dispatched and the function selector routine, $START, is entered. $START finds that the synch lock RDEVSYNC has been posted and opens the gate for card reading ($RCOM1) in the commutator table (see description of $START, above).

4. With the gate opened, $RRTN1 reads the VM/370 spool file, deblocks (reconstructs) it into original record sizes, reblocks them into tanks for transmission, and gets them transmitted:

   a. $RRTN1 issues a GIVE to DMTAXS, requesting a spool file be opened and requesting a virtual reader be defined for the line driver itself.

   b. On successful return from DMTAXS, $RRTN1 extracts tag information from the spool file block (SFB) to determine the spool file type (console, print, or punch). If the file is not a punch file, message DMTPOW940E is issued and the file is purged.

      If the file was not successfully opened, the gate in the commutator table is closed and control returns to $START.

   c. $TPOPEN is called, requesting permission to transmit ($TPOPEN does this by transmitting a Request to Initiate Function Record). If permission is granted, the message DMTPOW146I is issued to indicate that the file is being transmitted.

   d. VMDEBLOK is called. It reads VM/370 spool file blocks or pages into page buffers, and issues subsequent reads as records in the buffers are used up. The records are deblocked according to the CCW information embedded with the data during spooling. VMDEBLOK returns to the caller a deblocked record that is placed in a 136-byte field (RCTTDTA1) within the tank.

   e. The record given by VMDEBLOK is combined with the proper POWER MULTI-LEAVING transmission control bytes in preparation for transmission.

   f. The record and its transmission control bytes are passed to $TPPUT via a call. $TPPUT compresses the record into a "line" that complies with the POWER MULTI-LEAVING transmission protocol. The compressed line is packed into a teleprocessing buffer. When the buffer is filled, it is queued to the $OUTBUF chain for processing by COMSUP.

   g. COMSUP initiates the actual I/O activity to transmit TP

buffers. COMSUP dequeues buffers from $OUTBUF for transmission and calls DMTIOMRQ to initiate the I/O operation. COMSUP's gate is opened when the adapter synch lock, ADAECB gets posted by an I/O interrupt.

h. The EOF condition is eventually reached as VMDEBLOK reads the file from CP spool, in response to requests from $RRTN1 for more records to send. As soon as EOF is reached: (1) Message DMTPOW147I is issued, (2) a GIVE is issued to DMTAXS to close and purge the file, (3) $TPPUT is called to transmit a null record to inform the receiver of the EOF condition, (4) the gate in the commutator table is closed, and (5) control is returned to the start of $RRTN1 to process another file.

POW File Receive Function

This process starts with COMSUP's gate being opened when its synch lock ADAECB gets posted by an adapter asynchronous I/O interrupt. COMSUP examines the BSC control characters on the TP buffers received. The three kinds of control characters received are:

1. DLE-ACK, which specifies there is no block transmitted with this buffer.

2. DLE-STX, which specifies start of text. This acknowledges that RSCS received a buffer containing data, and that RSCS, in response, can send an output buffer.

3. NAK, which specifies that the buffer sent by RSCS to a remote location was not correctly received.

The control character significant to COMSUP when receiving a file is the DLE-STX. Upon receipt of this character from the TP line, COMSUP starts the process of receiving a file by chaining the received buffer to the input buffer chain, and opening the gate for $TPGET in the commutator table.

The general sequence of events that occur is as follows:

1. $TPGET is entered; it deblocks received telecommunication buffers into tanks, selects the appropriate processor, queues the tank to the selected processor's TCTTANK queue. $TPGET operates as follows:

   a. Get a buffer from the $INBUF queue and look for a matching TCT to attach the buffer to by comparing RCBs.

   b. Get a tank to decompress a buffer into.

   c. Decompress the buffer into the tank.

   d. Chain the tank to the TCTTANK queue for the processor selected.

   e. Open the commutator gate for that processor.

2. The selected processor is entered when $TPGET returns to the commutator. The processor selected is one of the following:

$PRTN1: This routine processes print files. It dequeues tanks queued to itself by COMSUP, obtains an output spool device, and outputs the tanks to a virtual printer as follows:

a. Call $GETTNK to obtain a tank.

b. Test for end-of-file. If not EOF, continue processing. If EOF exists and the file was previously opened, issue message DMTPOW145I and close the file via a call to DMTAXS. Otherwise, free the tank (see step g).

c. Convert the "CTR" characters into a CCW opcode and remove them from the record. If the resultant opcode indicates a special VSE/POWER CCW (X'FF' or X'FD'), free the tank.

   Note: "CTR" refers to the VSE/POWER command code that is contained in the first two bytes of all data transmitted from VSE/POWER to RSCS and in commands and messages sent from RSCS to VSE/POWER. The CTR is the expanded hex format (X'0B' becomes F0FB) of the CCW opcode that will be used to print the data. For commands, CTR is 00.

d. If the opcode is a NOP (X'03'), this indicates the possibility of a VSE/POWER MLX record in the data. If the record is not MLX1, free the tank. Otherwise, close the file via a call to DMTAXS (if it was open).

   Note: MLX records are internal VSE/POWER control records that allow VSE/POWER to send JECL information to another VSE/POWER system. These records are contained at the beginning of all list and punch files sent from VSE/POWER to RSCS. RSCS extracts the class, copy count, and number of records from the first of the three MLX records. These records are transmitted with a NOP CCW opcode (X'03') by VSE/POWER and are not printed or punched in the output produced by RSCS.

e. Extract the output class, number of copies, and number of records from the MLX1 record, open the spool file via a call to DMTAXS, and issue message DMTPOW144I.

f. Using the CCW opcode obtained from the "CTR" characters, send the output line to the VM/370 spool file via a call to DMTIOMRQ.

g. Free the tank - remove the current tank from the queue, open the commutator gate for $TPGET, and return to the beginning of the processor.

$URTN1: This routine processes punch files. It dequeues tanks queued to itself by COMSUP, obtains an output spool device, and outputs the tank to a virtual punch as follows:

a. Call $GETTNK to obtain a tank.

b. Test for end-of-file. If not EOF, continue processing. If EOF exists and the file was previously opened, issue message DMTPOW145I and close the file via a call to DMTAXS. Otherwise, free the tank (see step g).

c. Convert the "CTR" characters into a CCW opcode and remove them from the record.

d. If the opcode is a NOP (X'03'), this indicates the possibility of a VSE/POWER MLX record in the data. If the record is not MLX1, free the tank. Otherwise, close the file via a call to DMTAXS (if it was open).

e. Extract the output class, number of copies, and number of records from the MLX1 record, open the spool file via a call

to DMTAXS, and issue message DMTPOW144I.

f. Send the output line to the VM/370 spool file via a call to DMTIOMRQ.

g. Free the tank - remove the current tank from the queue, open the commutator gate for $TPGET, and return to the beginning of the processor.

$WRTN1: This routine writes received VSE/POWER commands to the RSCS operator. It functions as follows:

a. Close the commutator gate for this processor.

b. Call $GETTNK to obtain a tank.

c. If no tank is available, return to the commutator.

d. Remove the "CTR" characters from the incoming record.

e. Set up a message buffer and issue message DMTPOW944I via a call to DMTREX.

Note: This message will contain the received VSE/POWER command.

f. Remove the current tank from the queue and open the gate for $TPGET.

g. Return to the beginning of the routine and try to obtain another tank.

$MRTN1 This routine writes received VSE/POWER messages to the RSCS operator. It functions as follows:

a. Close the commutator gate for this processor.

b. Call $GETTNK to obtain a tank.

c. If no tank is available, return to the commutator.

d. Remove the "CTR" characters from the incoming record.

e. Set up a message buffer and issue message DMTPOW170I via a call to DMTREX.

Note: This message will contain the received VSE/POWER message.

f. Remove the current tank from the queue and open the gate for $TPGET.

g. Return to the beginning of the routine and try to obtain another tank.

POW Message Handler: MSGPROC

This routine is entered when the MSGECB is posted by the asynchronous exit routine, indicating messages are in the MSG queue for this line driver. It operates as follows:

1. Close the commutator gate for MSGPROC.

2. Call MFO in DMTCOM to remove a message from the queue. If none are available, send EOF to VSE/POWER via a call to $TPPUT and return to the commutator.

3. If the message is not from the system or the RSCS operator, issue message DMTPOW531I and return to the beginning of the routine to get another message.

4. If the message was from the RSCS operator, prefix the message with "DMTPOW170I from 'linkid'".

5. Put the message into the console tank and insert "CTR" prefix.

6. Call $TPOPEN to send a "Request to Initiate Function" transmission to VSE/POWER.

7. Call $PUT to send the actual message, then return to the beginning of the routine to get another messsage.

POW Command Handler: CMDPROC

This routine is called by $START when a wait request completes with the posting of the command arrival synchronization lock (CMDECB) following the acceptance of a command alert element from DMTREX. The command code table (CMDTABLE) is searched for a code matching that of the accepted element, and the command's processing routine is entered when a match is made. If no match is found, message DMTPOW531I is issued, and a return is made to the commutator, thus ignoring the command. The command processing routines are:

Routine	Command Processed
SETSTART	START
SETDRAIN	DRAIN
SETFREE	FREE
SETHOLD	HOLD
SETTRACE	TRACE
SETCMD	CMD

These individual command processors issue messages, set flags, and modify other line driver status, depending upon the particular command and the processing status of the line driver. When processing is complete, control is returned to CMDPROC which then returns to the commutator.

If the command specified was CMD, this indicates that DMTPOW must forward a command to VSE/POWER for execution. The processing is as follows:

1. Verify that the command was entered by the RSCS operator; if not, issue message DMTPOW531I and exit.

2. Syntax check the command text to ensure that the entered VSE/POWER command and its operands are supported for the RSCS-to-VSE/POWER connection. If not, message DMTPOW941E, 942E, or 943E is issued and an exit is made from the SETCMD routine.

3. Put the command into the console tank, move in "CTR" and "* .. " prefixes.

4. Call $TPOPEN to send a "Request to Initiate Function" transmission to VSE/POWER.

5. Call $PUT to send the command.

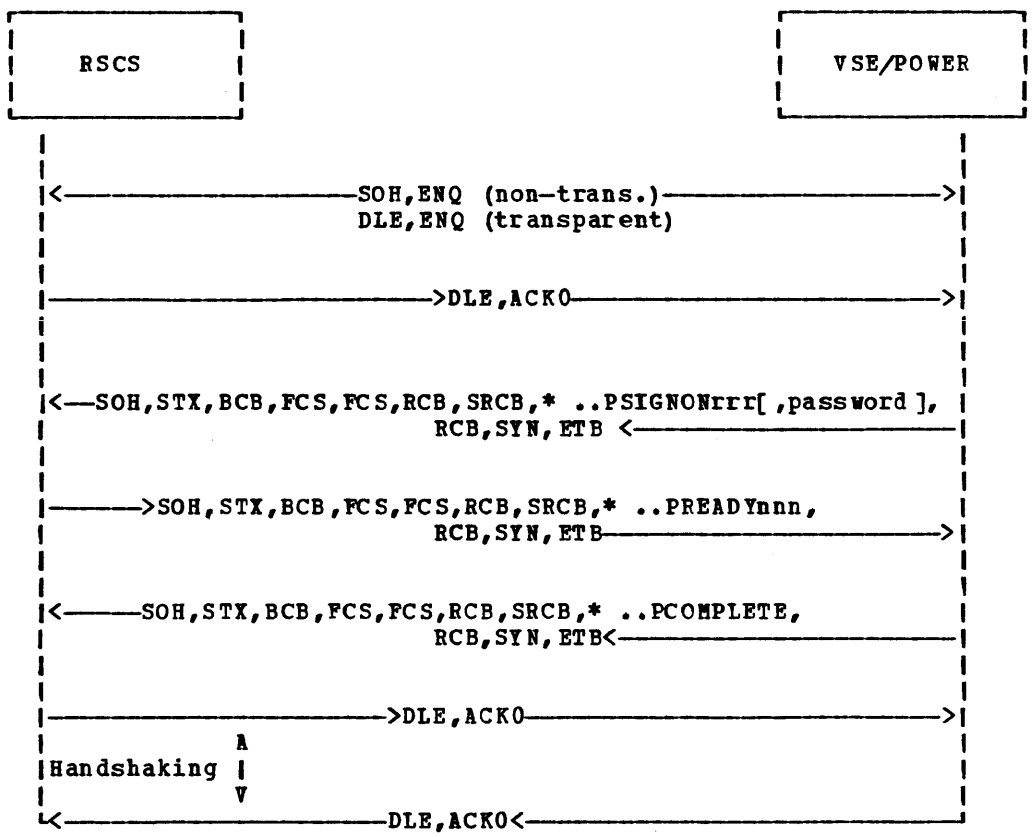
6. Call $TPPUT to send EOF.

7. Return control to CMDPROC which then returns to the commutator.

Typical RSCS to VSE/POWER Line Transmissions

Figures 2-15 to 2-20 show the character sequence used in the following:

- Signon procedure
- Initiation of a transmission
- Command transmission
- Stop procedure
- Transmission of text in one direction
- Transmission of text in both directions



rrr = global remote identifier generated within the POWER macro.
nnn = global remote identifier from the Cnnn parameter of RSCS START command.

Figure 2-15. Signon Procedure

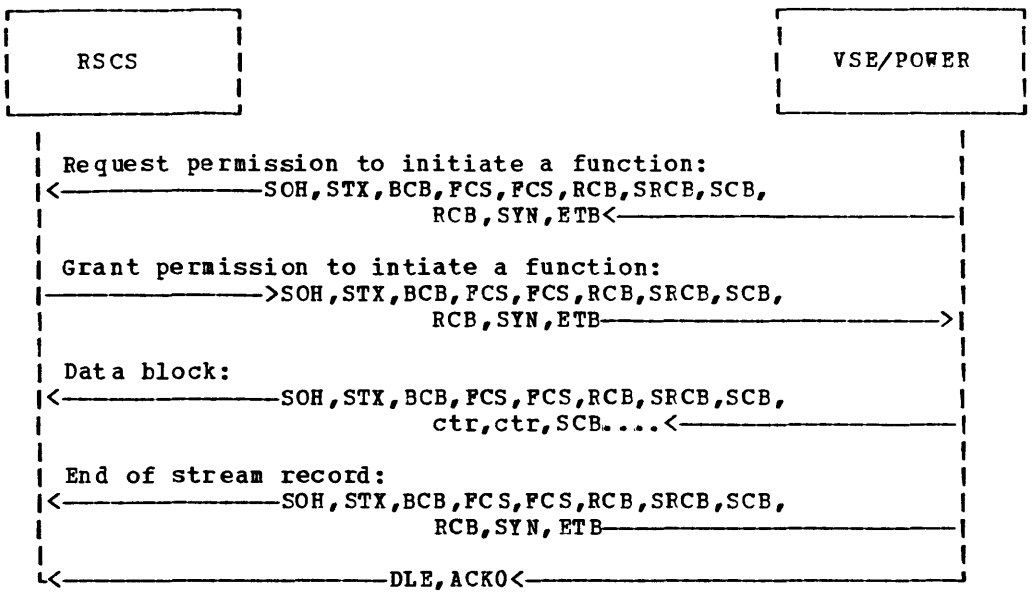


Figure 2-16. Initiation of a Transmission

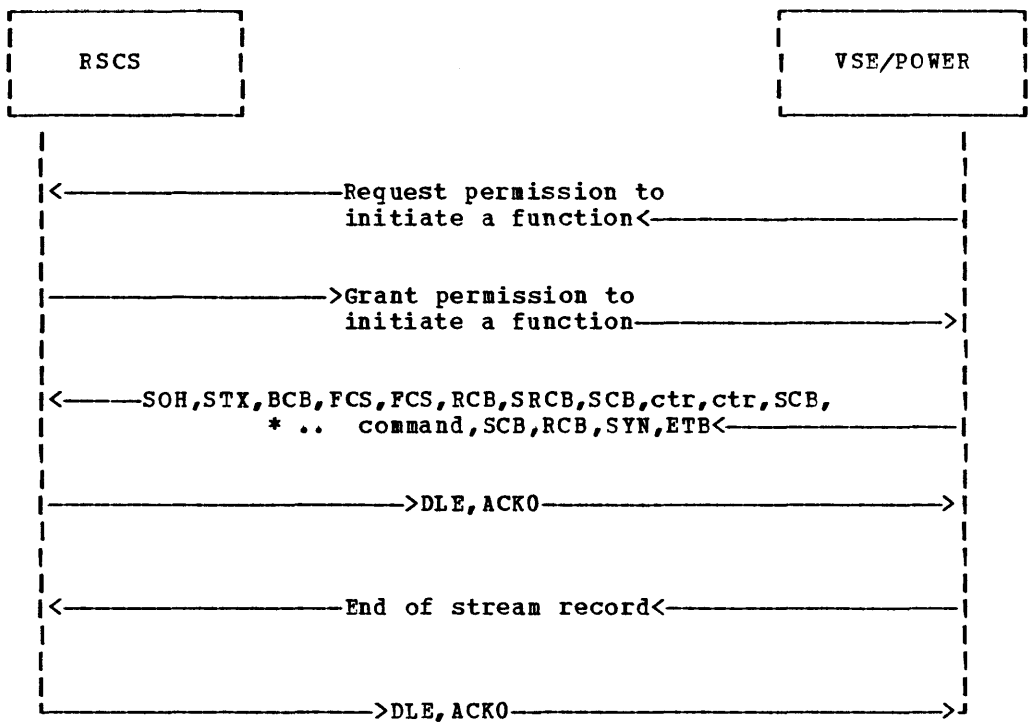


Figure 2-17. Command Transmission

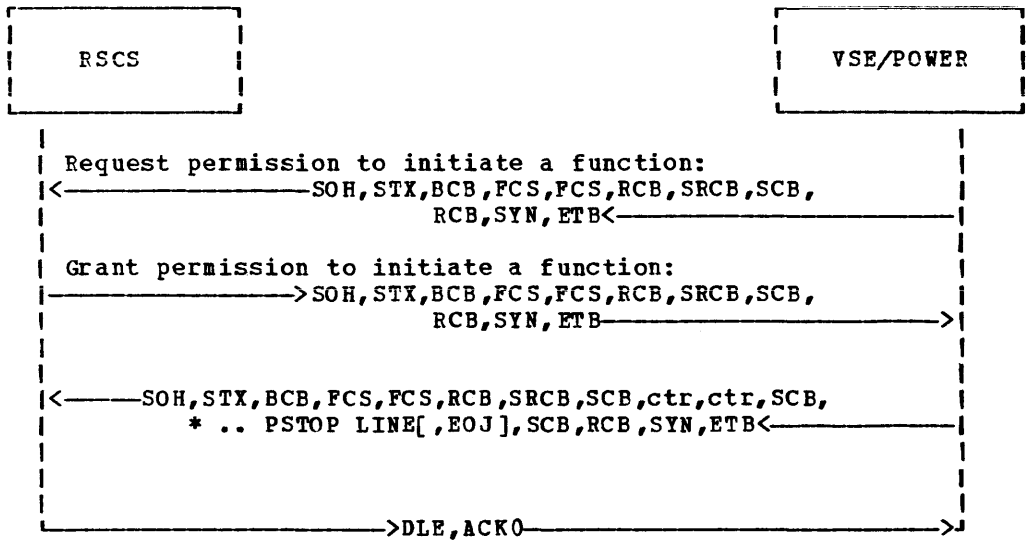


Figure 2-18. Stop Procedure

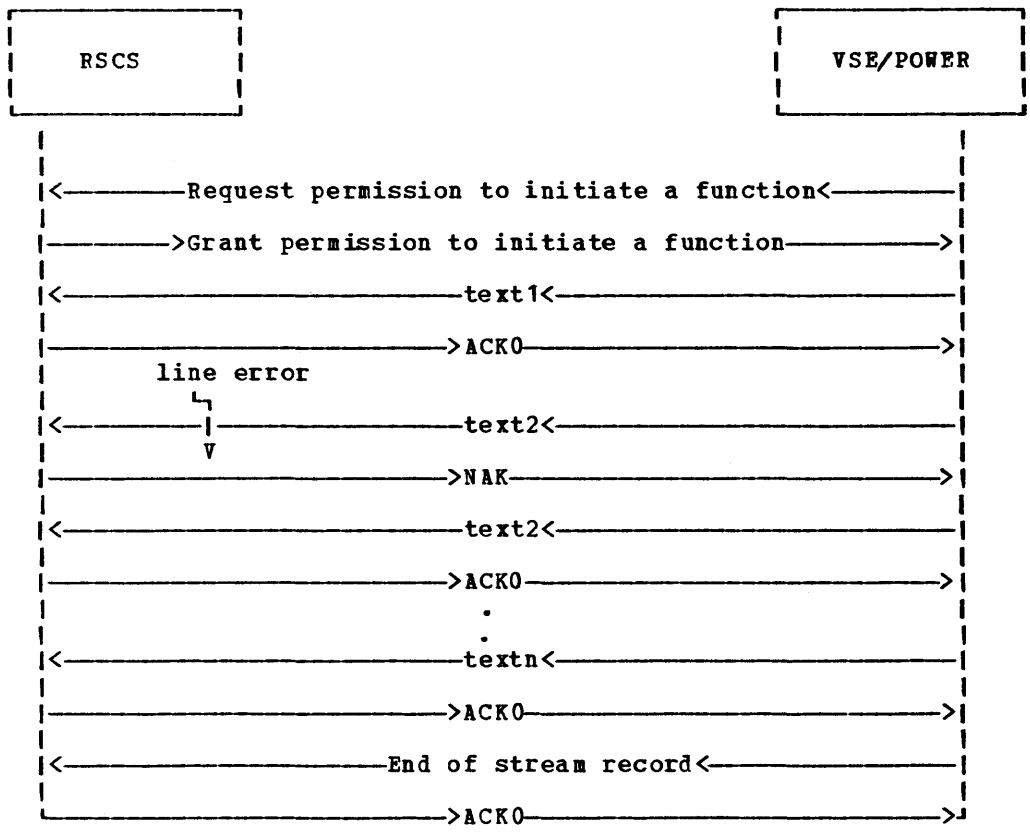


Figure 2-19. Text in One Direction

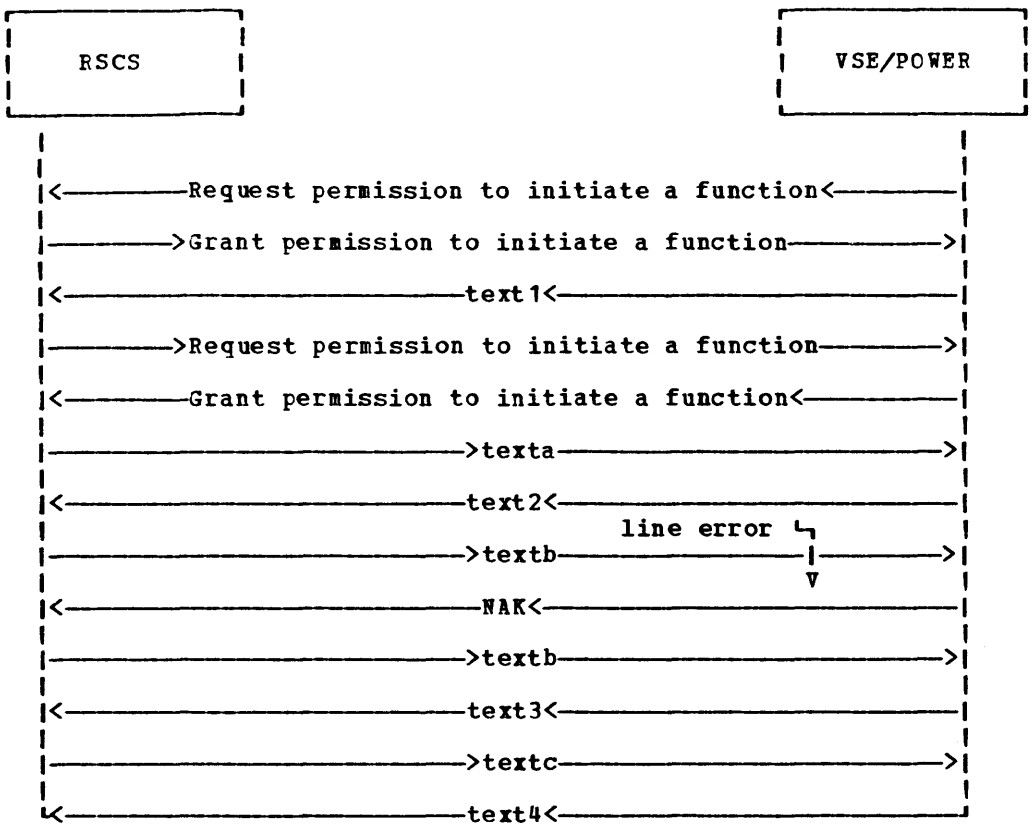


Figure 2-20. Text in Both Directions

NPT LINE DRIVER FUNCTION DESCRIPTIONS

The NPT line driver (DMTNPT) provides binary synchronous communication (BSC) line protocol for nonprogrammable remote terminals. This allows:

- Remote users of VM/370 to enter source decks, data, and jobs, on cards, into the VM/370 spool system
- VM/370 to send spooled output of virtual machine sessions to remote card punches and printers
- Remote stations to transmit card decks to one another
- Remote stations to send job streams to a CMS Batch virtual machine operating under the same VM/370 and have the output returned to the remote station
- Remote stations to submit jobs or commands to any node in the network and direct the output to any node or remote station in the network. The default is return to origin.

NPT is a line driver task operating under the control of the RSCS supervisor. Each NPT task drives one remote nonprogrammable station. In other words, each NPT task controls a single point-to-point communications line. The task is started by the RSCS operator, identified with a destination name, and provided with a leased or switched telephone line. The communications line is either identified

by the operator or derived from a table entry within RSCS. The line is then activated and the type of remote station and its configuration details are obtained from control cards entered at the remote station, or from a table entry within RSCS. After this initialization is done, the terminal may then be used to submit files via the card reader and receive files on the punch and printer.

The remote station operator can control I/O activity via control cards and standard station procedures. The RSCS operator controls the operation with commands from his console. The virtual machine user retrieves files sent to his virtual machine by using normal virtual card reader management programs and directs output to the appropriate station using the SPOOL and TAG commands of VM/370.

NPT operates with variations of the basic BSC protocol for each of the stations listed. The protocol is based upon the station identification information provided in a SIGNON card read at task initialization time.

NPT Line Driver Send Function

Check the status flags for the GETBLOCK processor:

BUFEMPTY: is the unpack buffer (BUFUNPK) used to fill the line buffer (LINEBUFF) empty?

HEADFLAG: is there a header record to be inserted into the line buffer?

FILACTIV: is there a file being transmitted now or does one have to be obtained from VM/370?

RDRCMD: are there any commands pending execution?

EOF: has end-of-file been reached for the file currently being processed?

1. Get a Spool File Block to Transmit

   Branch to GETFILE.

   Test to see if there is a file to send. If there is a file, request AXS to open it for transmission. When the file is obtained determine whether it is a print or punch file, initialize data counters, buffers, and registers, and write message DMTNPT146I, indicating that a file is being sent. If there are errors, write appropriate messages.

2. Write Header Record: HEADPREP

   If the header transmission flag is on, a printer header or printer/punch separator line is to be inserted into LINEBUFF via a branch to HEADPREP.

   HEADPREP tests for the type of file (print or punch) and inserts the appropriate line into the unpack buffer (BUFUNPK). On return from HEADPREP, the unpack buffer is packed for transmission on the BSC line.

3. Process Spool File Block for transmission: MAKEBLOC

a. At GETLINE, if the BUFEMTPY flag is off, a new record is ready to be inserted into the line buffer. Check the maximum byte count and record count allowed in LINEBUFF and see if this record exceeds the maximum. If this record exceeds the maximum counts, the appropriate BSC control characters are inserted into LINEBUFF to complete the transmission sequence and control is passed back to the caller.

   If the record can be inserted in LINEBUFF, it is moved into LINEBUFF along with the appropriate BSC control characters. BUFEMTY is then turned on and a new record obtained via a branch to GETNEW.

b. At VMSPOK, initialize counter registers and set FILLED flag on in GETFLAGS. If there is already a block to process an entry to MAKEBLOC, update the counter registers.

c. At VMSPCCW, process the spool block record. Move the record into BUFUNPK. In the case of an immediate CCW command, only the CCW command is moved into BOFUNPK and IMDCMD flag is set on.

d. At MAKERET, after the line has been converted for transmission a 0 is set as return code if there is more data to process. On end-of-file, a 4 is set as the return code. If there was an error during processing, message DMTNPT190E is written.

4. Transmit Line: NPTGET

a. At NPTGET, check for messages and try to get a block of data to write from VM/370. If there are no messages to process, branch to GETBLOCK to get a block of VM/370 data to transmit. On return from GETBLOCK with no data to transmit, determine whether there are files to read.

b. At NPTSTART, on return from GETBLOCK with data to transmit, initialize the output buffer (LINEBUF) by:

   (1) Testing flags in the GETFLAGS table and Device Block, and

   (2) Moving into the Device Block the BSC control characters describing the features of the remote station to which a file is to be transmitted.

   Load the ENQPROG channel program into the Device Block and branch to LINEIO to execute an I/O operation requesting permission to transmit to the Remote Station. Test for the line errors resulting from the I/O operation.

c. At NPTENQOK, when the response to ENQ is correct, (DLE, ACK0) initialize control counters (constants and registers) for transmission of the file. EXREPLY contains the expected line error checking via BSC line protocol. DCX and INDEVSEL contain data control characters for printer and punch devices. Move the TALKPROG channel program into the Device Block. Set the return address to the GETVRFY routine.

d. At NPTTALK, execute a write I/O operation by branching to the LINEIO routine. Set the return address to the GETVRFY routine.

e. At NPTEOT, on end of file, move the EOTPROG channel program into the Device Block and branch to the LINEIO routine to send an end of transmission signal to the remote station.

5. Verify Response to Transmission: GETVRFY

   a. Check the response from the last transmission for:

      EXREPLY
      Either ACK0 or ACK1, in the normal BSC transmission protocol.

      NAK
      Negative acknowledgement.

      EOT
      End of text - used to specify that the transmission is incomplete. Also, used to notify the receiving station that there are errors receiving data at this end of the line.

      ENQ
      Used to request permission to transmit a file. Also, used to "balance" the line in recovering from errors.

   b. When the EXREPLY flag specifies that the expected reply has been received, the next expected reply is set (ACK0 or ACK1) and control is returned to NPTGET.

   c. When a NAK is received, the NAK counter (STATNAK) is incremented a test is made to see if this is the second NAK by checking the NAKREC flag. If this is the first NAK received, retransmit the block by branching back to NPTGET. If this is the second NAK, reset the line by sending an EOT and an ENQ.

   d. When an EOT response is received, try to send the block again.

   e. At REPENQ, when an ENQ is received, check for I/O errors, set ACK1 as the next response, and branch to NPTGET to continue processing.

      ENQ is sent to request permission to transmit or to resume transmission of a file after errors on the line or timeouts. After multiple ENQ transmission, send EOT to reset the line.

6. Obtain each subsequent block of this file with DIAGNOSE X'14', subcode 0.

7. At GETPURGE, on end-of-file, write message DMTNPT147I, indicating that a file has been sent and purge the file via a branch to AXS.

NPT Line Driver Receive Function

TEST FOR REQUEST TO TRANSMIT FROM REMOTE STATION: At NPTGET, monitor the BSC line for tasks to perform.

1. Check for messages and try to get a block from VM/370 to write. If there are no messages to process or VM/370 files to transmit, read from the BSC line to determine whether a file is being transmitted from a remote station.

2. At NPTINIT, read a line of data.

   Load the READPROG channel program into the device table describing

the I/O operation and branch to LINEIO to execute the read. If there were no flags set in the DEVFLAGS table, there is nothing being transmitted on the BSC line. In this case, branch back to NPTGET to try to get a VM/370 spool file to transmit.

3. Test for request permission to transmit.

   When a line of data has been received over the BSC line, check the BSC control characters for a request to transmit. If such a request (ENQ) is not received, branch back to NPTDINIT to check for another operation to perform.

4. At NDTACK0, send permission to transmit.

   When an ENQ is received, load permission to transmit (ACK0) in the REPLY entry in the I/O Counter Table, load the ACKPROG channel program in the Device Table for this I/O operation and branch to LINEIO to transmit permission to transmit. On return from the I/O operation, check the results via a branch to PUTVRFY.

VERIFY READ OPERATION BSC DATA:

1. Check for I/O errors on the BSC line: PUTVRFY

   a. At PUTVRFY, check for I/O errors during the last transmission. If there are no errors, processing continues at CKBUFF.

   b. At NPTNAK, if there are any errors, check for timeout. If a timeout error, send a NAK. If the maximum NAKs allowed are sent, reset the line by sending an EOT. If the maximum number of timeouts has occurred, close the file via a branch to PUTCLOSE in the PUTBLOCK routine.

2. Check the BSC control characters.

   a. At CKBUFF, check the BSC control characters of the buffer just received: STX or DLE STX for the leader and ETB or ETX at end of buffer. If all characters are acceptable, branch to PUTBLOCK to continue processing.

   b. At NPTTALK, if the leader characters are not STX or DLE STX, check for ENQ, EOT, or NAK. If an ENQ is received, send ACK0 to the remote station.

      If an EOT or a NAK is received, set the EOTREC (EOT received) flag and branch to PUTBLOCK.

   c. At REPLY2, if the leader characters are unidentifiable, load the "not ready" program, send it, and check for I/O errors and an ENQ. If the reply is not ENQ, send an ACK0 back across the line.

   d. At NPTTALK1, if the reply is ENQ set the return address to PUTVRFY and go to LINEIO to write the next line. If there are I/O errors, exit to PUTCLOSE in the PUTBLOCK routine (REPLY3).

WRITE BLOCK OF DATA TO VM/370 SPOOL FILE:

1. Check status of file processing.

At PUTBLOCK, determine the status of file processing by checking for an already active file, end of file, an error in processing, and setting the correct response to the received transmission (ACK1 or ACK0). If a file is active, continue processing. On end of file, close the file. If there was an error in processing, resend the buffer via a branch to NPTTALK1.

2. Convert a line of data to VM/370 format.

   At TRT1, if a file is being processed, get the address of the line buffer and convert the characters to VM/370 format. When a block is completely reformatted, get another via a branch to NPTTALK1.

3. Determine whether the current record is a command.

   At FOUND, check the current record to see if it is a command. If the return code from the command processor is zero, branch to PUTSKIP to skip printing of the command line.

4. Validate the userid of an ID card.

   In FOUND, check the current record to determine if it is an ID card. If the record is an ID card, validate the card and move the VM userid into TAGTOVM, the target virtual machine for the file to be processed.

5. Open a file.

   At PUTOPEN, open the new file for processing by writing message DMTNPT144I, informing the operator that a file is being sent. Clear the request synchronization lock and request the supervisor GIVE request routine to handle the open request. On return from the GIVE processor, set the FILEOPEN flag and get the first block of data via a branch to NPTTALK1 (via PUTSKIP, where printing of the ID card is skipped).

6. Write a block of data to a VM/370 spool file.

   a. At PUTWRITE, write a block of data to a VM/370 spool file by decompressing the record, if required, loading the PUTPROG channel program, and branching to LINEIO to handle the I/O operation.

   b. At PUTSKIP, records that need not be written to the file (such as commands, ID cards, and blank records) are skipped.

      When a write operation is interrupted by EOT or an alert, processing is continued via a branch to PUTCLS4.

7. Close a file.

   At PUTCLOSE, close the file by writing message DMTNPT145I, informing the operator that a file has been received, clearing the synchronization lock, setting the close function code in the request block, and requesting the MSUP Supervisor GIVE request handler to close the file. If the file is not open, issue message DMTNPT934E.

VMB LINE DRIVER FUNCTION DESCRIPTION

The VMB line driver (DMTVMB) is for transmitting VM/370 spool files between VM/370 systems over BSC lines. DMTVMB communicates with another copy of itself using the file address specified on the VM/370 TAG command (location and userid) to determine the recipient virtual machine. DMTVMB supports both print and punch file transmission between users operating on two different VM/370 machines or transmission from a VM/370 user to a real unit record device on a remote VM/370 machine.

The VMB Line Driver employs a variation of standard BSC protocol similar to MULTI-LEAVING; it is transparent, symmetrical, and can sustain simultaneous two-way data transfer by interleaving block reception and transmission. All data records are compacted by reducing all sequences of five or more identical characters into a four-character sequence. Standard BSC transparency mode is used for data transmission to remove restrictions on data content. The protocol is symmetrical, with no master/slave relationship between the communicating systems.

VMB BSC Telecommunication Protocol

VMB protocol provides a quiesced state when neither system has data to transmit to the other. This provision avoids forcing the CPU cluster to an active state (thereby running the system meter) when communication is available but not actually active. Communication may be restarted by either of the connected systems when data arrives for transmission. Contention (simultaneous activation) is not a problem, because data flow can commence in both directions as soon as transmission exchange is synchronized.

Bidirectional interleaved half-duplex communication is achieved through the use of the block header for acknowledgment or rejection of the previously received transmission. The acknowledgment byte flag setting in the block header explicitly confirms successful reception and processing of the data transmitted by the remote system in the preceding line sequence. When a received block cannot be processed because system resources have been exhausted, an acknowledgment byte flag setting requests retransmission of the block.

If recoverable I/O errors occur during a block transmission, the receiver responds with a negative acknowledgment (NAK), requesting complete retransmission of the preceding sequence. Serious line I/O errors, negative acknowledgments to negative acknowledgments and long duration timeouts (>15 seconds) while waiting for response are conditions that terminate exchange synchronization. Exchange may then be restarted as from the quiesced state by either system with data to transmit.

BSC Transmission Sequences

BSC TRANSMISSION SEQUENCE WITHOUT DATA

AS TRANSMITTED -->

SYN, SYN, SYN, SYN, SOH, 00, 00, 00, 00, [(1)], ETB

AS RECEIVED -->

SOH, 00, 00, 00, 00, [(1)], ETB, (2)

BSC TRANSMISSION SEQUENCE WITH DATA

AS TRANSMITTED -->

SYN, SYN, SYN, SYN, SOH, 00, 00, 00, 00, [(1)], ITB, SYN, SYN,
'V', 'M', 'R', 'A', (3), (4 - FIVE BYTES), (5 - TWO BYTES),
DLE, STX, (6 - PACKED DATA), DLE, ETB

AS RECEIVED -->

SOH, 00, 00, 00, 00, [(1)], ITB, (2),
'V', 'M', 'R', 'A', (3), (4 - FIVE BYTES), (5 - TWO BYTES),
DLE, STX, (6 - PACKED DATA), DLE, ETB, (2)

Explanation of Underlined Digits in Above Sequences

1 Acknowledgment flag, present only when the preceding transmission from the remote station included a packed data block.

X'80' - Preceding block was successfully received and processed, and may be discarded by the sender.

X'A0' - Preceding block was successfully received, but could not be processed due to lack of available system resources - transmission must be retried later.

2 An error index byte that is generated by the BSC telecommunications adapter operating in "ITB Mode", to flag data errors detected by check sum mismatches.

3 A one-byte EBCDIC block sequence number, modulo 8 (X'F0'-X'F7'), used to detect possible duplicated transmissions so that the duplicated data may be discarded.

4 A five-byte EBCDIC data block content descriptor:

C'SYNCH' - Null data block - intended to be discarded. (This is used on line driver restart so that the initial data transmission will not be erroneously discarded due to a spurious block serial number match.)

C'PRINT' - Print image data.

C'PUNCH' - Card image data.

C'MSGHD' - Commands, messages, or link control records.

5 A two-byte formatted packed data count:

1 X X X X X X X	1 X X X X X X X
High-Order Count Bits	Low-Order Count Bits

The packed data count is constructed by discarding the high-order bit of each of the two count bytes and juxtaposing the remaining bits into a 14-bit binary count. (The high-order bit of each byte is always set to "1" to avoid possible duplication of a BSC control character, because only the packed data block itself is transmitted in transparency mode.)

6 One or more packed data records, to a maximum length of 824 bytes.

Figure 2-21 shows the protocol for transmission error retry and Figure 2-22 shows typical line transactions.

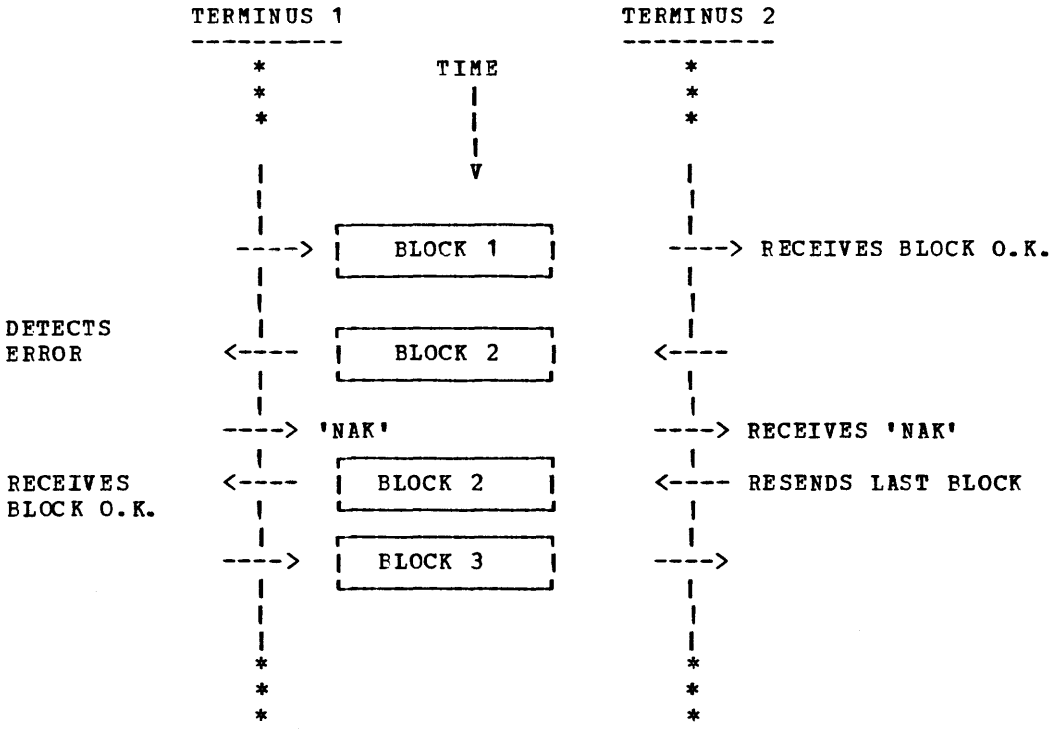


Figure 2-21. Protocol for Transmission Error Retry

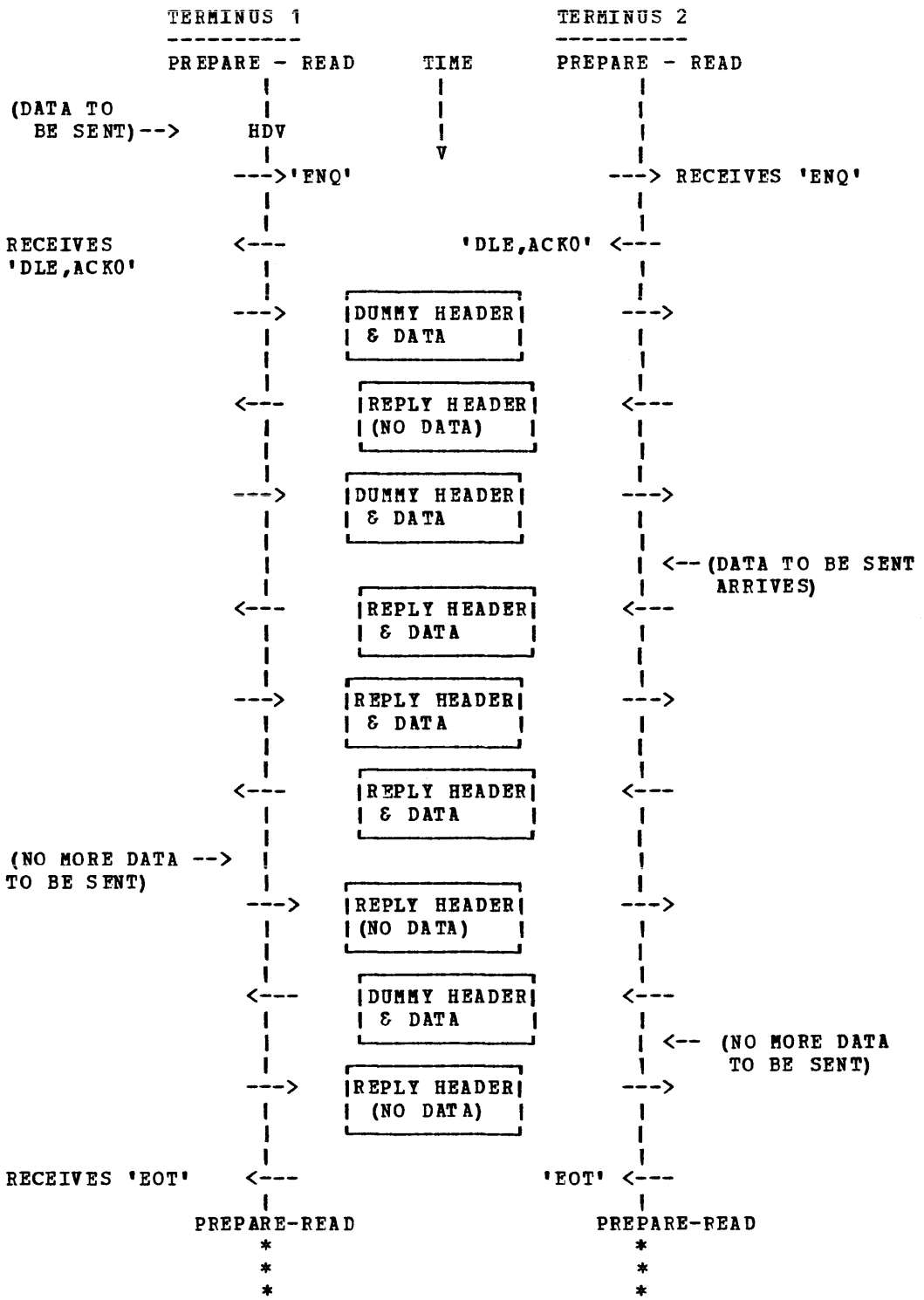


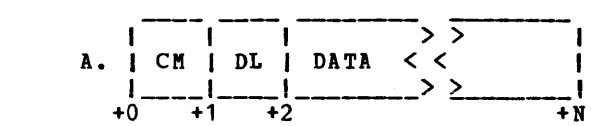
Figure 2-22. Typical Line Transactions

DMTVMB Packed Data Block Format

DMTVMB scans the output data and builds packed formatted blocks which are actually transmitted on the telecommunications lines. The data are packed by compressing all occurrences of five or more identical characters in a record into a coded field which adds only four bytes to the packed record. When the data blocks are received by the remote RSCS system, the records are correspondingly unpacked by DMTVMB before being entered as output in the VM/370 spool system.

Transmission data blocks are of variable length, depending on content, up to a maximum length of 824 bytes. Each block contains a variable integral number of packed records of variable length, each of which corresponds to a single unpacked record. There are no partial records in a block; if a packed record will not fit at the end of a block without extending the block beyond the 824-byte limit, the block is terminated and the record is placed at the start of the next block.

The packed data record format is:



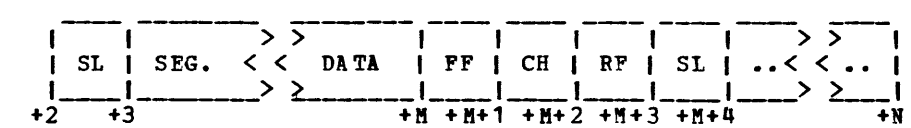
Where:

CM -- a System/370 Channel Command Word (CCW) command code used to output the record by the originating virtual machine.

DL -- number of bytes (in hexadecimal) in the packed DATA field.

DATA -- data comprising a single record in mixed packed and segment format as shown below.

The DATA field format is:



Where:

SL -- a one-byte hexadecimal count equal to one less than the length of the segment data, SEG. DATA.

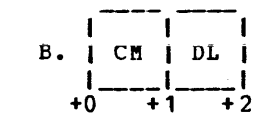
SEG. DATA -- a string of data to appear unmodified in the unpacked record.

FF -- a flag (X'FF') which indicates the start of a packed segment.

CH -- a data character to be replicated a number of times in the unpacked record.

RF -- a hexadecimal count equal to two less than the the total number of characters, all "CH", to appear in the unpacked record.

B.

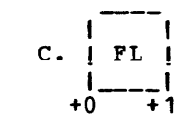


Where:

CM -- a System/370 immediate (non-data moving) control command code, for a print or punch output device.

DL -- a data count of X'00', which implies that the CM command is of the immediate control type.

C.



Where:

FL -- a flag byte which has one of the two following values:

X'FF' -- end of data block; more data blocks exist for the file.
X'EF' -- end of data block and end of file; no more data blocks exist for the file.

VMB Line Handling

A. VMRENABLE - This routine is entered after VMB initialization has completed, and each time the telecommunication line drops (signalling "intervention required") during VMB processing. It issues message DMTVMB141I to notify the operator that the line should be connected in case manual intervention is required (such as when a dialable port is in use), and starts execution of the line enabling sequence.

This enabling sequence comprises three command chained CCW's: DISABLE, SET MODE, and ENABLE.

The DISABLE command disconnects any previous dial port connection, and places the telecommunication adapter in the disabled state.

The SET MODE command places the telecommunication adapter in ITB mode, such that on subsequent read operations an incoming ITB BSC control character will be recognized, the following two BCC characters will be interpreted, and an EIB character reflecting presence or absence of errors will be entered into the read buffer.

The ENABLE command completes, placing the telecommunication adapter in the enabled state, when the port's modem signals "data set ready". For a dialable port, this occurs when the dial connection completes; and for a leased line, "data set ready" is signalled whenever the line and modem equipment are functional.

When this enabling sequence completes, control is passed to VMRSTART to verify link identifiers and passwords, and normal processing begins.

B. VMRGET - This routine is entered when VMB is in the quiesced communication state, and data may be available for transmission to the remote system. The hold and drain statuses are checked, and a call is made to GETBLOCK to prepare a data block for transmsission. If communication is allowed and a data block is ready, control is passed to VMRSTART to initiate the exchange. Otherwise, the read initial sequence (command chained PREPARE and READ commands) is started, and communication remains quiesced.

   The PREPARE command completes only when halted by the AXSALERT routine, or when the remote system initiates a transmission. In the former case, VMRGET is reentered from the beginning. In the latter case, a DLE-ACK0 sequence is written to the remote system; if signon has already occurred, control is passed to VMRVERFY (in the VMRGO routine) to begin processing incoming data. Otherwise, signon blocks are exchanged and verified before any data transfer is allowed to occur.

C. VMRSTART - This routine is entered upon successful completion of the line enabling process, and each time communication is to be reactivated from a quiesced state. The block exchange channel program is initialized, and an attempt is made to exchange the initial ENQ, DLE-ACK0, and signon sequences. When successful, control is passed to VMRCHARG (in the VMRGO routine) to initiate data exchange. If no response is received from the remote system for the duration of the ENQ retry sequence, control is passed to VMRDINIT (in the VMRGET routine) and communication remains quiescent.

D. VMRGO - This routine comprises the central control logic for normal VMB transmission-reception activity. At VMRGO, a transmission has been received from the remote system. If a NAK has been received, the preceding transmission is repeated. Otherwise, the received sequence is checked for validity, and a NAK is transmitted (at VMRNAK) if it is invalid. If a signon block is received, it is verified by a call to PASSCHEK, and a signon block is returned by a call to PASSSEND. If a data block is received, it is processed by a call to PUTBLOCK. The GETBLOCK routine is called to prepare a data block for transmission to the remote system, if any such data is available. If no data block is available for transmission, and no data block was received on the last transmission from the remote system, an EOT is transmitted at VMREOT and control is passed to VMRGET to quiesce communication.

   The main telecommunication channel program is executed at label VMRTALK. This channel program comprises several data chained write CCWs, command chained to a read CCW. It is built mainly by GETBLOCK, and its structure can include TICs, depending upon the presence of an acknowledgement byte and data block in the transmission to be made. When the exchange is successful, the transmission will have been written to the remote system, the response will have been read into the line input buffer (LINEBUFF), and control is passed back to VMRGO for another exchange cycle.

E. LINEIO - This routine is called to perform all I/O operations on the telecommunication port. The I/O request is executed as set in the line I/O table (LINE) by a call to XECUTE, and call is made to KLOGIT to log the results. If the I/O is successful, return is immediately made to the caller. If a serious error occurs, control is passed to VMRBADIO, which issues an error message and deactivates the line driver. When contention is detected, a read is executed, an error is flagged, and control is returned to the caller. If "intervention required" is detected, message 143 is issued and control is passed to VMRGET if no response is read. If an error is detected that can be corrected on retry, an error is

flagged, and control is returned to the caller.

F. TRTRAN, TRERR, TRTIMOT - The trace sum routines are called by the various line management routines to count the accumulated number of successful transmissions, line errors, and timeouts, respectively. These counts are constantly maintained in the line driver's link table, where they may be interrogated by an RSCS operator "QUERY linkid SUM" command. When sum tracing is active, message 149 is issued each time any of the counts reaches a threshold value, and all counts are reset to zero.

G. KLOGIT - This routine is entered after every line I/O transaction. When log trace is set on by a "TRACE linkid LOG" command a print output file is opened by a call to AXS, and each subsequent I/O execution is formatted and recorded in the print file. The print file is scheduled for printer processing when the file is closed by a call to AXS from the command processor on execution of a "TRACE linkid NOLOG" command.

VMB Data Handling Functions

A. GETBLOCK - This routine is called by the line management routines to prepare data blocks for transmission to the remote system. The data blocks may be initial null pad (SYNCH) blocks, CMD/MSG element (MSGHD) blocks, or spool data blocks.

Upon entry, control is passed to GETPAD if the block to be constructed is the first to be transmitted following initialization or line connection. In this case, a null data block is constructed with a SYNCH header, and control is passed to GETSETUP.

If no SYNCH block is to be generated, a call is made to MSGTRANS if at least one CMD or MSG element is queued for transmission. When the MSGHD block has been built by MSGTRANS, control is passed to GETSETUP.

If none of the above conditions exist, an attempt is made to build a spool data block. If no input spool file is open, control is passed to GETFILE. Otherwise, a call is made to MAKEBLOK to build a spool data block. If an end-of-file condition on the input spool file is encountered, message DMTVMB147I is issued, the old file is purged, and control is passed to GETFILE. When a spool data block is successfully built by MAKEBLOK, control is passed to GETGOT.

At GETFILE, the drain and hold request status are tested. If drain is set, no block is built and control is returned to the caller. If hold is requested, message DMTVMB611I is issued, the link is placed in hold status, and control is returned to the caller with no block built. If an input spool file is potentially available, a call is made to AXSGET to attempt to open an input file. If successful, control is passed to GETGOT; otherwise, control is returned to the caller with no block built.

At GETGOT, a spool data block has been built and is ready for transmission in GETBUFF. Pending commands are tested for validity and applicability, and rejected with diagnostic messages if invalid. Valid pending commands are executed with confirmation messages. If a new file has been opened, messages DMTVMB146I or DMTVMB148I are issued, as appropriate, and control is passed to GETSETUP.

At GETSETUP, the generated block length is determined and is stored

in the block write CCW (WRITDATA) and in the block header. The block serial number is generated and stored in the block header, and control is returned to the calling line management routine.

B. PUTBLOCK - This routine is called by the line management routines to process data blocks received from the remote system. Data blocks with serial numbers matching the previously-received serial number are discarded.

The received data block is processed by decompressing each record at PUTNEXT, and passing control to PUTOUT for individual record processing. When processing has been completed for a record, control is returned to PUTNEXT, which decompresses the next record in the block and passes control to PUTOUT. When no records remain in the block, control is passed to PUTDONE, which checks for end-of-file on spool data output. If end-of-file is detected, message DMTVMB145I is issued, the file is closed by a request to AXS, and all output file status is reset. Finally, control is returned to the calling line management routine.

At PUTOUT, the record to be processed is tested to determine if it is a link command element or a CMD/MSG element. For a RESTART link command element, a flag is set to cause GETBLOCK to restart its active input spool file from the beginning when such a file is present, and the next record is processed. For a PURGE link command element, the active output file (if any) is closed and purged by a call to AXS. For a TAGB control record, the active output spool file (if any) is closed and purged by a request to AXS, the new output tag status is updated and reset according to the contents of the tag record, and the next record is processed. For a CMD or MSG element, MSGRECV is called to pass the element to REX for further processing, and the next record is processed. All other records are interpreted as spool output records, and control is passed to PUTOPEN.

At PUTOPEN, an output spool file is opened by a request to AXS and message DMTVMB144I is issued if no output file was already open and if the link is not in drain status. If the link is in drain status and an input file is still being processed, a WABT response is set for transmission to the remote system. If the link is in drain status and no other file is being processed, control is passed to VMREOT to terminate communication and line driver processing. Control is passed to PUTWRITE when a spool output file is open.

At PUTWRITE, the decompressed record is written into the output spool file using the CCW command code supplied with the record. NOP records (command code X'03') are executed as normal writes, because they may represent transparent control information which is to be embedded in the spool file and preserved. If the write operation is successful, the next record is processed. Otherwise, a diagnostic message is issued and line driver processing is terminated by a call to VMRTILT.

C. MSGRECV - This routine is called by PUTBLOCK to process CMD and MSG elements as they are received. Each such record is given to REX as a request. REX, in turn, executes commands and issues messages contained in elements addressed to the local location, and forwards other elements to line drivers for transmission to remote systems. When the request has been accepted by REX, control is returned to the caller.

D. MSG - This routine is called throughout VMB, normally from within

the expansion of the MSG macro, to format and issue message requests to REX. On each call, a message request element is built using the message number and substitution parameters supplied by the caller. The request element is passed to REX, which builds and distributes the requested message. When REX has completed processing the request, control is returned to the caller.

E. AXSGET - This routine is called by GETBLOCK to prepare an input spool file for reading and transmission. It starts by issuing an OPEN INPUT request to AXS. If no input spool file is available, AXS returns an error indication and control is immediately returned to the caller, reflecting the error. If a file is successfully opened, the new input file status is set, a transmission block is built by a call to MAKEBLOK, and control is returned to the caller indicating successful completion.

F. AXSPURGE - This routine is called by GETBLOCK to terminate processing of an active input spool file. The file may be deleted or saved (reenqueued) for future processing, depending on the request set by the caller. A CLOSE request is built and passed to AXS for execution. When complete, VMB's active input spool file status is reset, and control is returned to the caller.

G. MSGTRANS - This routine is called by the line management routines to build a data block, containing CMD and MSG elements, for transmission. GMSGREQ in DMTCOM is called repeatedly to dequeue and retrieve elements stacked by AXSALERT. Each record retrieved is formatted into a NOP data record, compressed by a call to PACK, and entered into the data block buffer (GETBUFF). When no more elements are available, or when the 824-byte data block length limit is reached, the block is terminated, and is returned to the caller with a MSGHD header code. When no elements are available, an error indication is returned to the caller.

H. MAKEBLOK - This routine is called by the line management routine to build a data block, containing spool data records, for transmission. Page buffer spool data blocks are read from the open input spool file by a hypervisor call (DIAGNOSE X'14', subcode 0) to CP. Each spool page buffer is decomposed record by record. Each record is compressed by a call to PACK, and entered into the data block buffer (GETBUFF). When a data block has reached the 824-byte length limit, or when the spool data input has been exhausted, the block is terminated and control is returned to the caller. When input data records remain on completion of data block construction, the records and deblocking pointers are saved. The next call to MAKEBLOK resumes with the next sequential record.

I. PACK - This routine is called by MSGTRANS and MAKEBLOK to compress a line of data into the VMB packed data format (see DMTVMB Packed Data Block Format). A single line of data is accepted as input, along with its character count. The line is searched for sequences of five or more identical characters by a compare logical character instruction specifying overlapping one-byte offset fields, each four characters long. When such a sequence is located, the previous segment data string (if any) is entered into the caller's output buffer, and the packed sequence is entered with its length and replicated data character. If more input characters remain, the search for identical characters and the construction of the output string continues as described above. When the end of the input string is reached, the output buffer is completed with the

final data segment or packed sequence, the packed output string count is stored in the first byte of the output buffer, and control is returned to the caller.

VMB Processing Control Functions

A. SVMRINIT - This routine is executed only when a VMB line driver is initially started. The start parameter string is inspected for validity, and, if valid, the link passwords are set as specified. Otherwise, a diagnostic message is issued and the passwords are left unspecified. The link table address, the device address of the line in use, and the link ID are located and saved for future use. The prototype tag blocks to be used in OPEN OUTPUT requests to AXS are initialized to their default values. ASYNREQ in MSUP is called to specify AXSALERT as the line driver task's entry point for alert calls from other tasks (REX alerts line drivers for command and CMD/MSG element transfer, and AXS alerts line drivers for notification of file availability). A RESTART link command element is stacked for transmission to the remote system, to cause the remote system to terminate active output file processing and restart active input file processing. Finally, a flag is set to force transmission of an initial pad (SYNCH) block, SVMRINIT's storage page is released, and processing is begun at VMRENABL.

B. CMDPROC - This routine is called by XECUTE when a wait request completes with the posting of the command arrival synchronization lock (CMDECB), following the acceptance of a command alert element from REX. The command code table (CMDTABLE) is searched for a code matching that of the accepted element, and the command's processing routine is entered when a match is made. If no match is found, the command is ignored. The command processing routines are:

ROUTINE	COMMAND IT PROCESSES
SETSTART	START command
SETDRAIN	DRAIN command
SETFREE	FREE command
SETHOLD	HOLD command
SETTRACE	TRACE command
SETBACK	BACKSPAC command
SETFWD	FWDSPACE command
SETFLUSH	FLUSH command

These individual command processors normally issue messages, set flags, and modify other line driver status, depending upon the particular command and the processing status of the line driver. In addition, the SETTRACE routine requests open and close spool output from AXS for line activity log initiation and termination. When processing is complete, control is returned to the caller.

C. AXSALERT - This rcutine is entered asynchronously in masked-off supervisor state from the MSUP ALERT processor (DMTSIG) on an alert call from another task. If the alerting task is AXS, the call is a notification that an input file has become available for the line driver's link. In this case, "file available" status is set, line I/O is terminated by an HDV command when read initial (PREPARE command) is active, and control is returned to MSUP.

If the alerting task is REX, the element code being presented is inspected. For CMD and MSG elements, control is passed to AXSMENQ which calls PMSGREQ in DMTCOM to stack the element for future transmission to the remote system. Otherwise, REX is assumed to be presenting an operator command element. If an operator command is already in progress, the request is rejected and an error indication is returned to REX via MSUP on return. If no command is already in progress, the command element is moved to VMB's command element buffer (CMDRESP), the line I/O is halted if read initial (PREPARE command) is active, the active command presence is flagged, and control is returned to MSUP with a normal completion code for REX.

If the alerting task is other than AXS or REX (which should not happen), the alert call is ignored and control is returned to MSUP.

D. VMRTILT - This routine is entered from various locations in VMB, and its function is to terminate line driver processing. If the reason for termination is a fatal I/O error, entry is made at VMRBADIO. In this case, IOERRPRT is called to issue a diagnostic I/O error and the device in error is marked "inoperative" before termination processing.

At VMRTILT, line I/O logging is terminated if active, and any active line I/O is halted by an HDV instruction. If the line port remains operative, a DISABLE operation is executed to reset the port and disconnect (hang up) dialable telephone data sets. Finally, a task termination request is issued to REX, and VMB enters a permanent wait state until it is deleted from the system as a result of a call to MSUP, which is made by REX during its processing of the terminate request.

VMB I/O Management Functions

A. XECUTE - This routine is called throughout VMB to execute I/O channel programs for all I/O devices. The I/O table is prepared for execution by the caller, including the address of the channel program and the device on which it is to be executed, and the address of the I/O table is passed to XECUTE in register 13.

XECUTE begins by clearing the synchronization lock in the caller's I/O table and calling IOREQ in MSUP, specifying the I/O table, to schedule and execute the requested I/O. The address of the I/O table synchronization lock is stored in a wait list that also includes the command arrival synchronization lock, and WAITREQ in MSUP is called to suspend line driver task dispatching until the I/O completes, or until a command alert element is delivered. When a command alert element arrives, it is processed by a call to CMDPROC, and the wait request is repeated if the I/O request has not completed. When the I/O manager in MSUP signals completion by posting the synchronization lock in the I/O table, control is returned to the caller.

B. IOERRPRT - This routine is called from throughout VMB whenever the standard I/O error message 70 is to be issued. The caller passes the address of the I/O table containing the ending I/O error information to IOERRPRT in register 13. The device address, ending CSW, SIO condition code, and ending CCW are extracted from the caller's I/O table, converted to EBCDIC, and placed in the message request element. The message routing code is set to the VM/370 and

RSCS operator consoles, the MSG routine is called to issue the message request, and control is returned to the caller.

VMC LINE DRIVER FUNCTION DESCRIPTIONS

The VMC line driver (DMTVMC) is for transmitting VM/370 spool files between VM/370 systems over channel-to-channel adapters (CTCAs). DMTVMC passes VM/370 4K spool page buffers to another copy of itself, using a specially designed protocol to optimize utilization of the CTCA without creating heavy I/O activity. The 4K block is read from the VM/370 spool system, transmitted across the CTCA, and then written into the receiving machine's spool system with minimal SIO execution. Like DMTVMB, DMTVMC requires no special operating instructions, and supports the full RSCS command language except for BACKSPAC, HOLD IMMED, and FWDSPACE count.

CTCGO - Main Line Driver Control

This routine is responsible for the main DMTVMC line driver control of the channel-to-channel adapter (CTCA). CTCGO is entered from CTCINIT after line driver initialization is complete. A wait is issued on a list of four synch locks until there is work to be done. The synch locks are:

1. RDYRDY - This synch lock is posted by the asynchronous exit AXSALERT in DMTVMC whenever an alert notification is given from the AXS task indicating than an input file is ready to be transmitted.

2. MSGECB - This synch lock is posted by the asynchronous exit AXSALERT in DMTVMC whenever a command or message to be transmitted across the CTCA has been entered into the link's message stack.

3. CMDECB - This synch lock is posted by the asynchronous exit AXSALERT in DMTVMC whenever a command to be executed by the line driver locally is placed in the CMDRESP buffer after an alert from DMTCMX.

4. ATTNLOCK - This synch lock is posted by the asynchronous exit CTCATTN whenever an attention interrupt is received on the CTCA. This indicates that the other side of the CTCA is requesting that communications be established.

When one or more of these synch locks is posted, I/O activity on the CTCA is initiated to transmit or receive data. Calls are made to MSGTRANS and GETBLOCK to block data for transmission and to MSGRECV and PUTBLOCK to process received data blocks. When there is no more work to do, a wait on the synch lock list is again issued.

Calls are made to IINEIO to initiate an I/O operation on the CTCA. When an I/O request for a read or a write is made to DMTIOM, a timer request is made to DMTREX to post the TIMELOCK synch lock 15 seconds later. A wait is then issued on both the device synch lock and the time synch lock. If the other end does not complete the I/O operation before the time interval of 15 seconds expires, the I/O operation is terminated through an HDV instruction to the CTCA. This operation is done to prevent the read/write operation from inhibiting read channel activity if the remote system has gone down.

GETBLOCK - Input File Formatting

This routine is entered from the main processing routine in DMTVMC (CTCGO) whenever the File Available synch lock is posted, or after an input file has been processed. A call is made, if required, to AXSGET to obtain a new spool file to be transmitted. If a new file is not obtained, return is made to CTCGO. If a new file is obtained, message DMTVMC146I is issued, and return is made to CTCGO with a full block indicated. Successive calls to GETBLOCK will return 4K spool page buffers to be transmitted until an end of file condition. When end-of-file occurs on the input spool device, an end-of-file record is returned to be transmitted, to close the output spool file on the receiving system.

MSGRECV - Command or Message Receipt

This routine is entered from CTCGO when a block that contains a command or message is received. This record is converted into a routing request element and passed through GIVE/TAKE to DMTRGX for processing.

MSGTRANS - Command or Message Transmittal

This routine is entered from CTCGO when the MSGQUED flag and the MSGECB synch lock have been posted by the asynchronous exit, AXSALERT. The message or command is placed in a message transmission buffer for transmission across the CTCA. The MSGQUED flag is not reset, indicating that there are more messages or commands to be transmitted. When a non-zero return code is obtained from the call to GMSGREG, the MSGQUED flag is reset and control is returned to CTCGO.

PUTBLOCK

This routine is entered by CTCGO whenever a spool data block is received across the CTCA. When a tag record is encountered, a new spool output device is obtained via a GIVE/TAKE call to DMTAXS. If a file was previously open, a restart is assumed, and the file is closed and purged. If the file being received is destined for the local location, a header line indicating the file origin is placed in the output spool file. Each successive data block has the format of a VM/370 4K spool page buffer. This buffer is relocated in the same manner as performed by module DMKRSP in CP. Once the data has been relocated, one virtual SIO is issued to write the entire buffer into the VM/370 spool system. When an end-of-file record is received, the output spool file is closed via a GIVE/TAKE call to DMTAXS.

CMDPROC

(See CMDPROC under VMB Processing Control Functions.)

AXSALERT

(See AXSALERT under VMB Processing Control Functions.)

TRTRAN, TRERR, TRTIMOT

The trace sum routines are called by the various line management routines to count the accumulated number of successful transmissions, line errors, and timeouts, respectively. These counts are constantly maintained in the line driver's link table, where they may be interrogated by a "QUERY linkid SUM" command. When sum tracing is active, message DMTxxx149I is issued each time any of the counts reaches a threshold value, and all counts are reset to zero.

KLOGIT

This routine is entered after every line I/O transaction. When log trace is set on by a "TRACE linkid LOG" command, a print output file is opened by a call to AXS, and each subsequent I/O execution is formatted and recorded in the print file. The print file is scheduled for printer processing when the file is closed by a call to AXS from the command processor on execution of a "TRACE linkid NOLOG" command.

NJI LINE DRIVER FUNCTION DESCRIPTIONS

The NJI line driver (DMTNJI) communicates with non VM/370 NJI or NJE systems. The line driver consists of three preloaded modules (see Preloader, Appendix B):

DMTNCM - acts as main control of DMTNJI, manages the communications adapter and interfaces to the VM/370 spool system through the DMTAXS task.

DMTNHD - processes NJI/NJE header records for jobs, output, commands, and messages.

DMTNIT - handles line driver initialization. This module verifies parameters, obtains storage for teleprocessing buffers and unit record tanks, and constructs the initial SIGNON record. The storage for this module is freed upon return to DMTNCM.

DMTNJI conforms to the protocol defined by the Network Job Entry facility for JES2. This protocol is an extension to the remote job entry protocol used by the DMTSML line driver.

This protocol extends the definition of MULTI-LEAVING for symmetrical communication between host systems. For more information, refer to Appendix A. The data flow diagram for DMTSML is extended for DMTNJI (Figure 2-23).

The NJI line driver supports communication over bsc lines with MULTI-LEAVING and over channel-to-channel adapters.

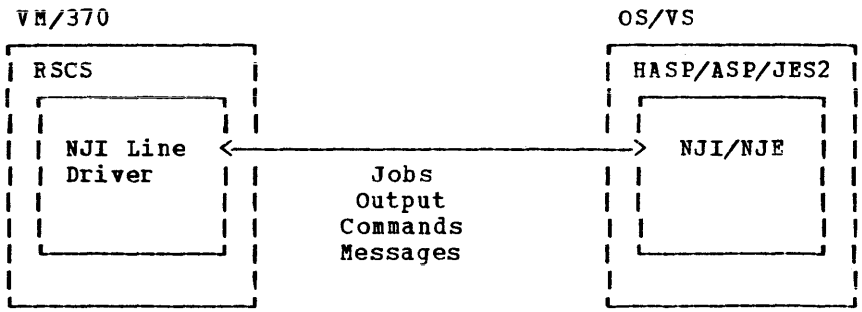


Figure 2-23. NJI Link Data Flow

The DMTNCM module comprises the following basic routines:

- A function selector that dispatches one of the processors or other routines in NCM when a request for services is received.
- Processors, that execute the main functions required by the network interface.
- An input/output routine that accepts and transmits data on the communication adapter.
- Buffer blocking and deblocking routines.

NCM Function Selector Routine: $START

The $START routine is entered when NCM is required (by either a remote system or a virtual machine) to perform a function. This routine selects a function to execute by using a commutator table, a list of synch locks, and task control tables.

The NCM commutator table is a branch table consisting of unconditional branch and no-operation instructions. The targets of the branch instructions are the seven processor routines, the I/O handling routine, and the buffer handling routines. When the service of a routine is not required, the commutator table entry for that routine is made a NOP instruction. When the function of the routine is required, the NOP instruction in the commutator table entry for that routine is replaced with an unconditional Branch instruction, thereby opening a gate in the commutator table.

The $START routine cycles through the commutator table, falling through any NOP instructions and taking any branches. Control is passed in this way to any routine whose gate in the commutator table is open.

When the routine completes the function requested, it closes that function's gate in the commutator table by replacing the unconditional branch instruction with a NOP instruction. $START continues cycling through the commutator table taking any open branches.

When the bottom of the commutator table is reached, $START tests a series of synch locks to see if any have been posted, signifying a request for an NCM function. If any synch lock is posted, $START opens the commutator table gate for the requested processor and goes to the top of the commutator table to start cycling through it again.

If the bottom of the commutator table is reached and there are no posted synch locks, NCM discontinues processing by issuing a wait request via a call to the supervisor module DMTWAT, waiting on a list of the synch locks. When any of the synch locks is posted, $START receives control, opens the appropriate gate, and starts cycling through the commutator table.

NCM Processors

Each processor performs a specific function necessary for network job entry. Figure 2-24 summarizes the processors in DMTNCM.

Each processor has a task control table (TCT) associated with it that defines data required by the processor. Within the TCT is a branch instruction to the appropriate processor. The commutator addresses these TCT branch instructions instead of branching directly to the processors.

PROCESSOR	FUNCTION
$CRTN1	Processes the following MULTI-LEAVING control records: Permission To Transmit Request To Transmit Negative Open Signon Control Records
$PRTN1	Processes output files from a remote system. Interfaces with DMTNHD to process network header records.
$URTN1	Processes job files from a remote system. Interfaces with DMTNHD to process network header records.
$WRTN1	Processes commands and messages from a remote system. Interfaces with DMTNHD to process network header records.
$RRTN1	Prepares records read from the VM/370 spool system for transmission. Interfaces with DMTNHD to build network header records.
CMDPROC	Executes local commands passed by DMTCMX.
MSGPROC	Prepares commands and messages for transmission. Interfaces with DMTNHD to build network header.

Figure 2-24. NCM Function Processors

NCM Line I/O Handler Routine: COMSUP

COMSUP controls all I/O activity on the communications adapter for the NJI line driver. It handles both BSC line and channel-to-channel communication. This routine receives data from the adapter and passes the data to the deblocker routine ($TPGET). COMSUP sends data (which has been blocked by the blocker routine, $TPPUT) to a remote system. COMSUP also acknowledges receipt of data over a BSC line using the standard BSC control characters.

Block and Deblock NCM Teleprocessing Buffers: $TPPUT and $TPGET

Data received over the communications adapter is placed in a teleprocessing (TP) buffer. The size of TP buffers is specified by a START command parameter and can be up to 1017 bytes.

Data contained in TP buffers is deblocked into tanks, which are unit buffers of a specific size used to deblock the larger TP buffers. There are 15 tanks; these are allocated as they are needed by processors. The size of tanks is determined by MULTI-LEAVING control bytes.

When an NCM function has been requested, the data must be either blocked for transmission (if it is data for a remote system) or deblocked for processing (if it has been received from a remote system).

$TPGET receives data from a communications adapter (via the COMSUP routine) and allocates tanks to output processors as they are needed.

$TPPUT receives tanks from input processors, blocks the data in these tanks into TP buffers, and gives control to COMSUP to transmit the buffers over the adapter.

Network Header Processor: DMTNHD

DMTNHD contains routines to process or construct the various network header and trailer records used by the NJI/NJE protocol. The network header records are read by another non VM/370 NJI system that uses them to reconstruct information about the files sent from RSCS. There are two major types of network header records: job headers and data set headers. DSECTs describing these header and trailer records are shown in Section 5. Other NJE/NJI systems also make use of the same mapping shown in these DSECTs. The header and trailer routines in DMTNHD are entered from the processors in DMTNCM whenever network header or trailer processing is required.

Command and Message Processing

Two routines are used to process network commands and messages. DMTNHDMI is entered from $WRTN1 to interpret the header record whenever a command or message is received from a remote system. From the information in this header record a routing request element is built and then passed by DMTNCM to DMTRGX for processing. If a global network command is recognized, it is translated into the appropriate RSCS command, if the command destination specifies the local RSCS.

DMTNHDMO is entered whenever a network command or message to be transmitted is processed by MSGPROC in DMTNCM. The network header is constructed from information contained in the routing alert element.

File Header Record Output Processing

Entry point DMTNHDHO is entered from $PRTN1 and $URTN1 each time a network header record is received by the output or job file processors in DMTNCM. A subroutine is then entered when a valid NJI/NJE header record is found.

Routine HOJOB is entered when a job header record is found. The header record is saved for later processing, various default fields are created in the output log slot, and message DMTNHD917I is issued.

HODATSET is entered each time a data set header is encountered. The fields in the default tag slot are updated from the fields in the data set header record. The logical output device table is then searched via a call to DVASSIGN. This routine will assign this data set to a device for processing, by comparing the attributes of the new data set with ones already existing. If an open data set is not found, a new device is obtained via a call to OPENADEV. This routine obtains, through a call to DMTAXS, a new spool device. Once a new device is obtained, routine HDROUT is called to output the job header as a segmented NOP spool record.

When the job trailer record is encountered, subroutine HOJOBTRL is entered. All open data sets are closed after the job trailer record is written to the spool system.

File Header Record Input Processing

Routines DMTNHDJH, DMTNHDDH, and DMTNHDJT are entered from processor $RRTN1 in DMTNCM when a file to be entered into the NJI line driver does not already contain NJI/NJE header records. DMTNHDJH and DMTNHDJT create the job header and trailer records from the information in the input tag slot. The data set header creation routine DMTNHDDH creates the data set header record from the input tag slot along with data from calls to the TAGSCAN routine to scan the user's tag data record for NJI/NJE parameters. Because non VM/370 systems generally distingush between job files and output files, RSCS must make a distinction between such files before they are sent to another NJI/NJE system. The terminal user specifies whether he wants his files to be JOB files or output files in the tag record before he sends them to RSCS to be sent to another system. DMTNHD then scans this tag record to determine the type of file to be sent and obtains other information about the file that it puts in the NJI/NJE header records.

DMTNJI Initialization Module: DMTNIT

This module is entered from DMTNCM during line driver initialization. It performs the following functions:

1. Scans the supplied parameter string and create a network SIGNON record from the information obtained from the scan.

2. Sets task alert asynchronous exit.

3. Obtains storage and builds the teleprocessing buffer queue and unit record tank queue.

When control is returned to DMTNCM, the page containing DMTNIT is freed via a call to FPAGEREQ in DMTCOM.

Section 3: Program Organization

Modules and Subroutines

Figure 3-1 lists the functions of all RSCS subroutines in order by module name.

Module Name	Entry Pt /Routine	Function
DMTAKE	DMTAKEEP	Contains the supervisor service that supplies task programs with the receiver interface to GIVE requests issued by other tasks. A single call causes DMTAKE to first respond to the previously supplied GIVE request and then supply a new GIVE request to the task for its processing.
DMTASK	DMTASKEP	A service routine that creates new tasks and deletes existing tasks executed by the MSUP dispatcher. The entry to DMTASK is via a BALR instruction from task programming. Any entry into DMTASK causes the calling task's execution to be suspended through the freeze SVC function.
DMTASY	DMTASYEP	A supervisor service module that starts and ends asynchronous exit requests for task programs. This routine handles asynchronous exit requests for asynchronous exit requests for I/O interruptions, and alert exit requests.
DMTAXA	DMTAXAAC	NOP entry -- No accounting record is cut when a file is accepted.
	DMTAXASE	Cuts a send accounting record when all copies of an input file have been sent.
	DMTAXAPU	NOP entry -- No accounting record is cut when a file is purged by purge command.
	DMTAXARE	Cuts a receive accounting record when an output file is written to the spool system.
	DMTAXATA	NOP entry -- user tag priority of a file is left unchanged.
DMTAXM	DMTAXMEP	Controls the interface of the line drivers to the VM/370 spool file system, enqueues files for transmission, and processes commands that manipulate spool files.
	AXSASYIO	Start of asynchronous exit routine; signals arrival of a request for asynchronous exit.
	AXSINIT	Initializes the AXS task.
	AXSCYCLE	Looks for work to do by examining the synch locks associated with the AXS task.

Figure 3-1. RSCS Modules and Their Subroutines (Part 1 of 13)

Module Name	Entry Pt /Routine	Function
	REQXEQ	Scans the request table for a match and branches to the appropriate subroutine, depending on the request code.
	CMDPROC	Executes AXS commands from the command buffer passed on by an alert exit from DMTREX.
	OPENIN	Starts spool file processing.
	CLOSEOUT	Ends processing for output files.
	CLOSIN	Terminate spool file processing.
	MSG	Sets the MSG request element. The MSG request element is passed via GIVE/TAKE to the message manager, DMTMGX. The code associated with entry points in this module format the MSG element variable areas in various ways and exit finally to MSG.
	HEXGET	Converts and validates a hex string.
	DECGET	Converts and validates a decimal string.
	DECPUT	Converts a hex fullword to decimal and generates an EBCDIC representation of it, suppresses leading zeros to a minimum count, which is optionally supplied by the caller.
	TODS370	Converts EBCDIC to the System/370 TOD value.
	TODEBCD	Converts System/370 TOD to EBCDIC date and time.
	GSUCCESS	Gets inactive successor spool file.
	ACCEPT	Inspects newly arrived files.
	UNPEND	Brings in a link's pending tags.
	GETROUTE	Gets a routing table entry.
	GETLINK	Gets link table entry.
	GETSLOT	Gets a free tag queue element.
	FREESLOT	Returns a tag queue element.
	TAGGEN	Builds a file tag from hypervisor information.
	TAGPLACE	Sets a file tag into a link queue immediately before the first tag of numerically higher priority (lower processing priority).
	REORDER	Reorders file queue after System reconfiguration.
	FILSELEC	Selects a file to be read from a link queue.
	TAGFIND	Locates a file with spoolid matching the one supplied by the caller, within the internal file tag queues.
	TAGCLOSE	Dequeues an active file tag, closes and re-enqueues input files, closes and purges output files.
	DEFINE	Gets a virtual spool device.
	DETACH	Undefines a virtual spool device.
	VCHANGE	Changes VM/370 file attributes.
	VCLOSE	Issues the VM/370 CLOSE command for a device.
	VPURGE	Purges an inactive reader file from the VM/370 spool.
	VTRANSFR	Transfers an incorrectly addressed file back to its original user.
	VSPOOL	Sets VM/370 virtual spool device options.
	VTAGD	Sets a VM/370 tag for a virtual spool device.
	VTAGMSG	Sets a VM/370 tag for a virtual spool device spooled to a user.
	VTAGF	Sets a VM/370 tag for an inactive spool file.

Figure 3-1. RSCS Modules and Their Subroutines (Part 2 of 13)

Module Name	Entry Pt /Routine	Function
DMTCMX	DMTCMXEP	This module is part of the REX system control task; it is called in several places in DMTREX, (the main REX control routine). DMTCMX accepts an EBCDIC string and executes the RSCS command that the string represents.
	CMXHIT	Calls the necessary individual command processing routine.
	CMXALERT	Passes a command element to another task via the alert task-to-task communications interface.
	CMXREORD	To pass to DMTAXS a REORD alert element.
	CMXULOOP	Check for looping responses to user.
	KEYWDGET	Decodes the next keyword on the input command line.
	LTABGET	Finds the link table entry implied by the first keyword in the command line described by the calling routine's register parameters.
	RTABGET	Find the root table entry implied by the first keyword in the command line described by the calling routine's register parameters.
	HEXGET	Converts and validates a hex string.
	DECPUT	Converts a hex fullword to decimal and generates an EBCDIC representation of it. It suppresses leading zeros to a minimum count, which is optionally supplied by the calling routine.
	FILGET	Locates a file, within the internal file tag queues with a spoolid matching that supplied by the calling routine.
	TODEBCD	Converts System/370 TOD to EBCDIC date and time.
	PARMGET	Scans an EBCDIC line and frames the next parameter on the line.
	FINDTAGQ	Finds a file tag with the same destination as the ROUTDEST in the routing table.
DMTCOM	DMTCOMEP	Contains various reentrant routines used by RSCS tasks.
	GETLINK	Scans the link table chain and returns a link table address.
	GETROUTE	Scans the link table chain and the routing table chain to select the next link for transmission.
	GETPAGE	Gets a free page of virtual storage.
	FREEPAGE	Returns a page of virtual storage.
	MFI	Stacks message elements in a LIFO stack for later processing. If no room is available in the current page, a new page is fetched if at least five free pages remain. If five free pages are not remaining, an error condition is returned. All tasks except REX are allowed only three pages of storage to stack messages.
	MFO	Unstacks message elements from the message queue for this task. If none are queued, an error condition is returned.
	TODEBCD	Converts System/370 TOD to EBCDIC date and time.
	TODS370	Converts EBCDIC to System/370 TOD.
	RCMSOPEN	Initializes reading of a CMS file.
	RCMSGET	Gets the next CMS file item.
	GETSUPAG	Allocates a page of virtual storage for supervisor use.

Figure 3-1. RSCS Modules and Their Subroutines (Part 3 of 13)

Module Name	Entry Pt /Routine	Function
DMTCRE	DMTCREEP	Creates new tasks under MSUP LOAD + QRQ.
DMTDSP	DMTDSPEP	This module is the MSUP dispatcher; it is entered when an exit occurs from supervisor functions that were entered following an interruption or that issued the freeze SVC function. DMTDSP must be entered with all PSW masks off (except for the machine check mask).
DMTEXT	DMTEXTEP	This module is the MSUP external interruption handler; it receives control directly on an external interrupt and saves the status of the executing task if one was interrupted.
DMTGIV	DMTGIVEP	This is a supervisor service routine that enqueues GIVE requests from tasks to be delivered to other tasks by DMTAKE.
DMTINI	DMTINI	Receives control after initial loading of RSCS, and performs general initialization functions common to all parts of RSCS. DMTINI writes a copy of the initial load to DASD, according to operator instructions, when RSCS is initial program loaded from the generation IPL deck. When IPL disk writing is complete, a masked off wait state PSW is loaded. Following IPL from RSCS system residence DASD, DMTINI finishes reading the saved RSCS load. When IPL disk reading or writing is complete, DMTINI passes control to DMTMIN.
DMTIOM	DMTIOMEP	This module contains both the MSUP I/O interrupt handler and the task I/O service routine. The I/O service provided by DMTIOM to the task programs includes sequential subchannel scheduling, channel program execution, automatic sense execution on unit check when requested, return of all pertinent information regarding the execution of the channel program, and notification via a POST upon completion of the channel program.
DMTIRX	DMTIRXEP	This module performs all non-MSUP oriented RSCS initialization.
	GENVNET	Builds RSCS system tables from directory.
	GETPARM	Locates next parameter on input record, and performs preliminary validity checking.
	PARMGET	Scans a character string left to right, and frames the first parameter (a substring of non-delimiters enclosed by delimiters or string boundaries on the left and right).
	DECPUT	Converts a hex fullword to decimal and generates an EBCDIC representation of it. It suppresses leading zeros to a minimum count, which is optionally supplied by the calling routine.

Figure 3-1. RSCS Modules and Their Subroutines (Part 4 of 13)

Module Name	Entry Pt /Routine	Function
	EBCHEX	Validates and converts EBCDIC hexadecimal input numbers to binary.
	EBCDEC	Validates and converts EBCDIC decimal input numbers to binary.
	DIRECT	Sequentially reads the directory file (RSCS DIRECT) and returns a data line or error condition to the caller.
	TYPE	Generates a numbered operator message by editing the message according to CP EMSG setting, calling CONW to write the message on the console, and returning to the caller.
	CONW	Writes a line to the RSCS operator console and provides console support until normal DMTREX console manager begins processing.
	MSG	Stacks a message for later output.
DMTLAX	DMTLAXEP	This routine is the line allocation task for RSCS. Most of this routine functions as an asynchronous exit being alerted by DMTREX.
DMTMGX	DMTMGXEP	Takes a message request buffer and builds the message from the information in that buffer and the message definition found in DMTMSG.
DMTMIN	DMTMINEP	Performs basic MSUP initialization operations, deletes itself, and issues the first call to the MSUP dispatcher to begin normal operations.
DMTMSG	DMTMSGEP	Contains a list of error messages to be used externally by DMTMGX. This module contains no executable code.
DMTNCM	DMTNCMEP	This line driver communicates with NJI/NJE systems on BSC lines or channel-to-channel adapters.
	NCMINIT	Initializes various parameters needed by DMTNCM. Saves its link table address, initializes output tags, and calls DMTNIT to complete initialization.
	ISIO	Performs the enable sequence on the communications line, analyzes the response received, and when correct, writes the "line connected" message.
	ASYNEXIT	This is the alert exit entered by DMTSIG. Two tasks may alert this line driver: DMTREX when a command has been entered for the DMTNCM line driver to process, or DMTAXS to asynchronously notify DMTNCM a file has arrived for transmission.
	$START	This is the supervisor routine for DMTNCM. The commutator will cycle looking for a routine to enter until all commutator entries are closed; then it will wait on a synch lock list to be posted.
	$CRTN1	Dequeues a tank from its tank queue and performs the action requested by the control record in the dequeued tank.

Figure 3-1. RSCS Modules and Their Subroutines (Part 5 of 13)

Module Name	Entry Pt /Routine	Function
	$PRIN1	Dequeues a tank from its tank queue, obtains a new output spool device if needed from DMTAXS, and outputs the tank to a virtual printer.
	$URTN1	Dequeues a tank from its tank queue, obtains a new output spool device if needed from DMTAXS, and outputs the tank to a virtual punch.
	$RRTN1	Inputs files from the VM/370 spool system, deblocks them into individual records, and issues a call to $PUT to block the record into a transmission buffer.
	AXSGET	This routine is the interface to DMTAXS for getting files to transmit, and it purges those files when transmission is complete.
	VMDEBLOK	Deblocks records from the VM/370 page spool buffers. It returns the deblocked record in the RCTTDATA buffer.
	$WRTN1	Processes received commands and messages and calls DMTNHD for processing. The records are dequeued from the console TCT.
	CMDPROC	Executes commands passed to it in the CMDRESP buffer after an alert from DMTREX indicating a command has been entered.
	MSGPROC	This routine is entered when the MSGECB is posted by this task's asynchronous exit, indicating messages are queued for this task. These messages are unstacked from the message queue by repeated calls to GMSGREQ and queued for transmission.
	MSG	Prepares and sends requests to the specialized task REX, in order to write messages on the operator's console.
	$TPPUT	Takes a line and packs it into a telecommunication buffer. When a buffer is filled, it is queued onto $OUTBUF for processing by COMSUP.
	$TPGET	Deblocks received telecommunications buffers into tanks and queues the tank onto the appropriate processor's TCTTANK queue.
	COMSUP	Performs all I/O on the communications line. It dequeues TP buffers from $OUTBUF for transmission and queues received TP buffers onto the $INBUF queue for deblocking by $TPGET.
	CERROR	Analyzes all errors on the communication line and takes corrective action depending on the type of error.
DMTNHD	DMTNHDMI	Edits network messages.
	DMTNHDMO	Network command processor.
	DMTNHDHO	Network output header processor.
	DVASSIGN	Assigns a data set header to an output device.
	OPENADEV	Opens a spool output device with characteristics defined in the output tag.
	HDROUT	Outputs NJE header record in 80-byte segments into the VM/370 spool system.
	HDRBUILD	Builds a job header record from multiple segments.
	DMTNHDJH	Job header creation routine.
	DMTNHDDH	Network data set header creation routine.
	DMTNHDJT	Network job trailer header creation routine.

Figure 3-1. RSCS Modules and Their Subroutines (Part 6 of 13)

Module Name	Entry Pt /Routine	Function
	TAGSCAN	Scans user tag data for NJE forms keywords.
	MSG	Prepares and sends requests to REX for issuance of messages.
	PARMGET	Input line scanning subroutine.
	EBCHEX	Validates and converts EBCDIC hexadecimal input numbers to binary.
	EBCDEC	Validates and converts EBCDIC decimal input numbers to binary.
DMTNIT	DMTNITEP	This module is the initialization module for the DMTNJI line driver. From the parameters passed to it at line driver initialization, it builds buffers and constructs initial SIGNON records.
	NITINIT	Initializes the various parameters needed by DMTNIT. Saves its link table address, initilizes output tags, and constructs the SIGNON card from information on the PARM field of the START command.
	IBLDBUFS	Builds TP and unit buffer for the DMTNJI line driver.
	PARMGET	Scans a character string left to right, and frames the first parameter (a substring of non-delimiters enclosed by delimiters or string boundaries on the left and right.)
	EBCHEX	Validates and converts EBCDIC hexadecimal input numbers to binary.
	EBCDEC	Validates and converts EBCDIC decimal input numbers to binary.
	MSG	Prepares and sends requests to REX for issuance of messages.
DMTNPT	DMTNPTEP	This module is the line driver that supports the 2770, 2780, 3770, and 3780 compatible nonprogrammable terminals.
	NPTGET	Maintains a cyclic control of the DMTNPT task on both sending and receiving operations.
	SENDEOT	Sends the BSC end-of-transmission character (EOT) on the line to the remote terminal.
	BUFFINIT	Initializes the line output buffer with the correct BSC character set, depending on the type of output file and features available at the terminal.
	XECUTE	Requests the supervisor to execute I/O operations. After starting the I/O operations, XECUTE waits for either a command to be entered or the completion of the requested I/O operation.
	LINEIO	Executes (by calling XECUTE) I/O operations on the BSC line and checks the results. LINEIO then flags any errors and normally returns to the caller.
	GETBLOCK	Prepares the line output buffer to be transmitted to the remote terminal.
	GETVRFY	Analyzes the response obtained from each buffer transmission and takes the appropriate action.
	PUTBLOCK	Deblocks received TP buffers and writes the deblocked record to the VM/370 spool system.

Figure 3-1. RSCS Modules and Their Subroutines (Part 7 of 13)

Module Name	Entry Pt /Routine	Function
	PUTVRFY	Verifies the content of each received TP buffer, and constructs an appropriate reply if the buffer is in error.
	COMMANDS	Passes commands received from the remote card reader to the RSCS command processor.
	CMDPROC	Executes commands passed to it in the CMDRESP buffer after an alert from DMTREX indicates that a command has been entered.
	MSGPROC	Unstacks messages from the task message queue and transmits them to the remote terminal printer.
	MSG	Prepares and sends requests to the specialized task REX to write console messages.
	HEADPREP	Provides, record by record, the separator and header for print files and the header card for punch files.
	MAKEBLOC	Saves the caller's registers for a call to VMSB2CP; upon return from VMSB2CP, it sets the return code and returns to the original caller.
	VMSB2CP	Deblocks the VM/370 spool page buffers into an unpacked buffer (PACKBLK).
	AXSGET	Requests AXS to open, close, and delete the spool files that the NPT task is processing.
	TODEBCD	Converts System/370 TOD to EBCDIC date and time.
	PARMGET	Scans character strings to find delimiters.
	NPTINIT	Initialization routine for NPT.
	NPTLINK	NPT sign-on routine.
	NPTERROR	Writes the terminal I/O error message and terminates the task.
	NPTTERM	Terminates the NPT task.
DMTPOW	DMTPOWEP	Functions as a remote VSE/POWER system using the VSE/POWER MULTI-LEAVING transmission protocol.
	POWINIT	This routine initializes the various parameters needed by DMTPOW. It saves the link table address and initialized output tags.
	ISIO	Performs the enable sequence on the communications line and analyzes the response received; if the response is correct, it writes the "line connected" message.
	ASYNEXIT	This is the alert exit entered by DMTSIG. Two tasks may alert this line driver: • DMTREX - When a command has been entered for processing by the DMTPOW line driver. • DMTAXS - When DMTAXS must asynchronously notify DMTPOW that a file has arrived for transmission.
	$START	This is the supervisor routine for DMTPOW. The commutator cycles while looking for a routine to enter until all commutator entries are closed. It then waits for a synch lock list to be posted.
	$CRTN1	Dequeues tanks from its tank queue and performs the action requested by the control record in the dequeued tank.

Figure 3-1. RSCS Modules and Their Subroutines (Part 8 of 13)

Module Name	Entry Pt /Routine	Function
	$PRTN1	Dequeues tanks from its tank queue, obtains a new output spool device, if needed, from DMTAXS, and sends the tank to a virtual printer.
	$URTN1	Dequeues tanks from its tank queue, obtains a new output spool device, if needed, from DMTAXS, and sends the tank to a virtual punch.
	$RRTN1	Reads files from the VM/370 spool file system, deblocks the files into 132-byte records, and issues a call to $TPPUT (via $PUT) to block the record into a transmission buffer.
	AXSGET	This routine is the interface to DMTAXS; it gets files ready to transmit and purges those files when transmission is complete.
	VMDEBLOK	This is the deblock routine for the VM/370 page spool buffers. It returns the deblocked record in the RCTTDTA1 buffer.
	$WRTN1	This routine writes received VSE/POWER commands to the RSCS operator.
	$MRTN1	This routine writes received VSE/POWER messages to the RSCS operator.
	CMDPROC	Executes commands passed to it in the CMDRESP buffer after an alert from DMTREX indicating that a command was entered.
	MSGPROC	Entered when the MSGECB is posted by this task's asynchronous exit indicating messages are in the message queue for this task. These messages are unstacked from the message queue by repeated calls to GMSGREQ and queued for transmission.
	MSG	Prepares and sends message requests to REX.
	PARMGET	Scans lines and tests for delimiters.
	$TPPUT	Takes a line and packs it into a telecommunication buffer. When the buffer is filled, it is queued onto $OUTBUF for processing by COMSUP.
	$TPGET	Deblocks received telecommunications buffers into tanks and queues the tank onto the appropriate processor's TCTTANK queue.
	COMSUP	Processes all I/O on the communications line. It dequeues buffers from $OUTBUF for transmission and queues received buffers onto the $INBUF queue for deblocking by $TPGET.
	CERROR	Analyzes all errors on the communications line and takes corrective action depending on the type of error.

Figure 3-1. RSCS Modules and Their Subroutines (Part 9 of 13)

Module Name	Entry Pt /Routine	Function
DMTPRE	DMTPREEP	RSCS preloader utility program (see Appendix B for details).
DMTPST	DMTPSTEP	This service routine may be called from anywhere in RSCS. DMTPST signals the completion of an event by posting the event's associated synch lock. This routine is entirely reentrant and does not change the state of the running PSW.
DMTQRQ	DMTQRQEP	Manages the MSUP supervisor status queue for other MSUP functions. DMTQRQ is for use within the supervisor and must be entered with all PSW masks off (except machine check).
DMTREX	DMTREXEP	This routine is the controlling supervisor task; DMTREX, DMTCMX, DMTMGX, DMTSYS, DMTCOM, DMTMSG, and DMTCRE make up the REX supervisor task.
	DMTREXIN	Performs the initialization for the DMTREX task.
	REXCYCLE	Monitors a list of synch locks when looking for work for DMTREX to perform.
	REXPCHEX	Processes program checks.
	REXITERM	Entered when RSCS initialization fails. Issues the initialization failure message, dumps the contents of main storage, types any remaining messages, and loads a disabled wait state PSW.
	REQXEQ	Scans the function table and calls either DMTCMX or DMTMGX as appropriate.
	INTCMD	Internal command processor.
	DEACT	Deactivates the link table entry.
	DMTREXEC	Process exec file content.
	DMTREXTR	Executes the terminate function for the FORCE command.
	MSG	Prepares message requests and calls DMTMGX.
	TIMERSET	Supports timer alert requests.
	TERMINAT	Terminates a specified task.
	QUIESCE	Executes as task code for a task in the process of termination. Looks for any outstanding I/O for the terminating task. If any outstanding I/O is found, issues HIO and waits for completion; upon completion it terminates the task.
DMTRGX	DMTRGXEP	Handles the command and message routing request elements.
	RGXCMD	Interfaces a CMD routing request element with DMTCMX.
	RGXMSG	Writes message DMTxxx170I or DMTxxx171I from message routing request element.
	RGXNTHRE	Processes CMD/MSG routing request element for store-and-forward.
	RGXDOIT	Message routing interface.
	RGXMSGER	Processes non-zero return code from DMTMGX.
DMTSIG	DMTSIGEP	Performs a task alert exit for a requesting task.

Figure 3-1. RSCS Modules and Their Subroutines (Part 10 of 13)

Module Name	Entry Pt /Routine	Function
DMTSML	DMTSMLEP	Functions as an RJE workstation into a remote system using the MULTI-LEAVING transmission protocol. It can also function as a host to a remote programmable workstation supporting a System/370, System/3, Model 20, 1130, 2922, or other compatible workstation systems.
	SMLINIT	Initializes various parameters for DMTSML. Saves the link table address, initializes output tags, constructs the SIGNON card from the operand field of the START command.
	ISIO	Performs the enable sequence on the communications line and analyzes the response received; if the response is correct, it writes the "line connected" message.
	ASYNEXIT	This is the alert exit entered by DMTSIG. Two tasks may alert this line driver: • DMTREX - When a command has been entered for processing by the DMTSML line driver. • DMTAXS - When DMTAXS must asynchronously notify DMTSML that a file has arrived for transmission.
	$START	This is the supervisor routine for DMTSML. The commutator cycles while looking for a routine to enter until all commutator entries are closed. It then waits for a synch lock list to be posted.
	$CRTN1	Dequeues tanks from its tank queue and performs the action requested by the control record in the dequeued tank.
	$PRTN1	Dequeues tanks from its tank queue, obtains a new output spool device, if needed, from DMTAXS, and sends the tank to a virtual printer.
	$URTN1	Dequeues tanks from its tank queue, obtains a new output spool device, if needed, from DMTAXS, and sends the tank to a virtual punch.
	$JRTN1	Dequeues tanks from its tank queue, obtains a new output spool device, if needed, from DMTAXS, and sends the tank to a virtual punch.
	$USREXIT	Validates the ID card in the front of decks read in from a remote card reader.
	$RRTN1	Reads files from the VM/370 spool file system, deblocks the files into 132-byte records, and issues a call to $TPPUT (via $PUT) to block the record into a transmission buffer.
	AXSGET	This routine is the interface to DMTAXS; it gets files ready to transmit and purges those files when transmission is complete.
	VMDEBLOK	This is the deblock routine for the VM/370 page spool buffers. It returns the deblocked record in the RCTTDTA1 buffer.
	HEADPREP	Provides, one record after the other, the separator and header for print files and the header card for punch files.
	TODEBCD	Converts System/370 TOD to EBCDIC date and time.
	$WRTN1	In RJE mode, writes received messages to the RSCS operator. In HOST mode, passes commands to DMTREX. These commands or messages are dequeued from console TCT.

Figure 3-1. RSCS Modules and Their Subroutines (Part 11 of 13)

Module Name	Entry Pt /Routine	Function
	CMDPROC	Executes commands passed to it in the CMDRESP buffer after an alert from DMTREX indicating that a command was entered.
	MSGPROC	Entered when the MSGECB is posted by this task's asynchronous exit indicating messages are in the message queue for this task. These messages are unstacked from the message queue by repeated calls to GMSGREQ and queued for transmission.
	MSG	Prepares and sends message requests to REX.
	PARMGET	Scans lines and tests for delimiters.
	$TPPUT	Takes a line and packs it into a telecommunication buffer. When the buffer is filled, it is queued onto $OUTBUF for processing by COMSUP.
	$TPGET	Deblocks received telecommunications buffers into tanks and queues the tank onto the appropriate processor's TCTTANK queue.
	COMSUP	Processes all I/O on the communications line. It dequeues buffers from $OUTBUF for transmission and queues received buffers onto the $INBUF queue for deblocking by $TPGET.
	CERROR	Analyzes all errors on the communications line and takes corrective action depending on the type of error.
DMTSTO	DMTSTOEP	Reserves pages of free storage by calling task programs that free storage pages by clearing the associated map byte to zero in the main storage map.
DMTSVC	DMTSVCEP	This module is the MSUP interrupt handler; it receives control directly when an SVC interrupt occurs.
DMTVEC	DMTVECEP	Describes the fixed address storage utilization for MSUP, beginning at main storage address X'200'. System/370 architecture defines the first 512 bytes of main storage, and MSUP uses this area as defined. This area is not assembled in the DMTVEC module to facilitate initial system loading. This area is initialized by DMTINI at IPL.
DMTVMB	XECUTE	Performs I/O on the supplied I/O block.
	LINEIO	Performs I/O and analysis.
	GETBLOCK	Processes input files for transmission.
	PUTBLOCK	Processes received data buffer outputting to spool system.
	MSGRECV	Process received commands and messages.
	MSG	Prepares and sends message requests to REX.
	CMDPROC	Executes commands passed to it in the CMDRESP buffer after an alert from DMTREX indicates that a command has been entered.
	SVMRINIT	Initializes line driver.
	AXSGET	Provides interface to AXS for input files.
	TODEBCD	Converts System/370 TOD to EBCDIC date and time.
	VMRTILT	Terminates line driver task.
	MSGTRANS	Unstacks MSG and command element and blocks for transmission.

Figure 3-1. RSCS Modules and Their Subroutines (Part 12 of 13)

Module Name	Entry Pt /Routine	Function
	VMSB2CP	Deblocks VM/370 4K spool page buffer.
	PACK	Packs a line into transmission buffer format.
DMTVMC	CTCGO	This line driver supports the use of a channel-to-channel adapter between two processors
	XECUTE	running VM/370. This routine requests the supervisor to execute I/O operations. After initiating the I/O operation, the routine waits for either a command to be entered or the completion of the requested I/O operation.
	LINEIO	Executes (calling XECUTE) I/O operations on the VMC and checks the final state, consequently setting the IOERR flag in the DEVFLAG byte.
	GETBLOCK	Processes input files from the VM/370 spool file system.
	MSGTRANS	Builds message buffer for transmission.
	PUTBLOCK	Relocates received spool block records, and sends them to the VM/370 spool system.
	MSGRECV	Processes received commands and messages.
	CMDPROC	Executes commands passed to it in the CMDRESP buffer after an alert from DMTREX indicating that a command has been entered.
	MSG	Prepares and sends requests to the specialized task REX, to write messages on the operator's console.
	MAKEBLOC	Sets up for a call to VMSB2CP.
	AXSGET	Passes requests to AXS to open, close, and delete the spool files that the VMC task is processing.
	TODEBCD	Converts System/370 TOD to EBCDIC date and time.
	PARMGET	Line scanning subroutine.
	CONVBLK	Converts a spool page buffer from real reader to virtual unit record output format.
	CTCINIT	VMC initialization routine.
	CTCERROR	Writes the terminal I/O error message and calls CTCTERM.
	CTCTERM	Terminates the VMC task.
DMTWAT	DMTWATEP	Called directly from task programs by a BALR instruction. It provides event synchronization by suspending a task's execution until some specified event is signalled complete by another process in the system.

Figure 3-1. RSCS Modules and Their Subroutines (Part 13 of 13)

Module-to-Module Execution Transfers (BALRs)

Figure 3-2 lists the code locations at which control is passed from one RSCS routine to another via a BALR instruction.

RSCS Module	BALR to Module	At Label	Comments
DMTAKE	DMTDSP	TAKEXIT	Resumes dispatching; processing of a TAKE request is complete.
	DMTPST	TAKEMUTE	Signals a task that it must process a TAKE request.
	DMTQRQ	TAKEMUTE	Frees a GIVE element.
DMTASK	DMTDSP	TAEXIT	Resumes dispatching; processing of a task request has completed.
	DMTPST	TAGPURGE	Signals the termination of a task.
	DMTQRQ	TAFREEOK	Frees a terminated task element.
	DMTQRQ	TAGPURGE	Frees a terminated GIVE element.
	DMTQRQ	TAMAKE	Gets a queue element for a new task.
	DMTQRQ	TAQPTEST	Frees requested elements for a terminated task.
	DMTQRQ	TASQTEST	Frees an I/O element associated with a task being purged.
DMTASY	DMTDSP	ASEXIT	Resumes dispatching; processing of an asynchronous exit request has completed.
	DMTQRQ	ASQEND	Gets a free queue element; frees a terminated queue element.
	DMTQRQ	ASQGOT	Gets a free queue element; frees a terminated queue element.
DMTAXM	DMTAKE	AXSACCPT	Takes a request for DMTAXS services from another task.
	DMTASY	AXSIGSET	Requests an asynchronous exit for task asynchrononous alerts.
	DMTASY	AXSIGSET	Requests an asynchronous exit for reader X'001'.
	DMTAXA	DMTAXAAC	Execute the accept account record routine.
		DMTAXASE	Execute the send account record routine.
		DMTAXAPU	Execute the purge account record routine.
		DMTAXARE	Execute the receive account record routine.
		DMTAXATA	Execute the tag priority change routine.
	DMTCOM	GETLINK	Gets a link table entry.
	DMTCOM	OPENIRTY	Gets a page of main storage.
	DMTCOM	CPENCLNK	Gets a page of main storage.
	DMTCOM	TODEBCD	Converts System/370 TOD to EBCDIC date and time.

Figure 3-2. Module-to-Module Execution Transfers (BALRs) (Part 1 of 7)

RSCS Module	BALR to Module	At Label	Comments
	DMTGIV	MSGDO	Gives a message element to DMTMGX for processing.
	DMTPST	AXSALRT1	Signals acceptance of a command to process.
	DMTPST	AXSASYIO	Signals arrival of a request for an asynchronous exit.
	DMTSIG	ACCEFIND	Alerts a line driver task that a newly arrived file has been accepted.
	DMTSIG	CHANDONE	Alerts a line driver task.
	DMTWAT	AXSCYCLE	Waits for a request for DMTAXS services.
	DMTWAT	MSGDO	Waits until processing by DMTGIV has completed.
DMTCMX	DMTCOM	QYOLINK	Finds a link table entry.
	DMTCOM	TODEBCD	Converts a System/370 TOD to EBCDIC date and time.
	DMTCRE	STALNGOT	Creates a line driver task, as specified in the START command.
	DMTMGX	CMXDOIT	Writes a message resulting from command processing.
	DMTMGX	CMXM001	Writes a message showing the number of free pages in storage.
	DMTMGX	CMXM003B	Writes a message showing the command now being executed by RSCS.
	DMTMGX	DISCHARG	Writes a message resulting from DISCONN command processing.
	DMTMGX	QYM654	Writes a message resulting from QUERY command processing.
	DMTMGX	QYM655	Writes a message resulting from QUERY command processing.
	DMTMGX	QYSYMSG	Writes a message resulting from command processing.
	DMTREX	DISCONN	DIAGNOSE instruction entry to CP console function.
	DMTREX	DISCHARG	DIAGNOSE instruction entry to CP console function.
	DMTSIG	CMXALRDY	Alerts a task for command processing.
	DMTSIG	STACREAT	Alerts DMTLAX to validate a line address used in a START command.
	DMTMGX	SENDIT	Sends a line of CPQUERY response data back to the issuer.
DMTCOM	DMTDSP	MFIXIT	Requests dispatching of a task for which a message has been stacked for transmission.
	DMTDSP	MFOXIT	Requests dispatching of a task for which a message has been unstacked for transmission.
	DMTSTO	GETPTRY	Requests main storage allocation.
	DMTIOM	CFILDOIO	Requests the I/O manager to read one DASD block from a file on a CMS-type system disk.
	DMTSTO	CRETRYIT	Requests main storage for the creation of a task.
	DMTWAT	CFILDOIO	Waits for a read I/O request to complete.
DMTCRE	DMTASK	CREQTASK	Requests the supervisor to start a new task.

Figure 3-2. Module-to-Module Execution Transfers (BALRs) (Part 2 of 7)

RSCS Module	BALR to Module	At Label	Comments
DMTEXT	DMTDSP	EXTGO	Resumes dispatching; processing of an external interruption is complete.
DMTGIV	DMTDSP	GIVEXIT	Resumes dispatching; processing of a GIVE request is complete.
	DMTPST	GIVESNIF	Signals a task to begin processing a GIVE request.
	DMTQRQ	GIVESCAN	Gets a free queue element.
DMTINI	DMTDSP	INIQDONE	Dispatches the first task.
	DMTQRQ	INIQDONE	Initializes the queue of free elements.
DMTIOM	DMTDSP	IODISPCH	Resumes dispatching; processing of an I/O request is complete.
	DMTPST	IONORMAL	Signals completion of an I/O event.
	DMTPST	IOPUNT	Signals an error on a request for a queue element.
	DMTQRQ	DMTIOMRQ	Gets an element for an I/O request.
	DMTQRQ	IODISMIS	Frees an element used for a SENSE request.
	DMTQRQ	IONORMAL	Frees an element used in an I/O request.
	DMTQRQ	IOUNITCK	Gets an element for a SENSE request.
DMTIRX	DMTCRE		Initiate AXS and LAX tasks.
	DMTASY		Initializes an asynchronous exit address.
	DMTCOM	IRXBLK	Obtains storage for supervisor use as buffer space.
	DMTCOM	DIRECT	Open 'RSCS DIRECT'
	DMTCOM	DIRREAD	Read 'RSCS DIRECT'
DMTLAX	DMTASY	LAXINIT	Sets up an asynchronous exit for DMTLAX.
	DMTWAT	LAXHANG	Terminates DMTLAX.
DMTMIN			Initializes MSUP after DMTINI loads it.
DMTMGX	DMTCOM	MGXBUILT	Gets a link table entry.
	DMTCOM	MGXTOLOC	Stacks a message.
	DMTREX	MGXNOPR	Writes a message to a local VM/370 userid.
	DMTREX	MGXNOVM	Writes a message to the VM/370 operator.
	DMTSIG	MGXBUILT	Alerts an originating task that a message has been handled.
DMTNPT	DMTASY	NPTNOPAS	Sets up an asynchronous interrupt for DMTNPT.
	DMTCOM	AXSMENQ	Enqueues a message on the message stack for processing by DMTMGX.
	DMTCOM	MSG2780	Unstacks a message for transmission to a remote station.
	DMTCOM	NPTNOPAS	Gets a page of storage for use as DMTNPT buffers.
	DMTCOM	TODEBCD	Converts System/370 TOD to EBCDIC date and time.
	DMTGIV	AXSGET	Requests DMTAXS to open a file.
	DMTGIV	AXSPURGE	Requests DMTAXS to purge a file.
	DMTGIV	COMMANDS	Passes a command element to DMTREX for processing by DMTCMX.
	DMTGIV	KLOGIT	Requests DMTAXS to open the log trace file for output.

Figure 3-2. Module-to-Module Execution Transfers (BALRs) (Part 3 of 7)

RSCS Module	BALR to Module	At Label	Comments
	DMTGIV	LINEDROP	Requests DMTAXS to close a file.
	DMTGIV	LOGCLOSE	Requests DMTAXS to close the log trace file for output.
	DMTGIV	MSG1	Passes a message element to DMTMGX for processing.
	DMTGIV	PUTCLS1	Requests DMTAXS to close a file for output.
	DMTGIV	PUTOPEN	Requests DMTAXS to open a file for output.
	DMTGIV	TASKILL	Requests DMTREX to terminate the requesting NPT line driver.
	DMTIOM	LOGCONT1	Requests an I/O operation for the LOG routine.
	DMTIOM	LOGPRINT	Prints a LOG message.
	DMTIOM	XECUTE	Requests an I/O operation (general usage by DMTNPT).
	DMTPST	AXSALRT1	Signals that DMTNPT accepted a command.
	DMTWAT	AXSGET	Waits for a request to open a file to complete processing.
	DMTWAT	AXSPURGE	Waits for a request to purge a file to complete processing.
	DMTWAT	COMMANDS	Waits for DMTCMX to process a command.
	DMTWAT	KLOGIT	Waits for completion of a request to open the log trace file for processing.
	DMTWAT	LINEDROP	Waits for a request to close a file to complete processing.
	DMTWAT	LOGCLOSE	Waits for a request to close the log trace file when processing is complete.
	DMTWAT	LOGCONT1	Waits for an I/O operation to complete logging processing.
	DMTWAT	MSG1	Waits for message processing to complete.
	DMTWAT	PUTCLS1	Waits for a request to close a file to complete processing.
	DMTWAT	PUTOPEN	Waits for completion of a request to open a file for processing.
	DMTWAT	TASKILL	Waits for task termination processing to complete.
	DMTWAT	XECQWAIT	Waits for an I/O operation to complete.
DMTPOW	DMTASY	SETNOBUF	Sets up an asynchronous exit for DMTPOW.
	DMTCOM	ASYNENQ	Stacks a message to be transmitted by DMTPOW.
	DMTCOM	BUFSDONE	Gets a page of storage for DMTPOW I/O tasks.
	DMTCOM	IBLDBUFS	Gets a page of storage for DMTPOW TP buffers.
	DMTCOM	MSGPROC1	Unstacks a message for transmission to a remote station.
	DMTGIV	AXS	Requests services of DMTAXS for the POW line driver task.
	DMTGIV	AXSGET	Requests DMTAXS to give a file for transmission.
	DMTGIV	AXSPURGE	Requests DMTAXS to purge a file.
	DMTGIV	EOJ	Requests termination of the POW line driver task.
	DMTGIV	MSG1	Gives a message to DMTMGX for processing.

Figure 3-2. Module-to-Module Execution Transfers (BALRs) (Part 4 of 7)

RSCS Module	BALR to Module	At Label	Comments
	DMTIOM	I27XXIO	Performs the initial I/O operation for the POW line driver task.
	DMTIOM	LOGPRINT	Requests an I/O operation (logs an I/O operation).
	DMTIOM	PCONT4	Requests an I/O operation for the printer.
	DMTIOM	RSIO	Requests a start I/O for the adapter.
	DMTIOM	UCONT2	Requests an I/O operation for the punch.
	DMTWAT	ALLCHK	Waits for the DMTPOW synch lock to be posted (waits for a request to process).
	DMTWAT	AXS	Waits for completion of an event by DMTAXS.
	DMTWAT	AXSGET	Waits for DMTAXS to GIVE a file for transmission.
	DMTWAT	AXSPURGE	Waits for DMTAXS to purge a file.
	DMTWAT	DEOJCONT	Wait for HDV.
	DMTWAT	DEOJGO	Wait for disable of adapter.
	DMTWAT	EOJ	Terminates the POW line driver task by issuing a terminal WAIT request.
	DMTWAT	LOGPRINT	Waits for I/O logging to complete.
	DMTWAT	MSG1	Waits until GIVE to DMTMGX is complete.
	DMTWAT	RISIO1	Waits for initial SIO for the DMTPOW line driver to complete.
DMTREX	DMTAKE	REXACCPT	Accepts a request to process a VM/370 file.
	DMTASK	QUIESE	Requests task termination.
	DMTASK	TERTKILL	Requests task termination.
	DMTASY	REXICGOT	Initializes an asynchronous exit.
	DMTCMX		Pass the Special Message to the Command Processor.
	DMTCOM	REXFLUSH	Requests DMTMGX to write any queued messages.
	DMTCOM	REXOUTRY	Removes a message for the message stack and writes it to the console.
	DMTCRE	REXICGOT	Creates the tasks DMTAXS and DMTLAX.
	DMTDSP	REXDQUIT	Terminates dispatching due to program check.
	DMTDSP	REXHEXIT	Resumes dispatching after program check processing.
	DMTIOM	REXCONON	Requests an I/O operation (console write).
	DMTIOM	REXFCONF	Requests an I/O operation (console write).
	DMTIOM	REXQUERY	Requests an I/O operation (console read).
	DMTMGX	MSG	Passes a message element to DMTMGX for processing.
	DMTMGX	TERMSET	Writes a task terminated message.
	DMTPST	REXASYN	Signals a console attention.
	DMTPST	REXHALT	Signals that DMTREX is undispatchable due to program check.
	DMTWAT	QUIESE	Waits for a task to terminate.
	DMTWAT	QUICK	Waits for task I/O to terminate.
	DMTWAT	REXSWAIT	Waits for a console write to complete.
	DMTWAT	REXWAIT	Waits for completion of an event.

Figure 3-2. Module-to-Module Execution Transfers (BALRs) (Part 5 of 7)

RSCS Module	BALR to Module	At Label	Comments
DMTRGX			Process command and message routing request elements.
DMTSIG	DMTDSP	ALSCAN ALNOGO	Resumes dispatching; processing of an alerted task has completed.
DMTSML	DMTASY	SETNCBUF	Sets up an asynchronous exit for DMTSML.
	DMTCOM	ASYNENQ	Stacks a message to be transmitted by DMTSML.
	DMTCOM	BUFSDONE	Gets a page of storage for DMTSML I/O tasks.
	DMTCOM	IBLDBUFS	Gets a page of storage for DMTSML TP buffers.
	DMTCOM	MSGPROC1	Unstacks a message for transmission to a remote station.
	DMTCOM	TODEBCD	Converts System/370 TOD to EBCDIC date and time.
	DMTGIV	AXS	Requests services of DMTAXS for the SML line driver task.
		KLOGIT	Requests DMTAXS to open a log trace output file.
		LOGCLOSE	Requests DMTAXS to close the log trace output file.
	DMTGIV	AXSGET	Requests DMTAXS to give a file for transmission.
	DMTGIV	AXSPURGE	Requests DMTAXS to purge a file.
	DMTGIV	EOJ	Requests termination of the SML line driver task.
	DMTGIV	MSG1	Gives a message to DMTMGX for processing.
	DMTGIV	WGET1A	Requests that a message be written to the RSCS console; pass a command to DMTREX.
	DMTIOM	I27XXIO	Performs the initial I/O operation for the SML line driver task.
	DMTIOM	JOUT1	Requests an I/O operation; sets up job processing controls.
	DMTIOM	PCONT2 PLINE	Requests an I/O operation (sets up printer controls.
	DMTIOM	RSIO	Requests a start I/O for the adapter.
	DMTIOM	UOUT2	Requests an I/O operation (sets up punch controls).
	DMTIOM	WRLOG1	Requests an I/O operation (logs an I/O operation).
	DMTPST	ASYNRET	Posts the reader synch lock.
	DMTWAT	ALLCHK	Waits for the DMTSML synch lock to be posted (waits for a request to process).
	DMTWAT	AXS	Waits for completion of an event by DMTAXS.
	DMTWAT	AXSGET	Waits for DMTAXS to GIVE a file for transmission.
	DMTWAT	AXSPURGE	Waits for DMTAXS to purge a file.
	DMTWAT	EOJ	Terminates the SML line driver task by issuing a terminal WAIT request.
		KLOGIT	Waits for DMTAXS to open a log trace output file.
		LOGCLOSE	Waits for DMTAXS to close a log trace output file.

Figure 3-2. Module-to-Module Execution Transfers (BALRs) (Part 6 of 7)

RSCS Module	BALR to Module	At Label	Comments
	DMTWAT	MSG1	Waits until GIVE to DMTMGX is complete.
	DMTWAT	RISIO1	Waits for initial SIO for the DMTSML line driver to complete.
	DMTWAT	WGET1A	Waits until message processing has completed.
	DMTWAT	WRLOG1	Waits for I/O logging to complete.
DMTSTO	DMTDSP	MAINDONE	Resumes dispatching; a request for a page of storage has been processed.
DMTWAT	DMTDSP	WAITGO	Resumes dispatching; processing of a WAIT request has completed.

Figure 3-2. Module-to-Module Execution Transfers (BALRs) (Part 7 of 7)

Control Flow Diagrams

Figures 3-3 through 3-11 illustrate the flow of control through the routines that make up the folllowing parts of RSCS:

- Multitasking supervisor MSUP
- REX system service task
- AXS system service task
- SML line driver task
- NPT line driver task
- NJI line driver task
- VMB line driver task
- VMC line driver task
- POW line driver task

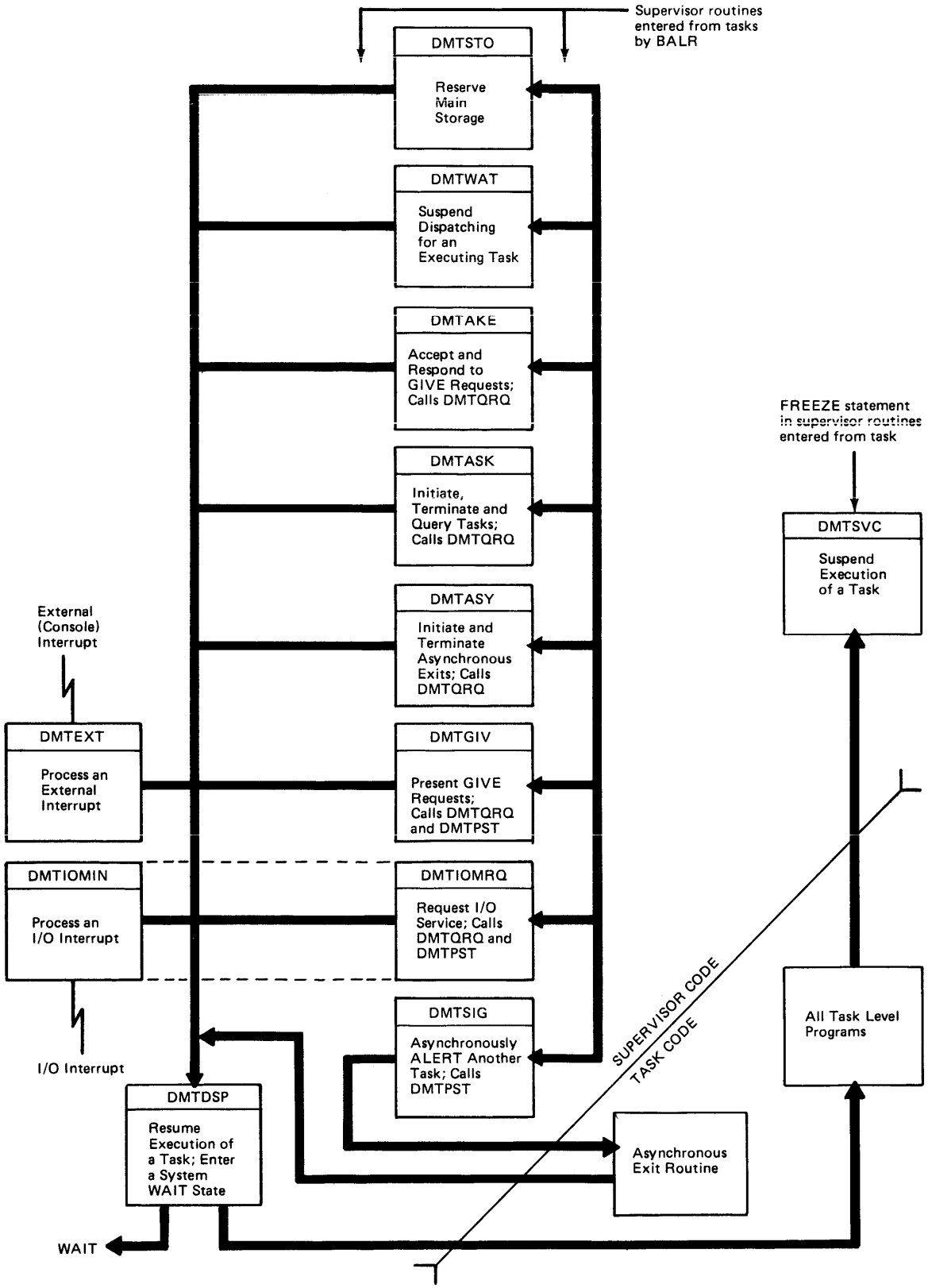


Figure 3-3. Program Organization for the Multitasking Supervisor MSUP

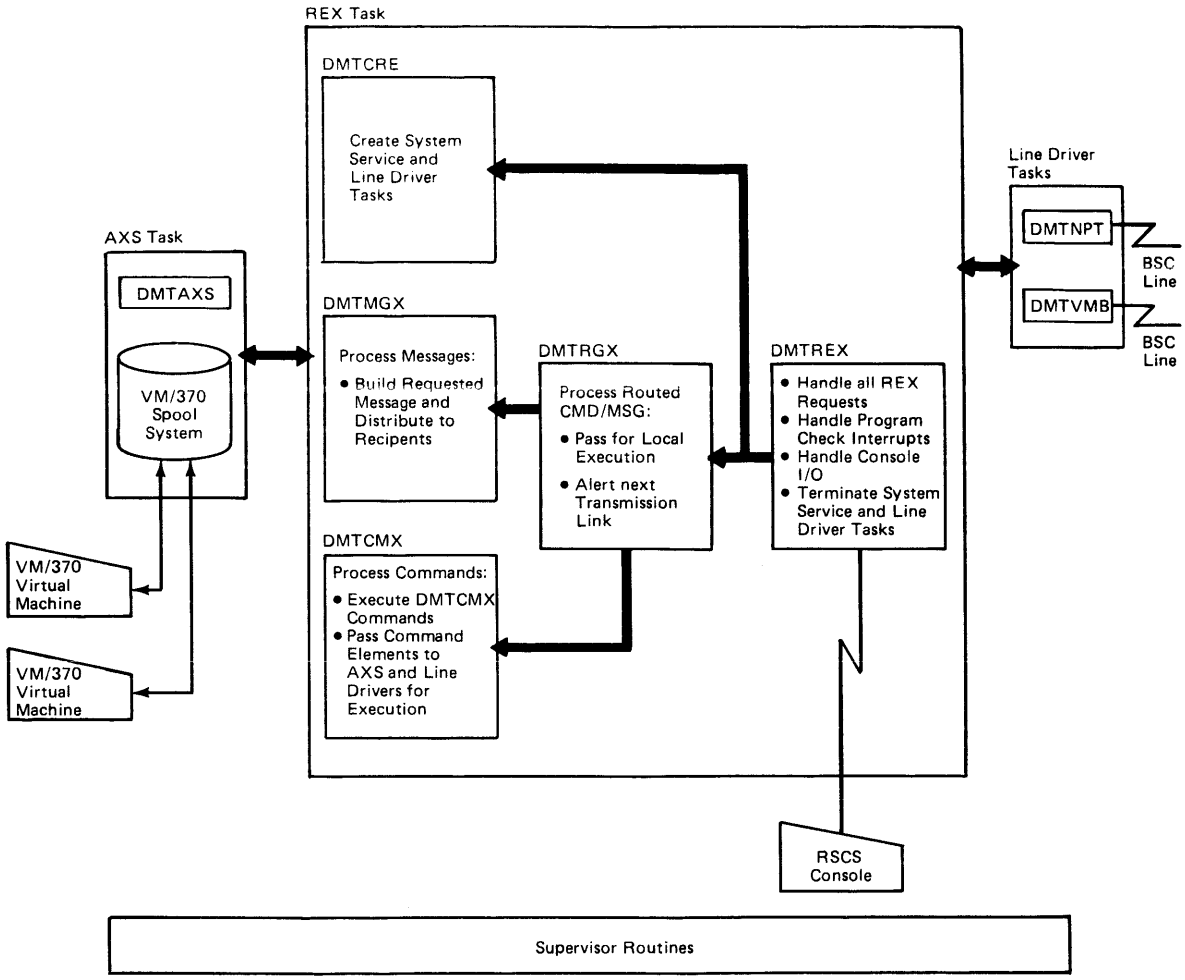


Figure 3-4. Program Organization for the REX System Service Task

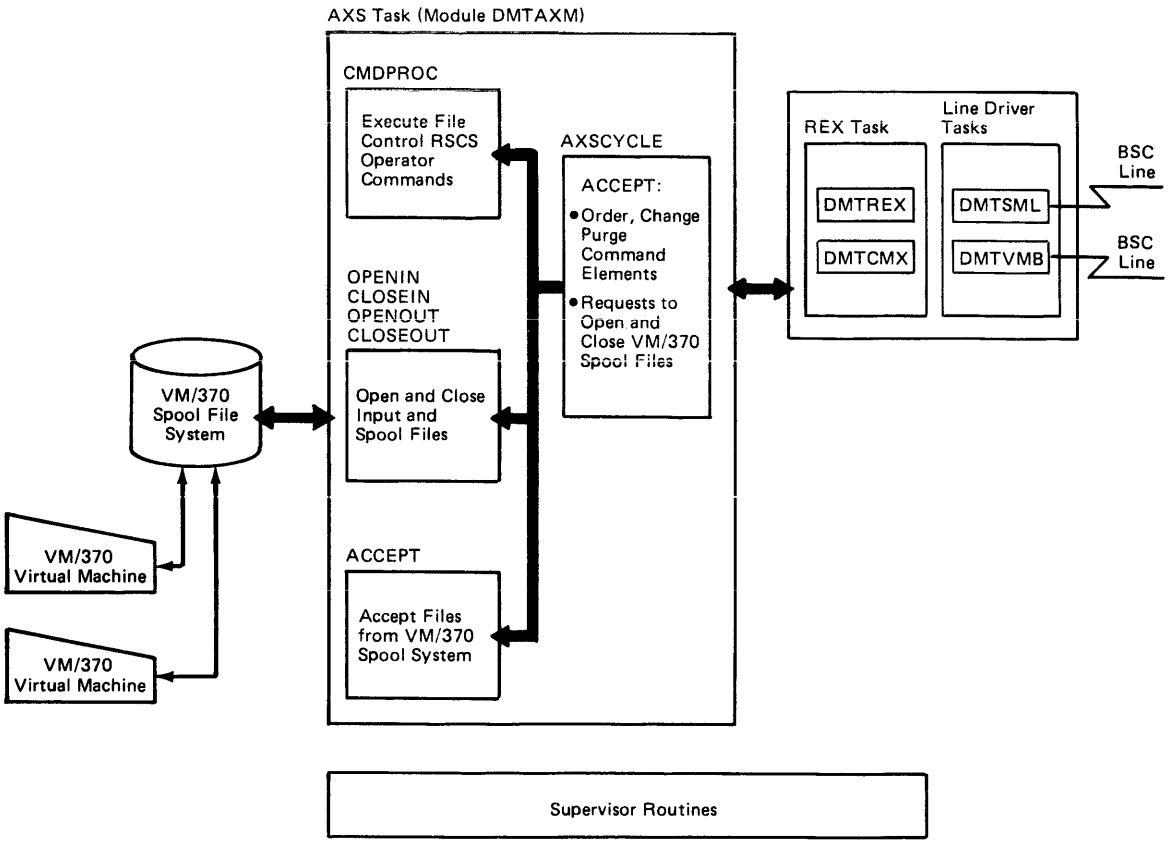


Figure 3-5. Program Organization for the AXS System Service Task

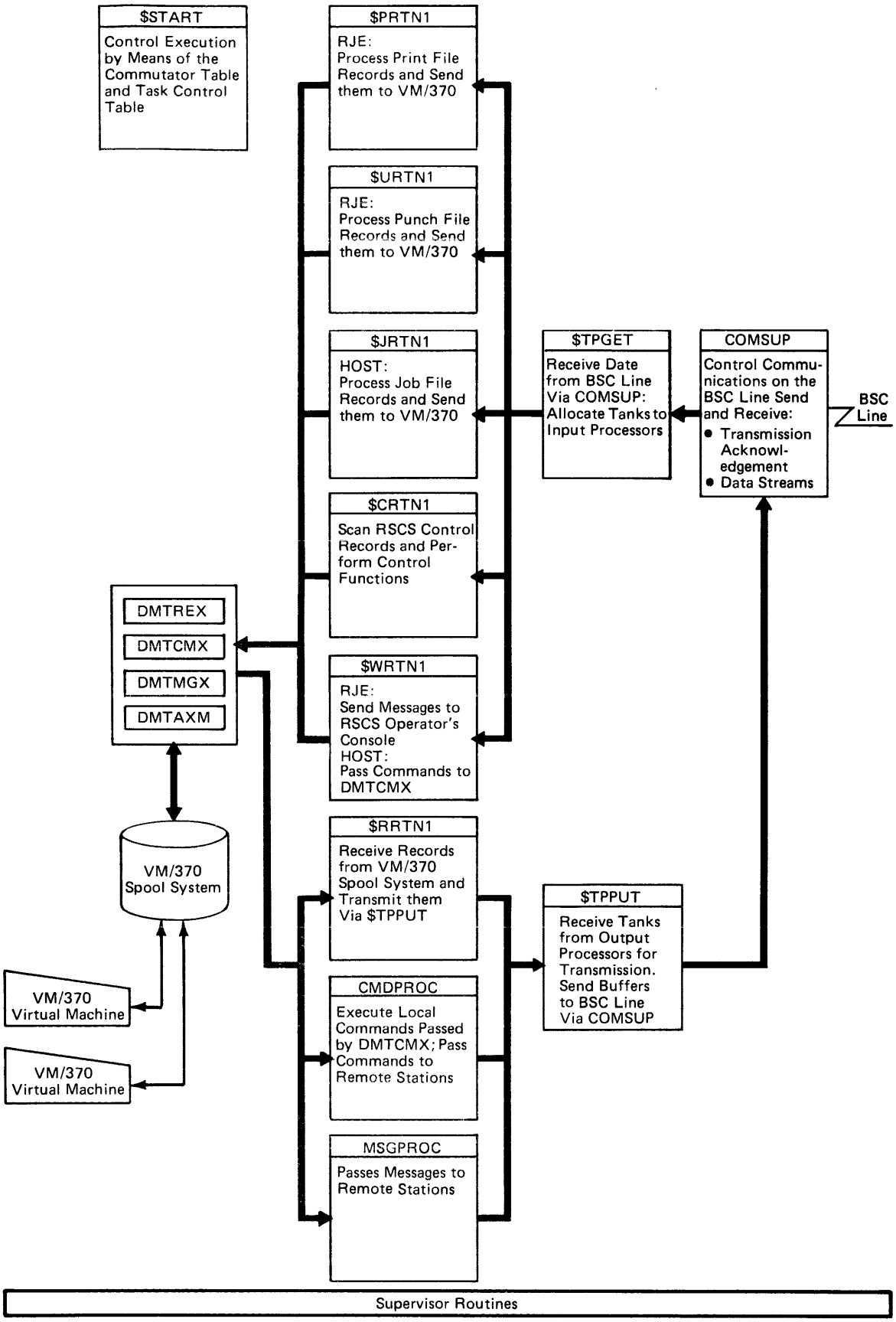


Figure 3-6. Program Organization for the SML Line Driver Task

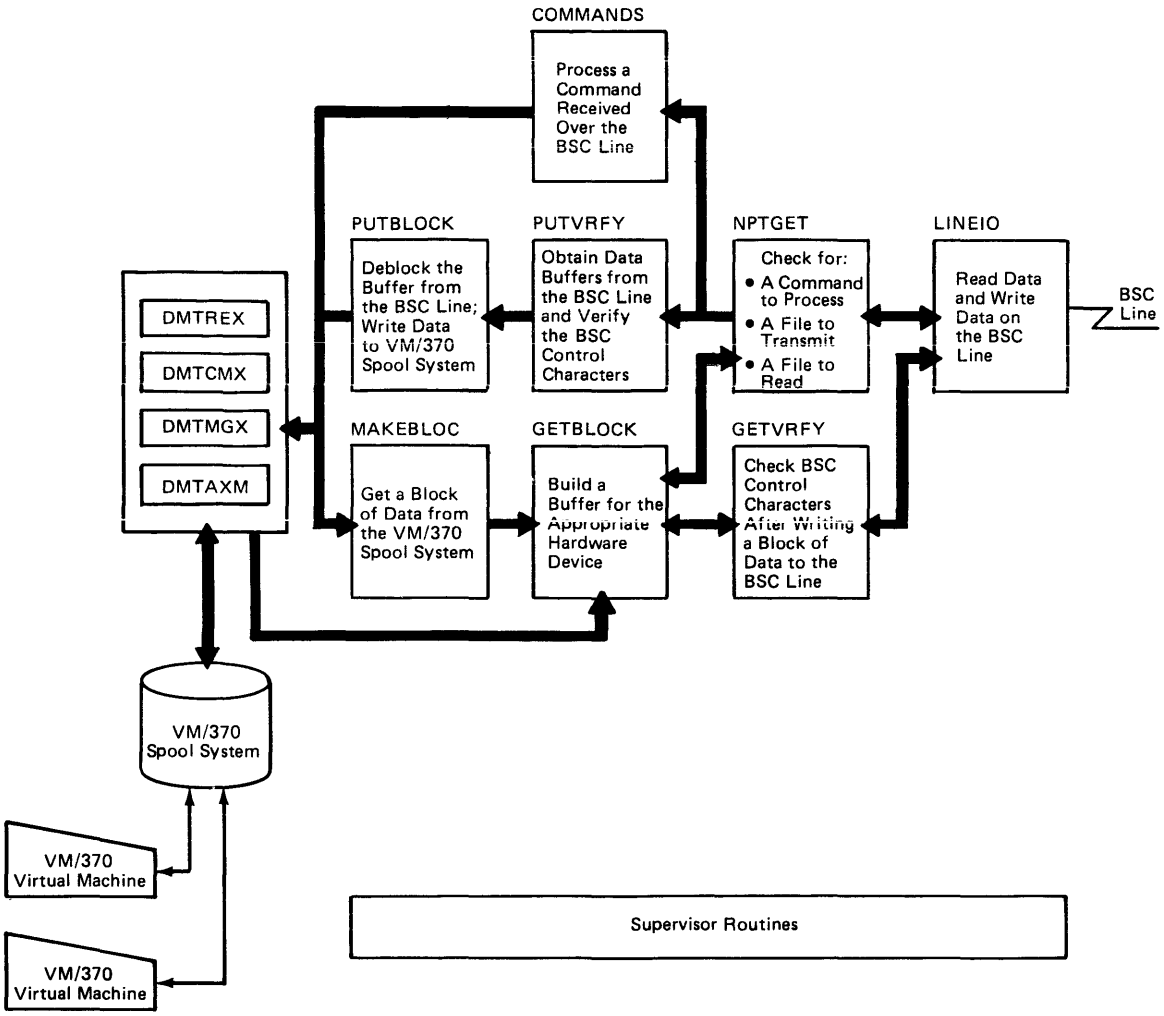


Figure 3-7. Program Organization for the NPT Line Driver Task

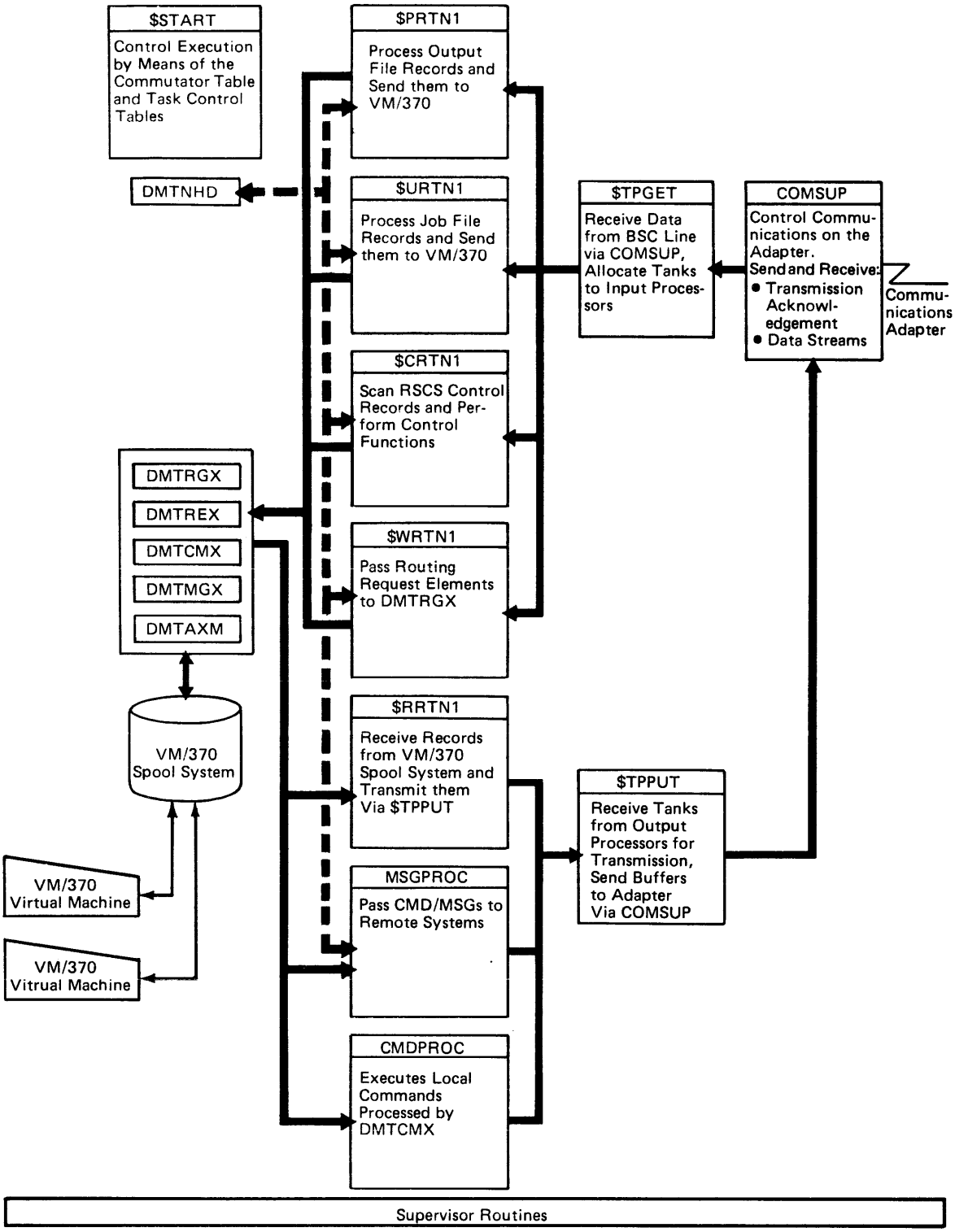


Figure 3-8. Program Organization for the NJI Line Driver Task

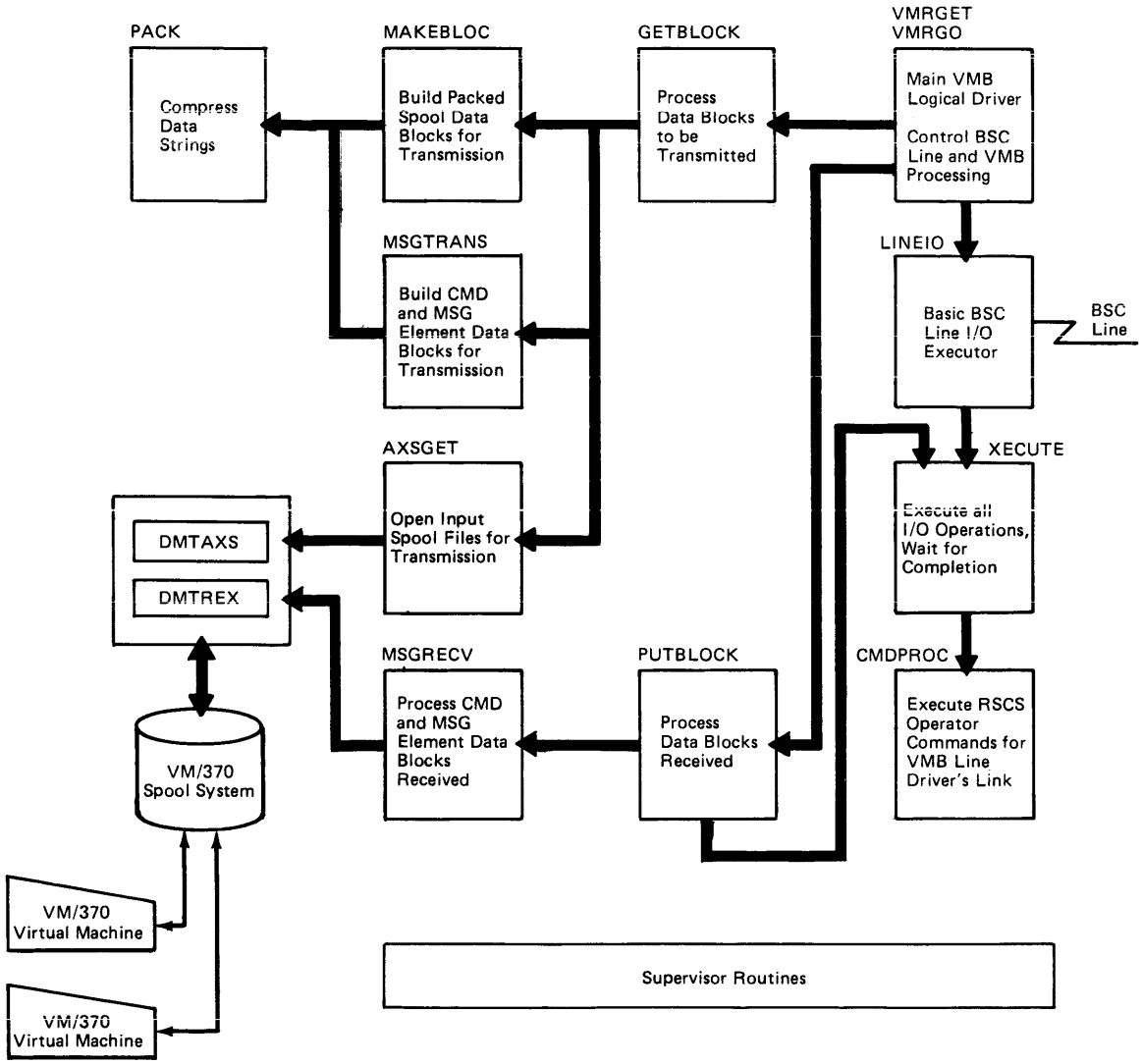


Figure 3-9. Program Organization for the VMB Line Driver Task

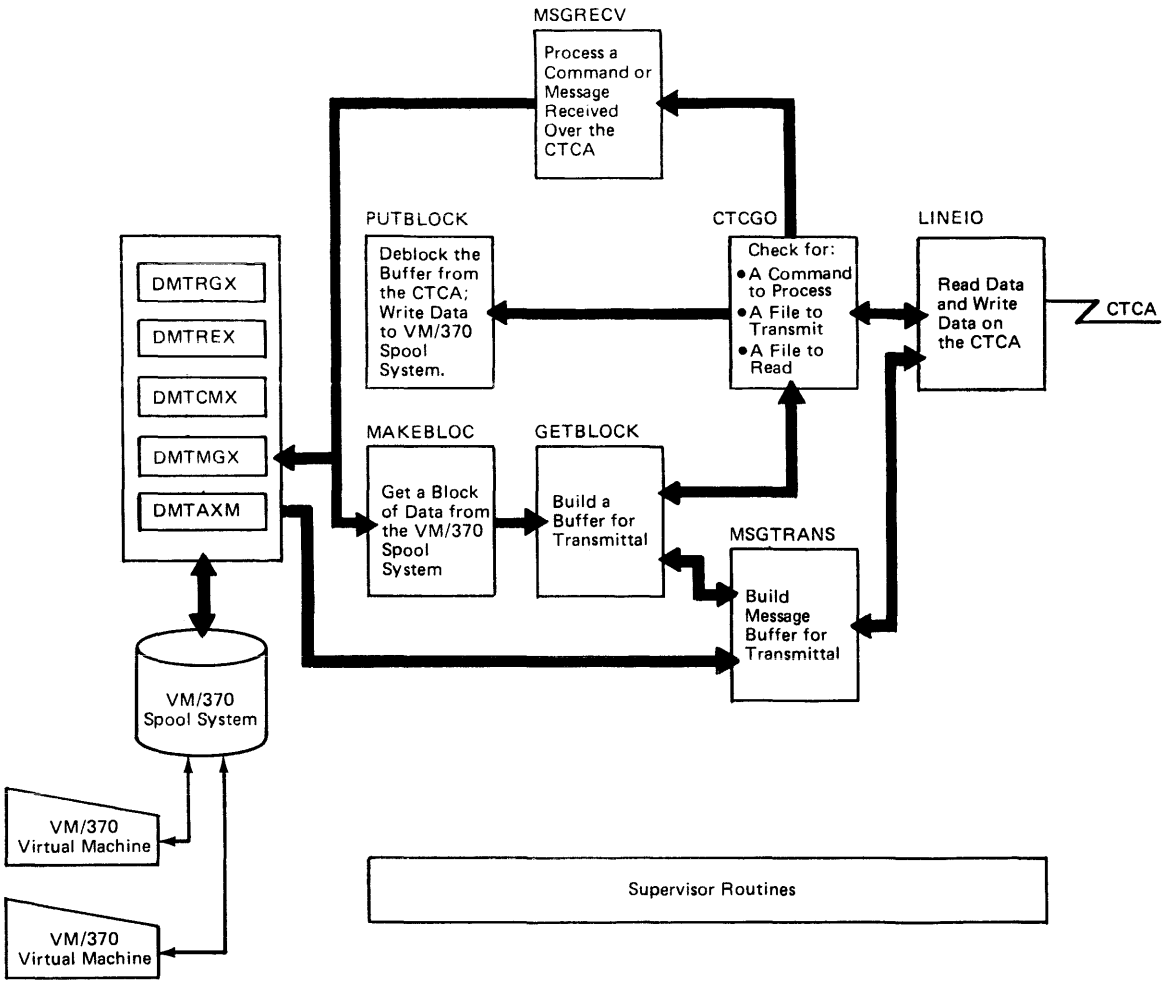


Figure 3-10. Program Organization for the VMC Line Driver Task

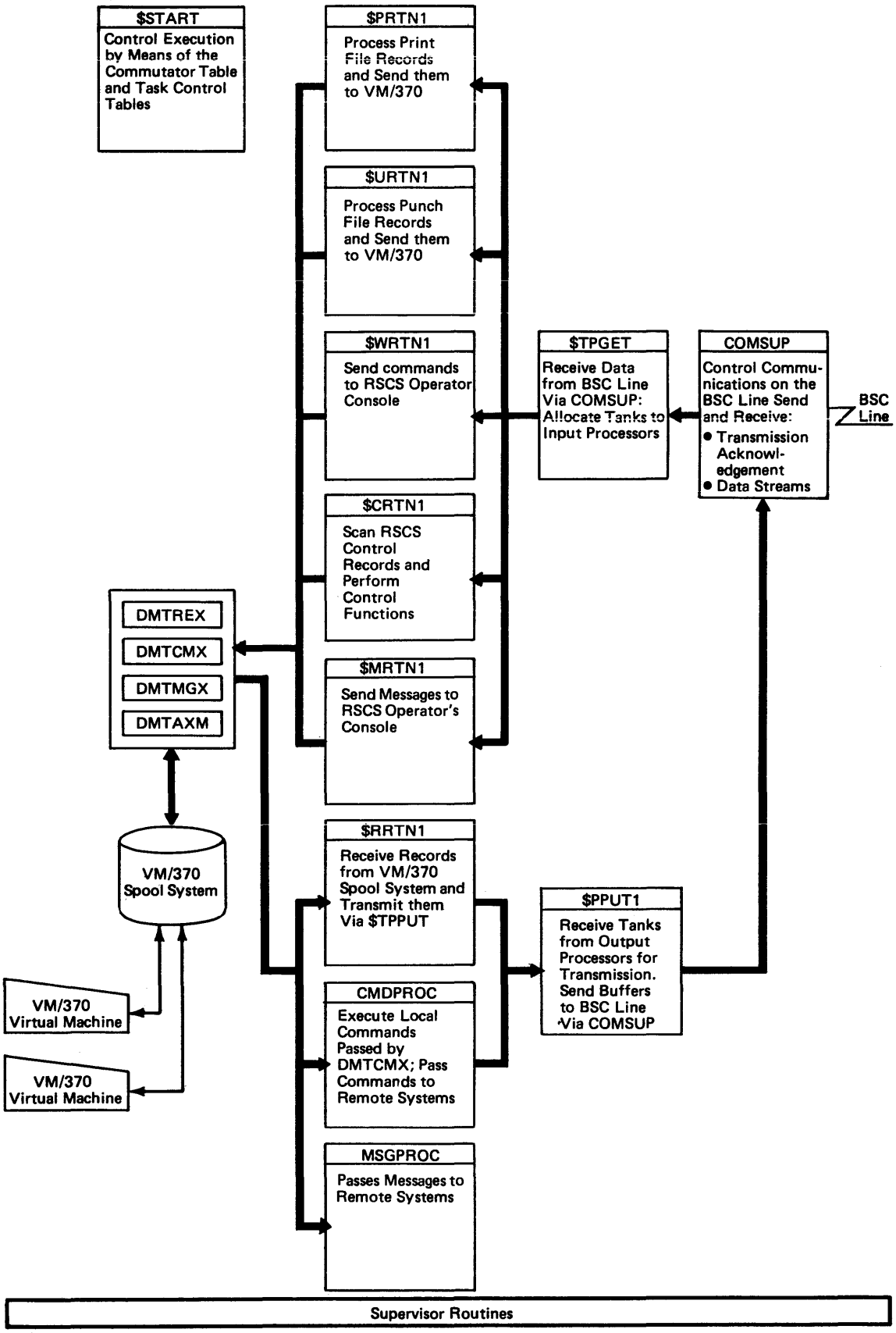


Figure 3-11. Program Organization for the POW Line Driver Task

Section 4: Directory

Name	Type	Object	Ref on Page	Ref in Fig 3-1
ACCEPT	Routine	DMTAXM		yes
ASYNEXIT	Routine	DMTNCM		yes
ASYNEXIT	Routine	DMTPOW		yes
ASYNEXIT	Routine	DMTSML		yes
AXSALERT	Routine	DMTVMB	92	
AXSALERT	Routine	DMTVMC	96	
AXSASYIO	Routine	DMTAXM		yes
AXSCYCLE	Routine	DMTAXM		yes
AXSGET	Routine	DMTNCM		yes
AXSGET	Routine	DMTNPT		yes
AXSGET	Routine	DMTPOW		yes
AXSGET	Routine	DMTSML		yes
AXSGET	Routine	DMTVMB	91	yes
AXSGET	Routine	DMTVMC		yes
AXSPURGE	Routine	DMTVMB	91	
AXSINIT	Routine	DMTAXM		yes
BUFFINIT	Routine	DMTNPT		yes
CERROR	Routine	DMTNCM		yes
CERROR	Routine	DMTPOW		yes
CERROR	Routine	DMTSML		yes
CLOSEOUT	Routine	DMTAXM		yes
CLOSIN	Routine	DMTAXM		yes
CMDPROC	Routine	DMTAXM		yes
CMDPROC	Routine	DMTNCM		yes
CMDPROC	Routine	DMTNPT		yes
CMDPROC	Routine	DMTPOW	65, 72	yes
CMDPROC	Routine	DMTSML	59	yes
CMDPROC	Routine	DMTVMB	92	yes
CMDPROC	Routine	DMTVMC	95	yes
CMXHIT	Routine	DMTCMX		yes
CMXULOOP	Routine	DMTCMX		yes
CMXREORD	Routine	DMTCMX		yes
COMMANDS	Routine	DMTNPT		yes
COMSUP	Routine	DMTNCM		yes
COMSUP	Routine	DMTPOW	65	yes
COMSUP	Routine	DMTSML	59	yes
CONVBLK	Routine	DMTVMC		yes
CONW	Routine	DMTIRX		yes
CTCERROR	Routine	DMTVMC		yes
CTCGO	Routine	DMTVMC	94	yes
CTCINIT	Routine	DMTVMC		yes
CTCTERM	Routine	DMTVMC		yes
DEACT	Routine	DMTREX		yes
DECGET	Routine	DMTAXM		yes
DECPUT	Routine	DMTAXM		yes
DECPUT	Routine	DMTCMX		yes
DECPUT	Routine	DMTIRX		yes
DEFINE	Routine	DMTAXM		yes
DETACH	Routine	DMTAXM		yes
DIRECT	Routine	DMTIRX		yes
DMTAKE	Object,CSECT	DMTAKE		yes
DMTAKEEP	EP	DMTAKE		yes
DMTASK	Object,CSECT	DMTASK		yes
DMTASKEP	EP	DMTASK		yes
DMTASY	Object,CSECT	DMTASY		yes
DMTASYEP	EP	DMTASY		yes
DMTAXA	Object	DMTAXA	29	yes

Name	Type	Object	Ref on Page	Ref in Fig 3-1
DMTAXAAC	Routine	DMTAXA		yes
DMTAXAPU	Routine	DMTAXA		yes
DMTAXARE	Routine	DMTAXA		yes
DMTAXATA	Routine	DMTAXA		yes
DMTAXM	Object,CSECT	DMTAXM		yes
DMTAXMEP	EP	DMTAXM		yes
DMTAXS	Object	DMTAXS	30	
DMTCMX	Object,CSECT	DMTCMX	28, 29	yes
DMTCMXEP	EP	DMTCMX		yes
DMTCOM	Object,CSECT	DMTCOM	29	yes
DMTCOMEP	EP	DMTCOM		yes
DMTCRE	Object,CSECT	DMTCRE	28, 37	yes
DMTCREEP	EP	DMTCRE		yes
DMTDSP	Object,CSECT	DMTDSP	22	yes
DMTDSPEP	EP	DMTDSP		yes
DMTEXT	Object,CSECT	DMTEXT		yes
DMTEXTEP	EP	DMTEXT		yes
DMTGIV	Object,CSECT	DMTGIV		yes
DMTGIVEP	EP	DMTGIV		yes
DMTINI	Object,CSECT	DMTINI	38	yes
DMTIOM	Object,CSECT	DMTIOM		yes
DMTIOMEP	EP	DMTIOM		yes
DMTIOMIN	Routine	DMTIOM	34	yes
DMTIOMRQ	Routine	DMTIOM	33, 34	yes
DMTIRX	Object,CSECT	DMTIRX	38	yes
DMTIRXEP	EP	DMTIRX		yes
DMTLAX	Object,CSECT	DMTLAX	29	yes
DMTLAXEP	EP	DMTLAX		yes
DMTMGX	Object,CSECT	DMTMGX	29, 30	yes
DMTMGXEP	EP	DMTMGX		yes
DMTMIN	Object,CSECT	DMTMIN	38	yes
DMTMINEP	EP	DMTMIN		yes
DMTMSG	Object,CSECT	DMTMSG		yes
DMTMSGEP	EP	DMTMSG		yes
DMTNCM	Object,CSECT	DMTNCM	29	yes
DMTNCMEP	EP	DMTNCM		yes
DMTNHD	Object,CSECT	DMTNHD	29	yes
DMTNHDDH	Routine	DMTNHD		yes
DMTNHDHO	Routine	DMTNHD		yes
DMTNHDJH	Routine	DMTNHD		yes
DMTNHDJT	Routine	DMTNHD		yes
DMTNHDMI	Routine	DMTNHD		yes
DMTNHDMO	Routine	DMTNHD		yes
DMTNIT	Object,CSECT	DMTNIT	29	yes
DMTNITEP	EP	DMTNIT		yes
DMTNJI	Object	DMTNJI	29	
DMTNPT	Object,CSECT	DMTNPT	29	yes
DMTNPTEP	EP	DMTNPT		yes
DMTPOW	Object,CSECT	DMTPOW	29	yes
DMTPOWEP	EP	DMTPOW		yes
DMTPRE	Object,CSECT	DMTPRE		yes
DMTPREEP	EP	DMTPRE		yes
DMTPST	Object,CSECT	DMTPST	22	yes
DMTPSTEP	EP	DMTPST		yes
DMTQRQ	Object,CSECT	DMTQRQ	37	yes
DMTQRQEP	EP	DMTQRQ		yes
DMTREX	Object,CSECT	DMTREX	29, 30	yes
DMTREXEP	EP	DMTREX		yes
DMTREXEC	Routine	DMTREX		yes
DMTREXTR	Routine	DMTREX		yes
DMTREXIN	Routine	DMTREX	39	yes
DMTREXPI	Routine	DMTREX	39	yes
DMTRGX	Object,CSECT	DMTRGX	29, 30	yes
DMTRGXEP	EP	DMTRGX		yes

Name	Type	Object	Ref on Page	Ref in Fig 3-1
DMTSIG	Object,CSECT	DMTSIG		yes
DMTSIGEP	EP	DMTSIG		yes
DMTSML	Object,CSECT	DMTSML	29	yes
DMTSMLEP	EP	DMTSML		yes
DMTSTO	Object,CSECT	DMTSTO	37	yes
DMTSTOEP	EP	DMTSTO		yes
DMTSVC	Object,CSECT	DMTSVC		yes
DMTSVCEP	EP	DMTSVC		yes
DMTVEC	Object,CSECT	DMTVEC		yes
DMTVECEP	EP	DMTVEC		yes
DMTVMB	Object,CSECT	DMTVMB	29	yes
DMTVMC	Object,CSECT	DMTVMC	29	yes
DMTWAT	Object,CSECT	DMTWAT	22, 26	yes
DMTWATEP	EP	DMTWAT		yes
DVASSIGN	Routine	DMTNHD		yes
EBCDEC	Routine	DMTIRX		yes
EBCDEC	Routine	DMTNHD		yes
EBCDEC	Routine	DMTNIT		yes
EBCHEX	Routine	DMTIRX		yes
EBCHEX	Routine	DMTNHD		yes
EBCHEX	Routine	DMTNIT		yes
FILGET	Routine	DMTCMX		yes
FILSELEC	Routine	DMTAXM		yes
FREEPAGE	Routine	DMTCOM		yes
FREESLOT	Routine	DMTAXM		yes
GENVNET	Routine	DMTIRX	39	yes
GETBLOCK	Routine	DMTNPT		yes
GETBLOCK	Routine	DMTVMB	89	yes
GETBLOCK	Routine	DMTVMC	95	yes
GETLINK	Routine	DMTAXM		yes
GETLINK	Routine	DMTCOM		yes
GETPAGE	Routine	DMTCOM		yes
GETPARM	Routine	DMTIRX		yes
GETROUTE	Routine	DMTAXM		yes
GETROUTE	Routine	DMTCOM		yes
GETSLOT	Routine	DMTAXM		yes
GETSUPAG	Routine	DMTCOM		yes
GETVRFY	Routine	DMTNPT	79	yes
GSUCCESS	Routine	DMTAXM		yes
HDRBUILD	Routine	DMTNHD		yes
HDROUT	Routine	DMTNHD		yes
HEADPREP	Routine	DMTNPT	77	yes
HEADPREP	Routine	DMTSML		yes
HEXGET	Routine	DMTAXM		yes
HEXGET	Routine	DMTCMX		yes
IBLDBUFS	Routine	DMTNIT		yes
INTCMD	Routine	DMTREX		yes
IOERRPRT	Routine	DMTVMB	93	
ISIO	Routine	DMTNCM		yes
ISIO	Routine	DMTPOW		yes
ISIO	Routine	DMTSML		yes
KEYWDGET	Routine	DMTCMX		yes
LINEIO	Routine	DMTNPT		yes
LINEIO	Routine	DMTVMB		yes
LINEIO	Routine	DMTVMC		yes
LTABGET	Routine	DMTCMX		yes
MAKEBLOC	Routine	DMTNPT	77	yes
MAKEBLOC	Routine	DMTVMB	91	yes
MAKEBLOC	Routine	DMTVMC		yes
MFI	Routine	DMTCOM		yes
MFO	Routine	DMTCOM		yes
MSG	Routine	DMTAXM		yes
MSG	Routine	DMTIRX		yes
MSG	Routine	DMTNCM		yes

Name	Type	Object	Ref on Page	Ref in Fig 3-1
MSG	Routine	DMTNHD		yes
MSG	Routine	DMTNIT		yes
MSG	Routine	DMTNPT		yes
MSG	Routine	DMTPOW		yes
MSG	Routine	DMTREX		yes
MSG	Routine	DMTSML		yes
MSG	Routine	DMTVMB	90	yes
MSG	Routine	DMTVMC		yes
MSGPROC	Routine	DMTNCM		yes
MSGPROC	Routine	DMTNPT		yes
MSGPROC	Routine	DMTPOW	65, 71	yes
MSGPROC	Routine	DMTSML	59	yes
MSGRECV	Routine	DMTVMB	90	yes
MSGRECV	Routine	DMTVMC	95	yes
MSGTRANS	Routine	DMTVMB	91	yes
MSGTRANS	Routine	DMTVMC	95	yes
NCMINIT	Routine	DMTNCM		yes
NITINIT	Routine	DMTNIT		yes
NPTERROR	Routine	DMTNPT		yes
NPTGET	Routine	DMTNPT	78	yes
NPTINIT	Routine	DMTNPT	79	yes
NPTLINK	Routine	DMTNPT		yes
NPTTERM	Routine	DMTNPT		yes
OPENADEV	Routine	DMTNHD		yes
OPENIN	Routine	DMTAXM		yes
PACK	Routine	DMTVMB	91	yes
PARMGET	Routine	DMTCMX		yes
PARMGET	Routine	DMTIRX		yes
PARMGET	Routine	DMTNHD		yes
PARMGET	Routine	DMTNIT		yes
PARMGET	Routine	DMTNPT		yes
PARMGET	Routine	DMTPOW		yes
PARMGET	Routine	DMTSML		yes
PARMGET	Routine	DMTVMC		yes
POWINIT	Routine	DMTPOW		yes
PUTBLOCK	Routine	DMTNPT		yes
PUTBLOCK	Routine	DMTVMB	90	yes
PUTBLOCK	Routine	DMTVMC	95	yes
PUTVRFY	Routine	DMTNPT	80	yes
QUIESCE	Routine	DMTREX		yes
RCMGET	Routine	DMTCOM		yes
RCMOPEN	Routine	DMTCOM		yes
REORDER	Routine	DMTAXM		yes
REQXEQ	Routine	DMTAXM		yes
REQXEQ	Routine	DMTREX		yes
REXCYCLE	Routine	DMTREX		yes
REXITERM	Routine	DMTREX		yes
REXPCHEX	Routine	DMTREX		yes
RGXCMD	Routine	DMTRGX		yes
RGXDOIT	Routine	DMTRGX		yes
RGXMSG	Routine	DMTRGX		yes
RGXMSGER	Routine	DMTRGX		yes
RGXNTHRE	Routine	DMTRGX		yes
RTABGET	Routine	DMTCMX		yes
SENDEOT	Routine	DMTNPT		yes
SMLINIT	Routine	DMTSML		yes
SVMRINIT	Routine	DMTVMB	92	yes
TAGCLOSE	Routine	DMTAXM		yes
TAGFIND	Routine	DMTAXM		yes
TAGGEN	Routine	DMTAXM		yes
TAGPLACE	Routine	DMTAXM		yes
TAGSCAN	Routine	DMTNHD		yes
TERMINAT	Routine	DMTREX		yes
TIMERSET	Routine	DMTREX		yes

Name	Type	Object	Ref on Page	Ref in Fig 3-1
TODEBCD	Routine	DMTAXM		yes
TODEBCD	Routine	DMTCMX		yes
TODEBCD	Routine	DMTCOM		yes
TODEBCD	Routine	DMTNPT		yes
TODEBCD	Routine	DMTSML		yes
TODEBCD	Routine	DMTVMB		yes
TODEBCD	Routine	DMTVMC		yes
TODS370	Routine	DMTAXM		yes
TODS370	Routine	DMTCOM		yes
TYPE	Routine	DMTIRX		yes
UNPEND	Routine	DMTAXM		yes
VCHANGE	Routine	DMTAXM		yes
VCLOSE	Routine	DMTAXM		yes
VMDEBLOK	Routine	DMTNCM		yes
VMDEBLOK	Routine	DMTPOW		yes
VMDEBLOK	Routine	DMTSML		yes
VMRTILT	Routine	DMTVMB	93	yes
VMSB2CP	Routine	DMTNPT		yes
VMSB2CP	Routine	DMTVMB		yes
VPURGE	Routine	DMTAXM		yes
VSPOOL	Routine	DMTAXM		yes
VTAGD	Routine	DMTAXM		yes
VTAGF	Routine	DMTAXM		yes
VTAGMSG	Routine	DMTAXM		yes
VTRANSFR	Routine	DMTAXM		yes
XECUTE	Routine	DMTNPT		yes
XECUTE	Routine	DMTVMB	93	yes
XECUTE	Routine	DMTVMC		yes
$CRTN1	Routine	DMTNCM		yes
$CRTN1	Routine	DMTPOW	65	yes
$CRTN1	Routine	DMTSML	59	yes
$JRTN1	Routine	DMTSML	59, 63	yes
$MRTN1	Routine	DMTPOW	65, 71	yes
$PRTN1	Routine	DMTNCM		yes
$PRTN1	Routine	DMTPOW	65, 69	yes
$PRTN1	Routine	DMTSML	59, 63	yes
$RRTN1	Routine	DMTNCM		yes
$RRTN1	Routine	DMTPOW	65, 68	yes
$RRTN1	Routine	DMTSML	59, 61	yes
$START	Routine	DMTNCM		yes
$START	Routine	DMTPOW	65	yes
$START	Routine	DMTSML	60	yes
$TPGET	Routine	DMTNCM		yes
$TPGET	Routine	DMTPOW	66	yes
$TPGET	Routine	DMTSML	60, 62	yes
$TPPUT	Routine	DMTNCM		yes
$TPPUT	Routine	DMTPOW	66	yes
$TPPUT	Routine	DMTSML	60	yes
$URTN1	Routine	DMTNCM		yes
$URTN1	Routine	DMTPOW	65, 70	yes
$URTN1	Routine	DMTSML	59, 63	yes
$USREXIT	Routine	DMTSML		yes
$WRTN1	Routine	DMTNCM		yes
$WRTN1	Routine	DMTPOW	65, 71	yes
$WRTN1	Routine	DMTSML	59	yes

Section 5: Data Areas

Data Area Aids

This section begins with a set of graphics to help with using some of the pointers in SVECTORS. Details of the data in the various tables, blocks, and queue entries are described later in this section.

MAINMAP LOCATION

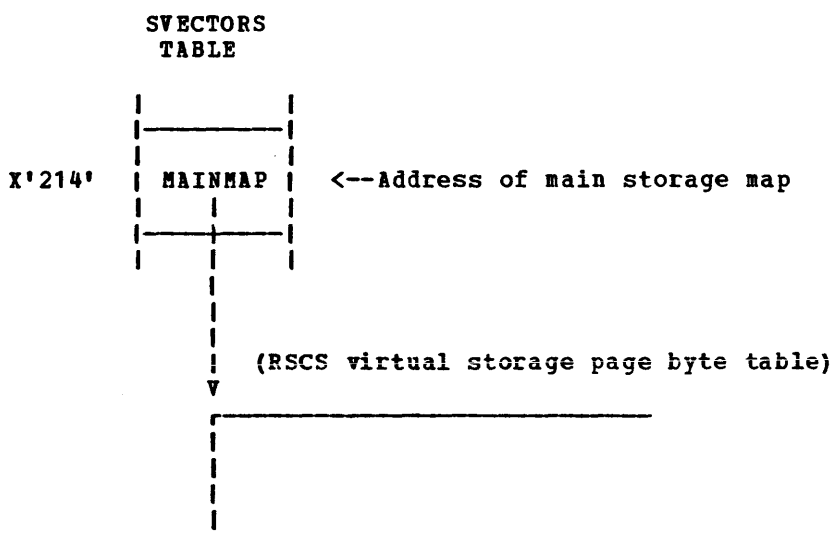


Figure 5-1. MAINMAP Location

The main storage map pointed to by MAINMAP (Figure 5-1) has a byte for each page (4K) of virtual storage of the virtual machine occupied by RSCS. When a page is in use (allocated to or occupied by RSCS processor), its MAINMAP byte contains the taskid (from the task element) of the task that is using the storage page. (This value is X'FF' if it is an MSUP page.) Every unused (available) storage page has a MAINMAP byte value of X'00'.

QUEUE ELEMENT STORAGE AREA AND FREEQ QUEUE

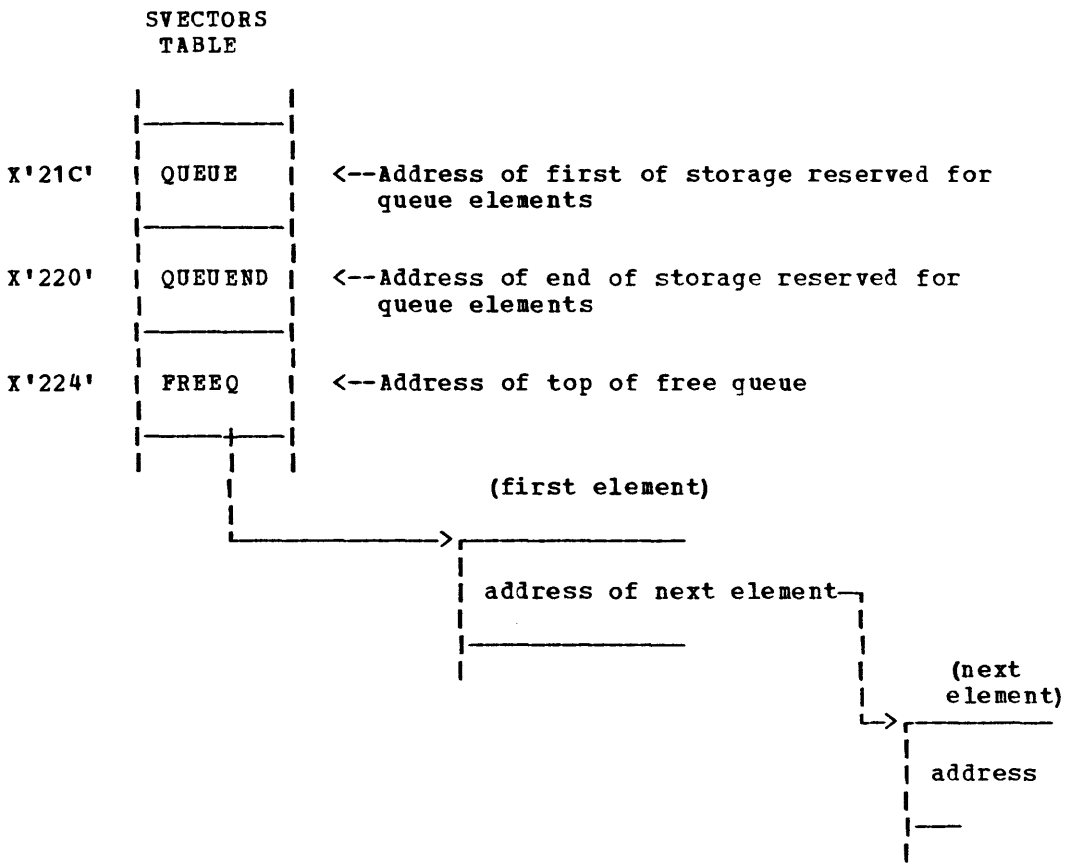


Figure 5-2. Queue Element Storage Area and FREEQ Queue

The DMTQRQ module obtains and frees 16-byte queue elements upon request from an RSCS task that either requires an element for, or is finished with an element in, a queue that the task maintains.

Because of the dynamic nature of queue element management, any given queue's elements are not restricted to a certain contiguous and sequential set of elements within the queue element storage area; individual elements that comprise a given queue may be anywhere within the storage area. To support this, each queue has a "top-of-queue" element, pointed to by an SVECTORS entry (Figure 5-2). This element has a pointer to the next element in the queue, and the chaining (pointer to address of next) continues through the remaining elements of that queue, scattered throughout the queue storage area.

The FREEQ queue is the set of unassigned queue elements at a given point in time.

Note that this discipline applies only to queues pointed to in SVECTORS, and not to tables, etc.

TASK QUEUE LOCATION

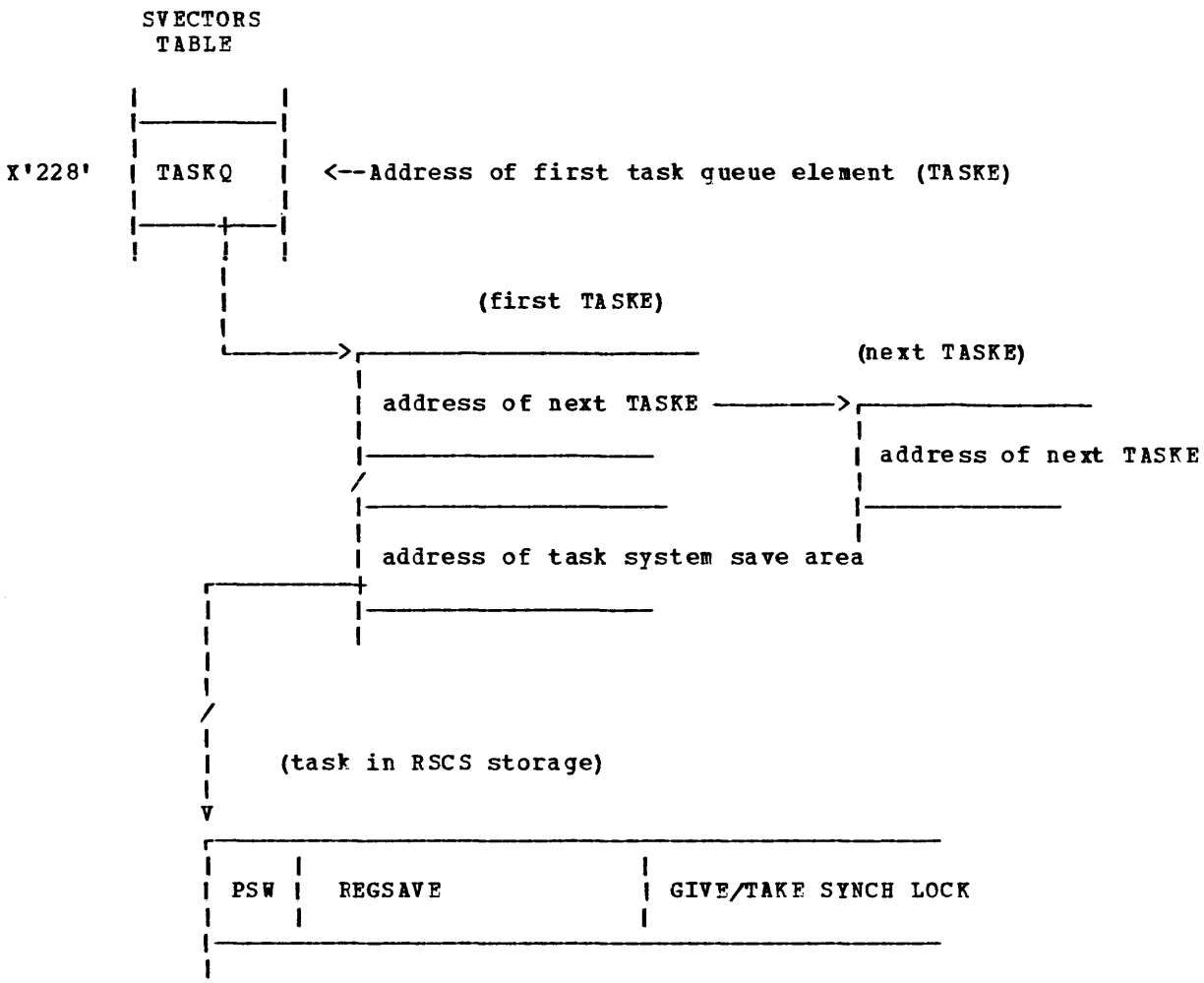


Figure 5-3. Task Queue Location

There is a task queue element for every active (started) task. This queue (Figure 5-3) is used by the RSCS MSUP dispatcher to start eligible tasks.

I/O QUEUE ORGANIZATION

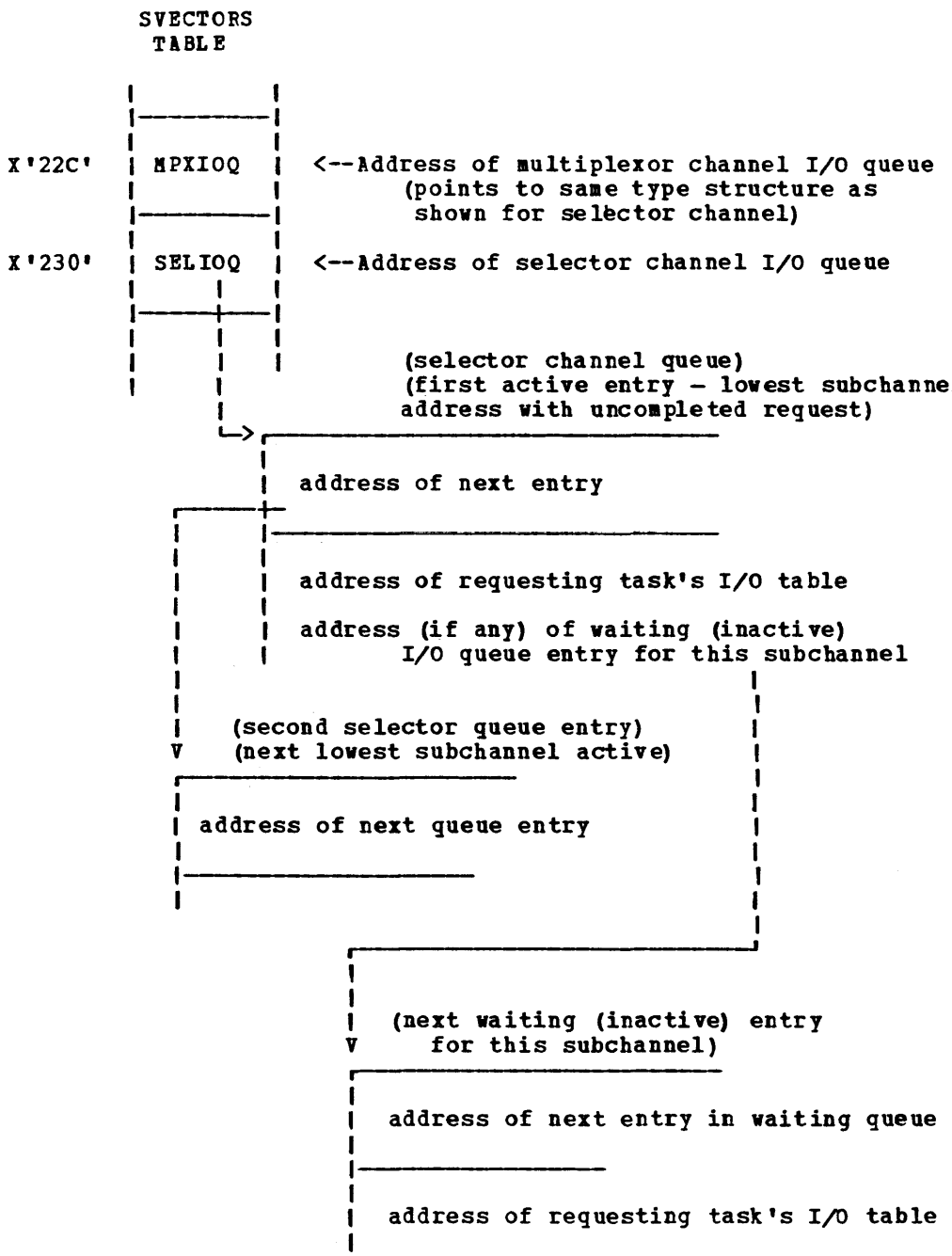


Figure 5-4. I/O Queue Organization

Each I/O queue (Figure 5-4) contains only entries that are in use. All unused entries are in the FREEQ. Requests for I/O queue additions and deletions, performed by DMTQRQ in MSUP come from the IODISMIS subroutine in the DMTIOM module in MSUP.

ASYNCHRONOUS INTERRUPT QUEUE POINTERS

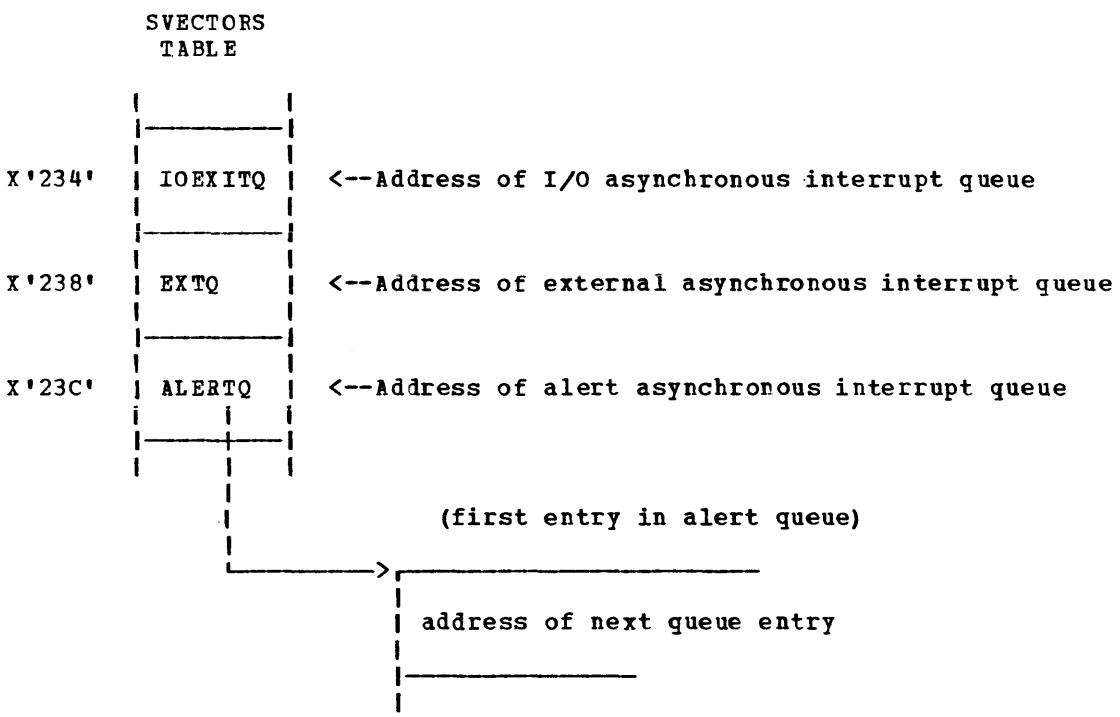


Figure 5-5. Asynchronous Interrupt Queue Pointers

Tasks that have exit routines to process certain types of asynchronous interrupts must build asynchronous interrupt queue entries for their exit routines beforehand, by asynchronous exit requests to DMTASY. The queue entry that DMTASY builds for each request reflects the type of exit condition specified in the request.

When any asynchronous interrupt condition occurs, the asynchronous interrupt handler scans the appropriate queue (Figure 5-5) to see which task (if any) has specified an exit routine to receive control immediately upon occurence of an asynchronous interrupt condition of this type.

A task's I/O exit routine is entered when its queue entry reflects the device address that has just generated an asynchronous interrupt.

A task's external exit routine is entered when its queue entry reflects the bit setting of the external interrupt that just occurred.

A task's alert exit routine is entered when it is the task that has just been specified in another task's alert request call to the DMTSIG routine in MSUP.

For the contents of the fields in the asynchronous exit queue elements, see "Asynchronous Exit Queue Elements."

GIVE ELEMENT QUEUE LOCATION

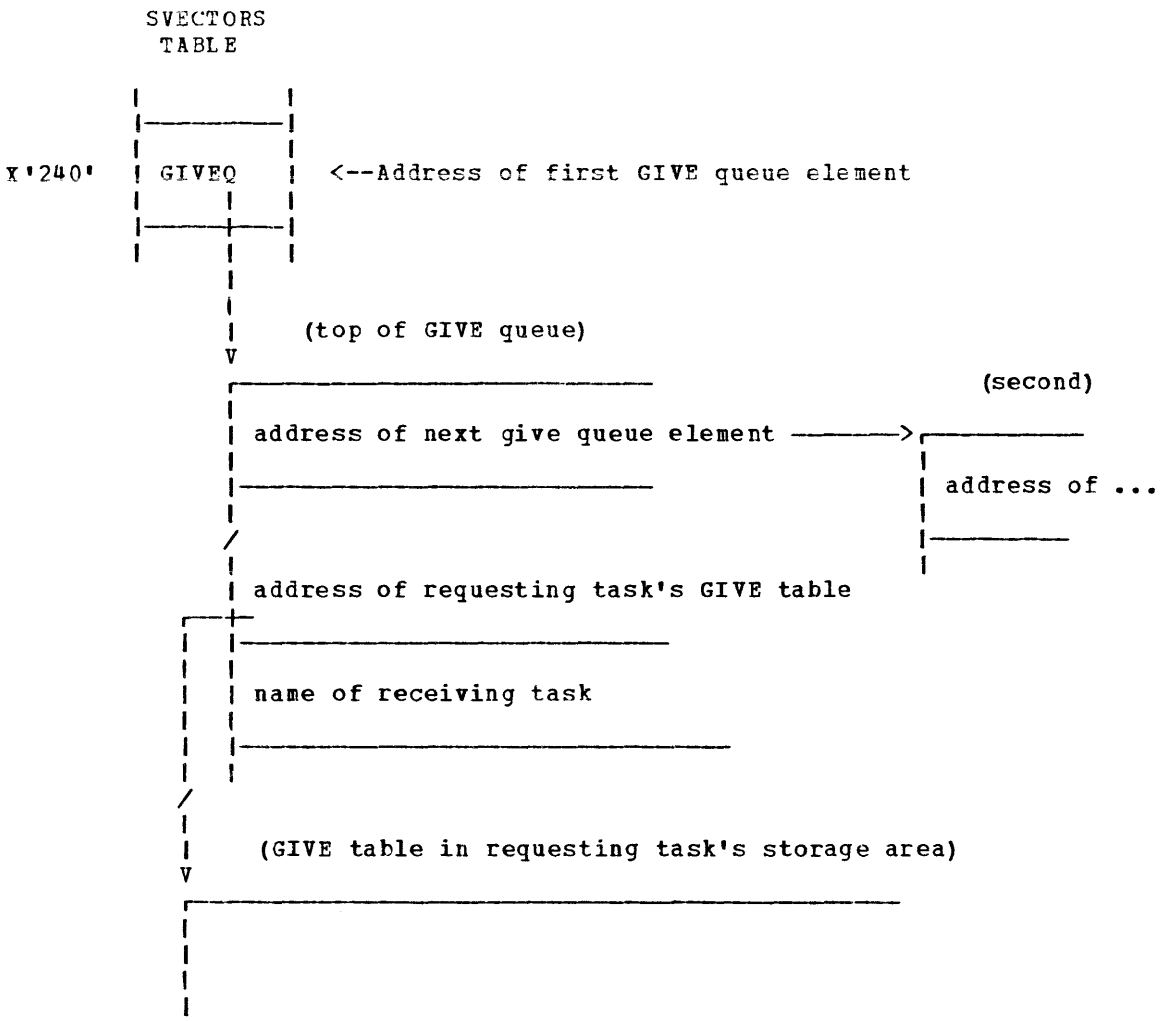


Figure 5-6. GIVE Element Queue Location

When the requesting task issues a GIVE request, the MSUP DMTGIV module builds a give element in the GIVE queue (Figure 5-6), posts the GIVE/TAKE synch lock in the requested task, and makes the requested task's TASKE entry dispatchable by a call to DMTPST.

LINK TABLE LOCATION

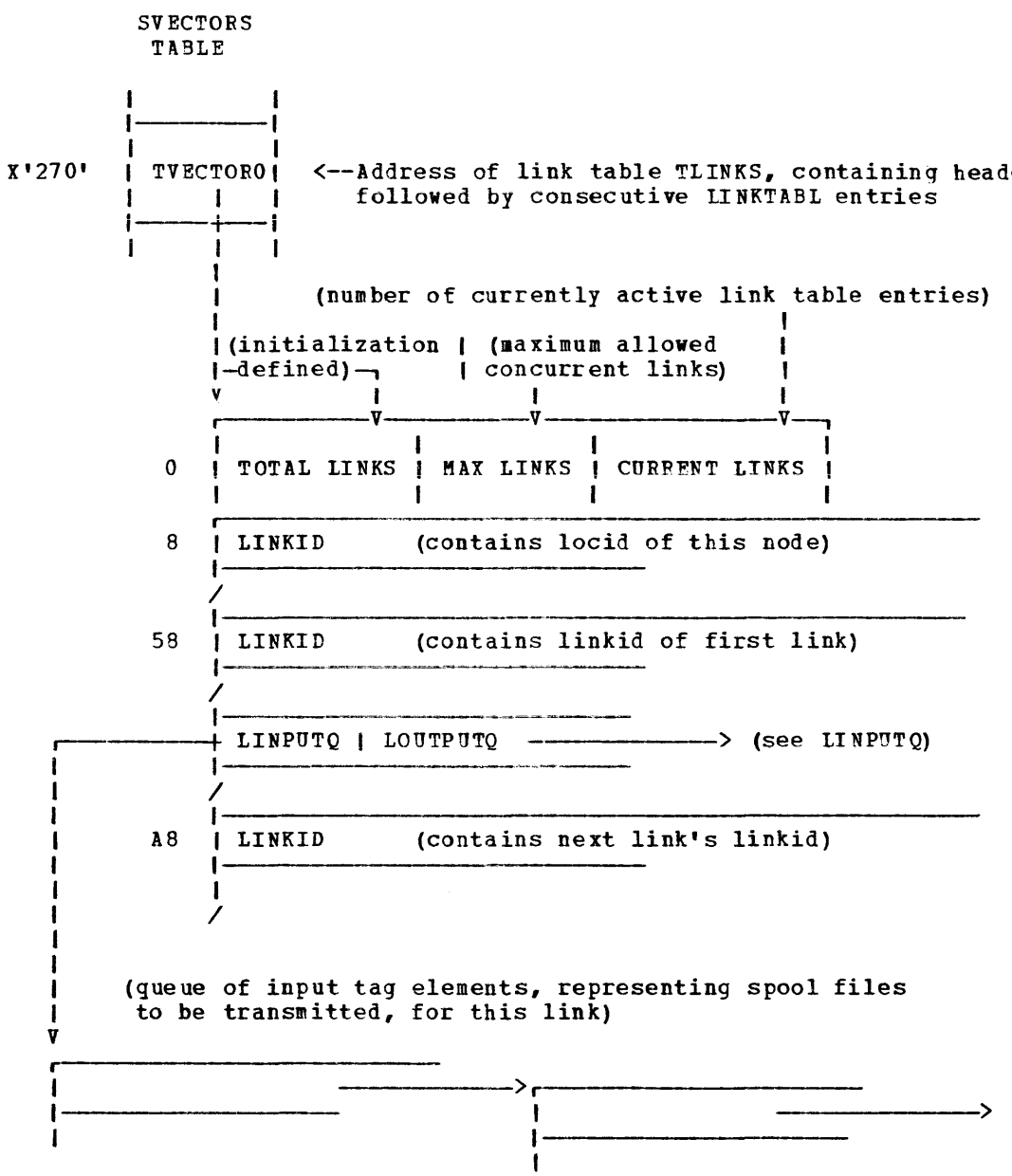


Figure 5-7. Link Table Location

There is an eight-byte information block just ahead of the first link table entry. The first entry itself is a special purpose entry, initialized with the VM/370 system locid, for reference by RSCS.

The LINKID field at the first of each LINKTABL block (Figure 5-7) identifies the link by the LOCID name of the adjacent node specified in the LINK statement in the dynamic directory, RSCS DIRECT, or specified by an operator DEFINE command. Each block contains data about one link (one port's line driver task).

ROUTE TABLE LOCATION

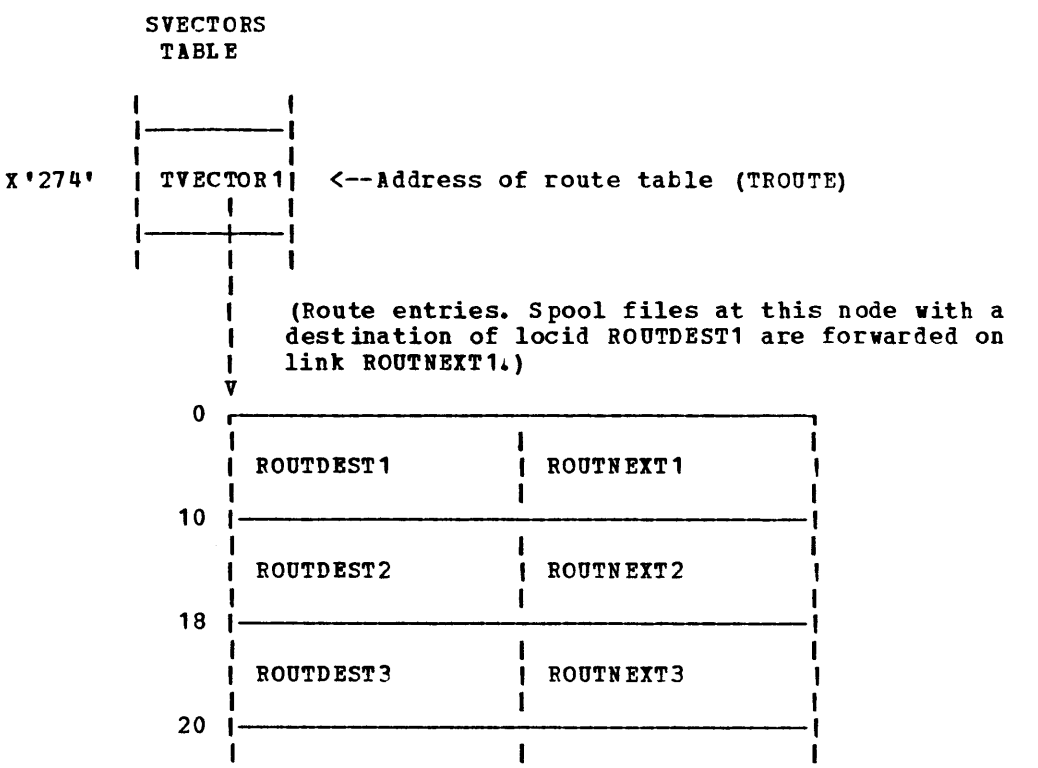


Figure 5-8. ROUTE Table Location

The route table entries (Figure 5-8) contain the routing information submitted in ROUTE statements in the dynamic directory (RSCS DIRECT) or specified by an RSCS operator ROUTE command.

SWITCHABLE PORTS (TPORTS) LOCATION

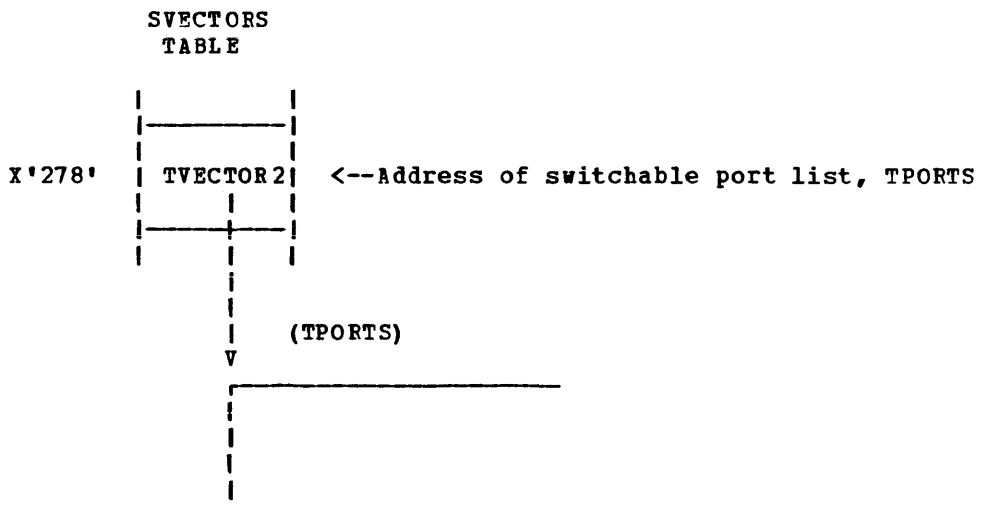


Figure 5-9. Switchable Ports (TPORTS) Location

The switchable port list (Figure 5-9) contains the information submitted in the RSCS dynamic directory (RSCS DIRECT) in PORT statements, or specified by the RSCS operator PORT command.

TAGSLOT QUEUE LOCATION

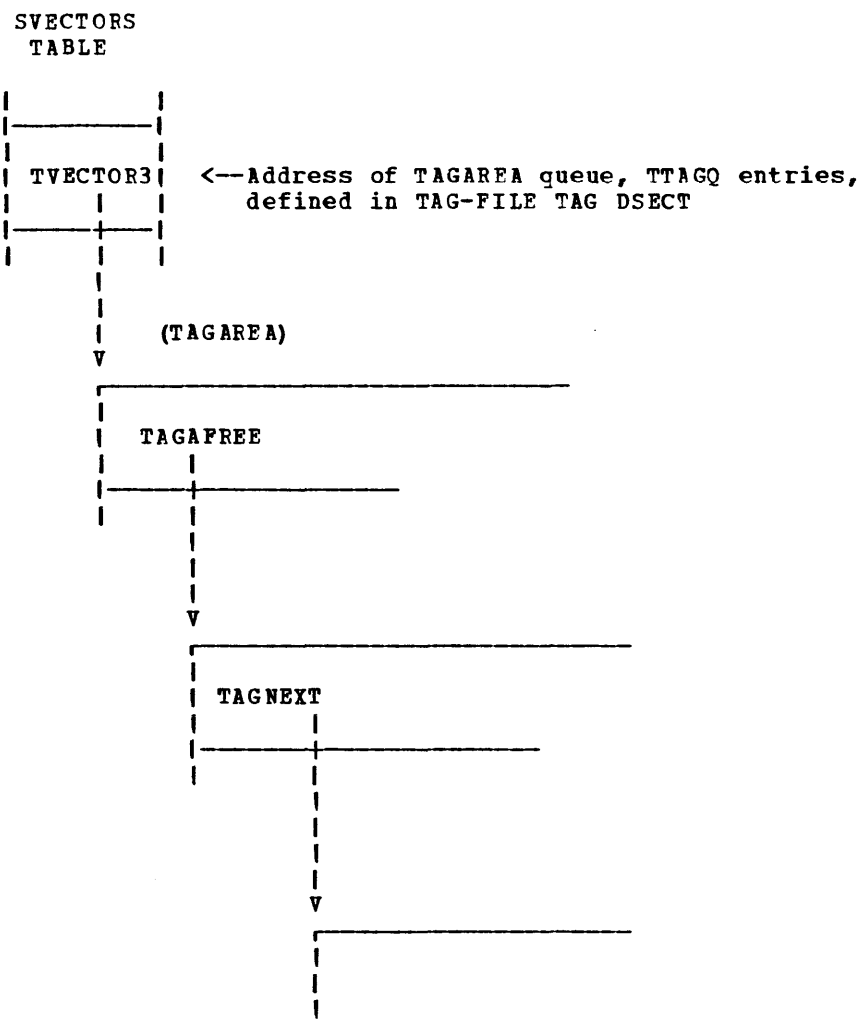


Figure 5-10. TAGSLOT Queue Location

TAGAREA is a queue (Figure 5-10) of free (available) tagslot elements. When they are in use, they are enqueued on the LINKTABLE of the link responsible for the spool file whose information is contained in the tagslot.

COMMON ROUTINE VECTOR TABLE (COMDSECT) ADDRESS

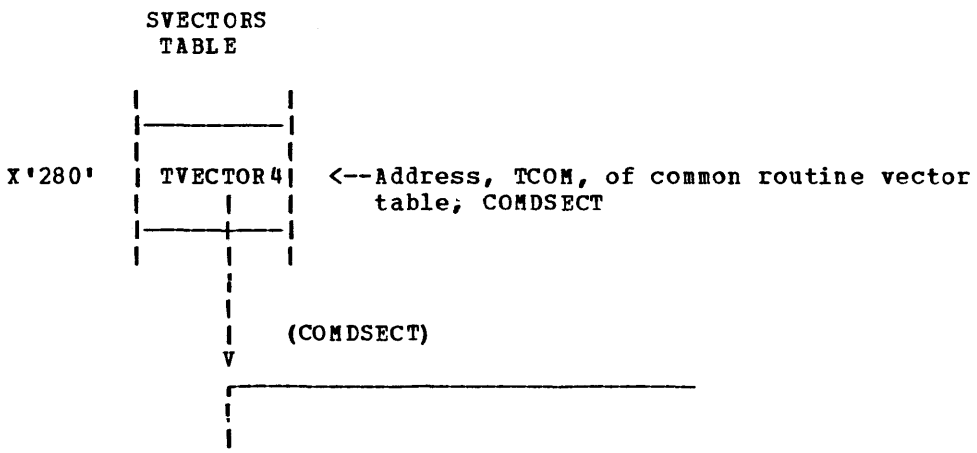


Figure 5-11. Common Routine Vector Table (COMDSECT) Address

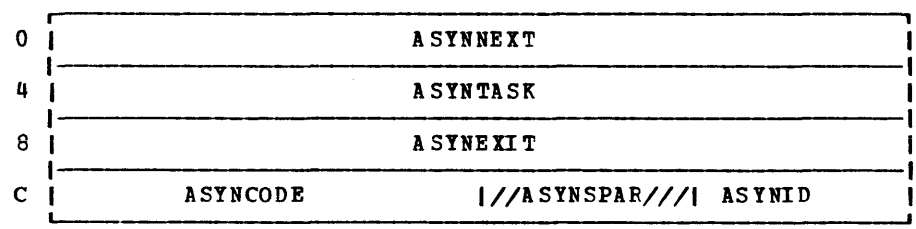
Data Areas and Control Blocks

This section describes in detail the primary data areas and control blocks used by RSCS. Offsets are shown in hexadecimal notation at the right of each diagram. Following each diagram is a table that presents the hexadecimal offset, name, type, and description of each field.

ASYNCHRONOUS EXIT QUEUE ELEMENT: ASYNE

ASYNE defines symbolic addresses for elements on an asynchronous exit queue. An asynchronous exit queue element contains information by which a task requests that it handle asynchronous interrupts.

IOEXITQ, EXTQ, and ALERTQ in SVECTORS are the heads of three asynchronous exit queues. Each of these queues comprises supervisor elements defined by the ASYNE DSECT. IOEXITQ points to requests for I/O exits, EXTQ points to requests for external exit requests, and ALERTQ points to requests for ALERT exits.



0	ASYNNEXT	DS	1F	Address of the next asynchronous interrupt exit request element
4	ASYNTASK	DS	1F	Address of task element describing the task that requested the asynchronous interrupt
8	ASYNEXIT	DS	1F	Address of the requested asynchronous exit routine
C	ASYNCODE	DS	AL2	Address of the device for which asynchronous I/O interrupts are requested or interrupt bit code
E	ASYNSPAR	DS	1X	Reserved for IBM use
F	ASYNID	DS	1X	1-byte ID of the task owning the asynchronous exit routine

CMS FILE ACCESS WORK AREA

CMSACCES DSECT

When entering either of the read only CMS file access routines (CMSOPEN and CMSGET, pointed to in COMDSECT), R13 must point to a work area of this format.

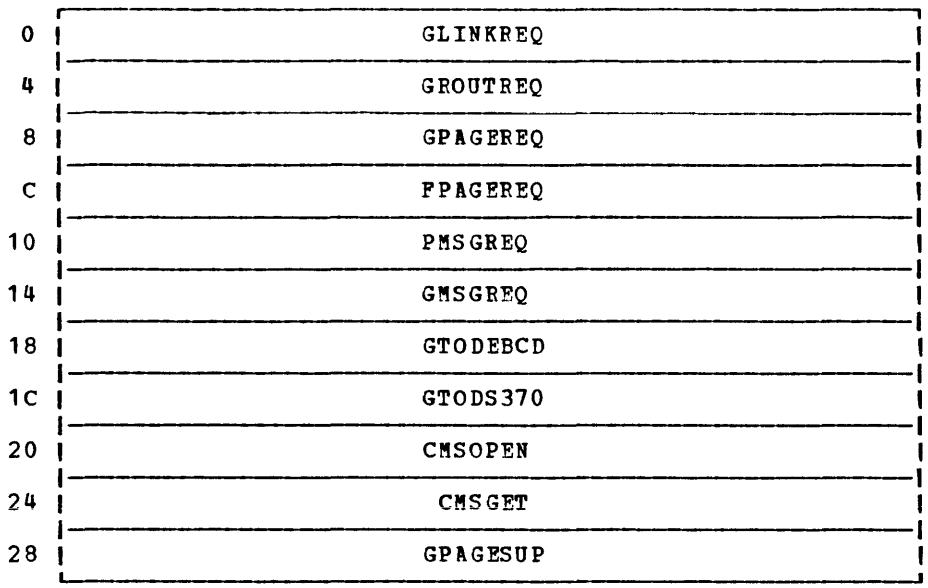
```
OPENFILE DC   0F'0',CL8' ',CL8'TEXT'  File name for OPEN request

STATABLE DC   0F'0'             CMS file status table
FILENAME DC   8C' '             CMS filename
FILETYPE DC   8C' '             CMS filename
FILEDATE DC   F'0'              Creation date - decimal mmddhhmm
FILEWNUM DC   H'0'              Write item number
FILERNUM DC   H'0'              Read item number
FILEMODE DC   2C' '             CMS filemode
FILEINUM DC   H'0'              Number of items in entire file
FILELINK DC   H'0'              CMS block number of first chain link
FILEFORM DC   C' '              File format:
                                   C'F' for fixed; C'V' for variable
FILEFLAG DC   X'00'             File flags - always zero?
FILEILEN DC   F'0'              (Maximum) length of file data items
FILESIZE DC   H'0'              Number of 800-byte blocks in file
FILEYEAR DC   2C' '             Year of file creation (last two digits
                                   in EBCDIC)
STATLEN  EQU  *-STATABLE        Length of CMS file status table

FSTFOP   DS   F                 Alt. file origin pointer
FSTADBC  DS   F                 Alt. number of data blocks
FSTAIC   DS   F                 Alt. item count
FSTNLVL  DS   XL1               Number of pointer block levels
FSTPTRSZ DS   XL1               Length of a pointer element
FSTADATI DS   CL6               Alt. date and time (yymmddhhmmss)
         DS   F                 Reserved
FSTDSIZE EQU  (*-STATABLE)      File status table size in bytes

* CMS Disk Access Control Area

DASD     DC   0F'0'             CMS DASD I/O request table
         DC   F'0'              Synch lock
         DC   AL2(0)            DASD device address (set by INIT)
         DC   AL1(24)           Sense information required for DASD
         DC   X'00'             Device type code (set by INIT)
         DC   A(0)              Address of disk read channel program
         DC   2F'0'             Return SIO condition code and end CSW
         DC   24X'00'           Return sense information on unit check

* The values below are set by DMTINI according to the type and
* format of the RSCS system disk:

PERCYL   DC   F'0'              CMS records per cylinder
PERTRACK DC   F'0'              CMS records per track or pair
OVERNUM  DC   F'0'              Record number of overlapper
DASDSECS DC   20X'00'           Record number - sector table
DISKCLAS DC   X'00'             System disk class
         DS   3X                Reserved
DISKBSZE DS   F                 System disk blocksize
         DS   F                 Reserved
*
DASDREAD CCW  X'07',BBCCHHR,CC+SILI,6    Seek (bbcchhr)
DASETSEC CCW  X'03',SECTOR,CC+SILI,1     NOP or set sector
DASEARCH CCW  X'31',BBCCHHR+2,CC+SILI,5  Search ID equal (cchhr)
         CCW  X'08',DASEARCH,X'00',1     Back to search if unequal
```

```
DARDDATA CCW   X'86',0,SILI,800          Read data multi-track
BBCCHHR  DC    0F'0',7X'00'              DASD address for reference
SECTOR   DC    X'00'                     Sector number of record above

* Channel Program for Fixed Block Architecture Devices

CFBAREAD DC    0D'0'
FBACCWD2 CCW   X'63',FBAD2A,CC+SILI,16   Define extent
         CCW   X'43',FBAL2A,CC+SILI,8    Define extent
FBACCWX2 CCW   X'42',0,SILI,80           Define extent
FBAD2A   DS    X'40'                     Mask (Inhibit Write)
         DS    X'000000'                 Reserved
         DS    F'0'                      Extent offset
         DS    F'0'                      First block offset
FBAD2ALB DS    F'1'                      Last block offset

         DS    0F                      Locate list
FBAL2A   DS    X'06'                   Operation (Read)
         DS    X'00'                   Aux byte
FBAL2ANB DS    H'1'                    Number of blocks
FBAL2ABO DS    F'1'                    Block offset
VOLCCWS  DS    0D
         CCW   X'07',0,CC+SILI,6
         CCW   X'31',0,CC+SILI,5
         CCW   X'08',*-8,0,0
VOLCCW1  CCW   X'06',0,SILI,80

         DC    0D'0'
NEXTLINK DC    A(0)              Address of pointer to next chain link
LINKEND  DC    A(LINKLIST+80)    End of chain link list
LINKLIST DS    80C               List of CMS chain link block numbers
EDFOVBUF EQU   LINKLIST          This 80-byte area is used for an
                                    overflow buffer when the system disk
                                    is EDF format

         DC    0D'0'
NEXTBLOK DC    A(0)              Address of pointer to next data block
BLOKEND  DC    A(BLOKLIST+120)   End of current data block list
BLOKLIST DS    800C              List of CMS file data block numbers

         DC    0D'0'
NEXTITEM DC    A(0)              Address of next (unread) data item
ITEMEND  DC    A(FILCHBUF+800)   End of FILCH data buffer
FILCHBUF DS    800C              Buffer for CMS block FILCH routine

CFILSAVE DS    8F                Save area for CMS FILCH routine
COPNSAVE DS    9F                Save area FOR CMS OPEN routine
CGETSAVE DS    9F                Save area FOR CMS GET routine

ENXTITEM DS    F'0'              Absolute value of next (unread) data item
MVLEN    DS    F'0'              Move length when record spans two blocks
NXTRECPT DS    F'0'              Address of next record in buffer
DABLKCNT DS    F'0'              Number of blocks processed

APTRBLK  DS    A                 Address of EDF pointer block
AEDFBUF  DS    A                 Address of EDF data block

         DS    0F                Allign
ADTIDENT DS    CL4               Volume/label identifier
ADTID    DS    CL6               Volume start / volume identifier
ADTVER   DS    CL2               Version level
ADTDBSIZ DS    F                 Disk block size
ADTDOP   DS    F                 Disk origin pointer
ADTCYL   DS    F                 Number of formatted cylinders on disk
ADTMCYL  DS    F                 Maximum number of formatted cylinders
```

```
                                          on disk
ADTNUM   DS   F                 Disk size in blocks
ADTUSED  DS   F                 Number of disk blocks in use
ADTFSTSZ DS   F                 Size of file status table
ADTNFST  DS   F                 Number of file status tables per block
ADTCRED  DS   CL6               Disk creation date (yymmddhhmmss)
         DS   CL30              Reserved
ADTLABSZ EQU  *-ADTIDENT        Length of label
ADTDKFOR DS   F
ADTDVTYP F

CMSACCL  EQU  ((*-CMSACCES)+7)/8  Number of double words in CMS Access
                                    area
```

COMDSECT TABLE CONTENTS

The COMDSECT table contains pointers to common supervisor routines.

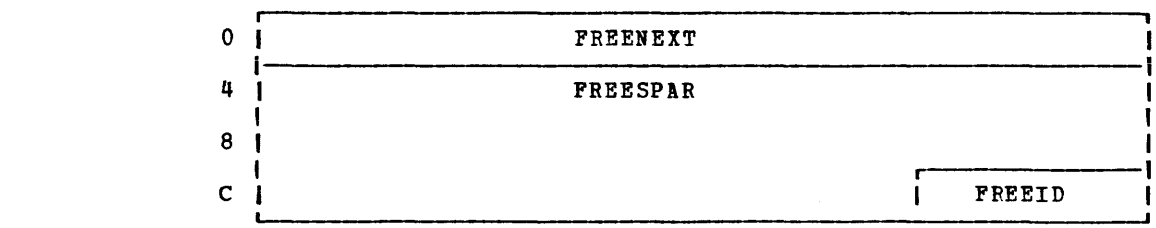



```
  0     GLINKREQ  DS  1A    Get link table entry routine
  4     GROUTREQ  DS  1A    Get routing table entry routine
  8     GPAGEREQ  DS  1A    Get page of main storage
  C     FPAGEREQ  DS  1A    Free page of main storage
 10     PMSGREQ   DS  1A    Put message element into message stack
 14     GMSGREQ   DS  1A    Remove message element from message stack
 18     GTODEBCD  DS  1A    Convert S/370 TOD* to EBCDIC TOD
 1C     GTODS370  DS  1A    Convert EBCDIC TOD to S/370 TOD
 20     CMSOPEN   DS  1A    Open CMS file**
 24     CMSGET    DS  1A    Read next record of CMS file**
 28     GPAGESUP  DS  1A    Allocate a page of virtual storage for
                              supervisor use

                            * Time-Of-Day
                           ** See CMS File Access Work Area
```

FREE QUEUE ELEMENT: FREEE

FREEE defines an element in the chain of elements that comprise the free element queue.

FREEQ in SVECTORS points to the chain of free elements, each of which is defined by the FREEE DSECT.



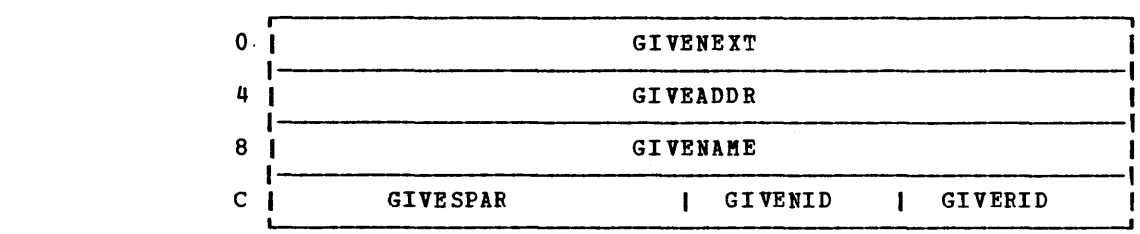
0	FREENEXT	DS	1F	Address of next element in free queue
4	FREESPAR	DS	CL11	Spare field
F	FREEID	DS	1X	Standard taskid offset (X'00' denotes free element)

GIVE QUEUE ELEMENT: GIVEE

GIVEE defines symbolic addresses for items used in processing a GIVE request.

GIVEQ in SVECTORS points to the queue of GIVE elements used in task-to-task communications.

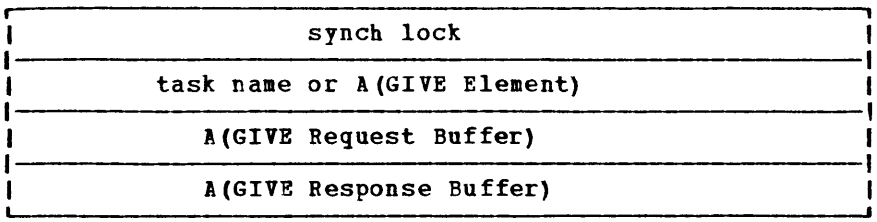
The GIVEADDR field of this DSECT is the address of a GIVE request table, which, in turn, contains addresses of buffers for elements describing requests and responses to requests. These tables are described below; the elements that fill the buffers are described in "Request Elements".



0	GIVENEXT	DS	1F	Address of next GIVE element
4	GIVEADDR	DS	1F	Address of GIVE request table in sending task's storage
8	GIVENAME	DS	CL4	Task name of receiving task
C	GIVESPAR	DS	AL2	Unused
E	GIVENID	DS	1X	1-byte ID or receiving task after TAKE
F	GIVERID	DS	1X	1-byte ID or sending task

GIVE REQUEST TABLE IN GIVE/TAKE REQUESTING TASK

The format of a GIVE Request Table is:



When a task requests the services of another task via a GIVE request, the second field of the table above contains the task name of the task to which the task is to be sent. When DMTGIV builds a GIVE element for the request, it overlays this task name with the address of the GIVE element.

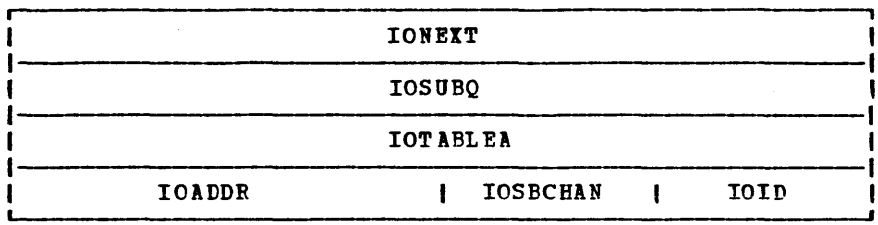
The task performing the requested service builds a table called the TAKE request table, which corresponds to the GIVE request table.

I/O REQUEST QUEUE ELEMENT: IOE

IOE defines symbolic addresses of elements and other information associated with an I/O operation requested by a task.

MPXIOQ and SELIOQ in SVECTORS point to queues of I/O elements for the multiplexer and selector channels, respectively.

The IOTABLEA field points to the address of an I/O table defined by IOTABLE, which is described in this section.



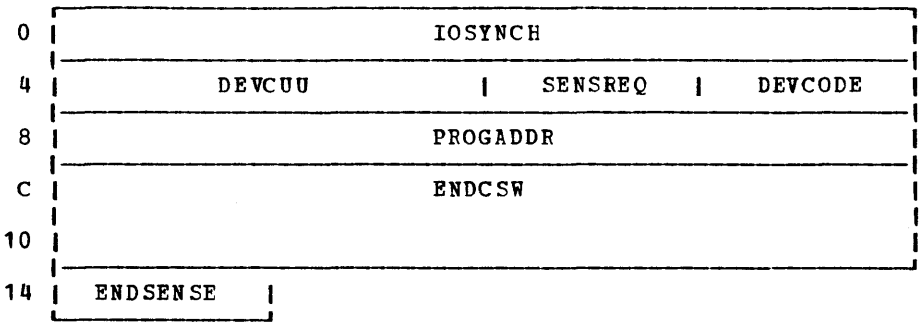
0	IONEXT	DS	1F	Address of next active I/O element
4	IOSUBQ	DS	1F	Address of next inactive I/O element for a given subchannel
	IOSTAT	EQU	*	Status flags for current I/O operation (First byte of IOTABLEA)

Bits defined in IOSTAT

	SENSING	EQU	X'80'	Flag set to 1 while automatic sense is active
	CHANDONE	EQU	X'40'	Flag set to 1 when subchannel terminates
8	IOTABLEA	DS	1F	Address of I/O request table in task storage
C	IOADDR	DS	AL2	Address (cuu) of the device requesting current I/O operation
E	IOSBCHAN	DS	1X	Subchannel address; 1-byte; assigned by MSUP
F	IOID	DS	1X	ID of task associated with this I/O operation; 1-byte; assigned by MSUP

I/O REQUEST TABLE IN REQUESTING TASK: IOTABLE

The I/O request table contains data used in processing an I/O request. The first five fields are filled in by the task to convey information about the I/O request to the supervisor. The last three fields are filled in by the supervisor to convey status information about the I/O operation to the task.

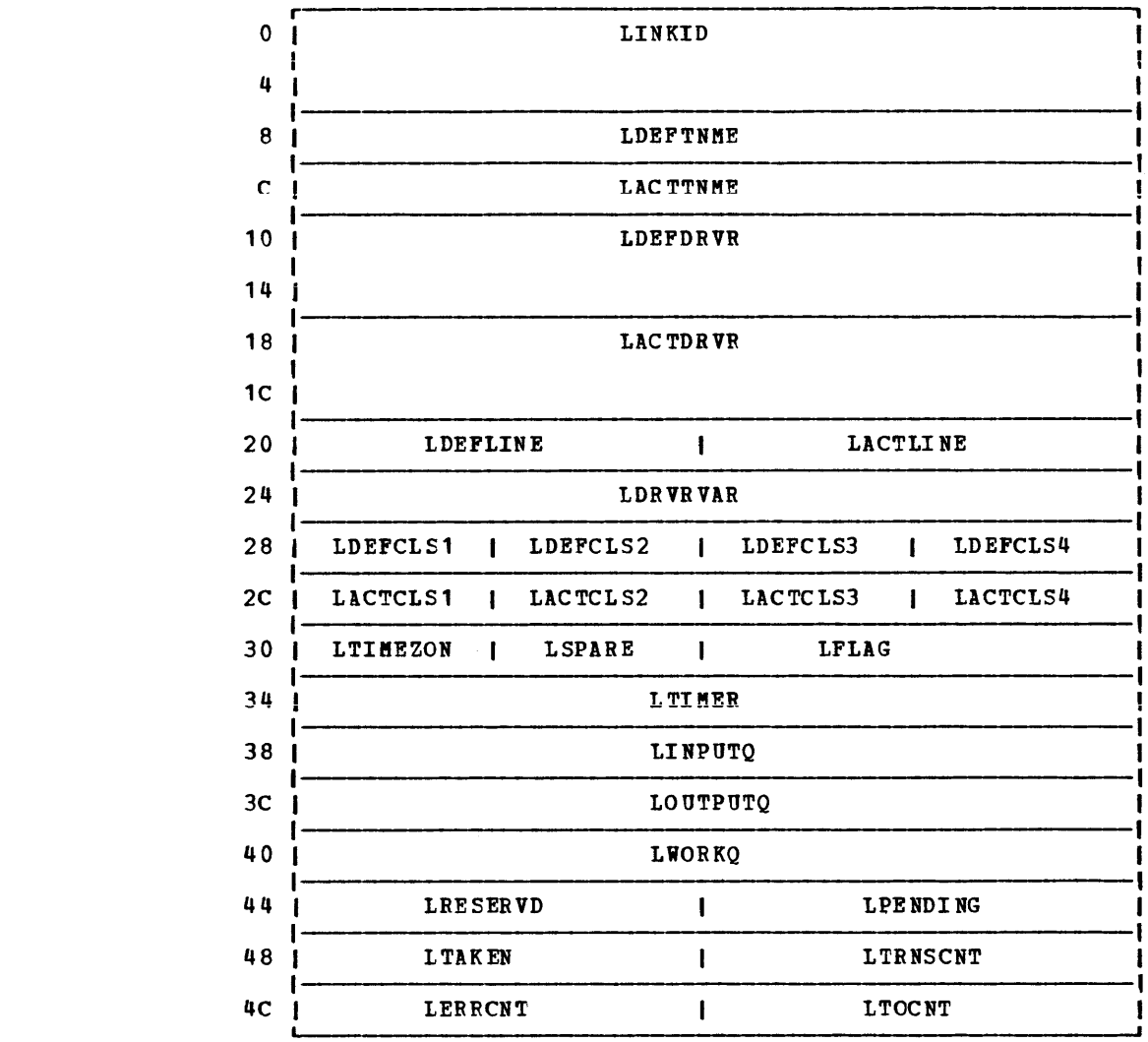



```
0     IOSYNCH  DS   1F      Synchronization lock for I/O operation
4     DEVCUU   DS   AL2     Address (cuu) of device associated with this
                            I/O operation
6     SENSREQ  DS   AL1     Number of sense bytes requested on unit check
7     DEVCODE  DS   AL1     1-byte VM/370 device type code (not used by
                            I/O manager)
8     PROGADDR DS   1F      Address of channel program for the I/O
                            operation

      SIOCOND  EQU  *       1-byte SIO condition code return information

C     ENDCSW   DS   2F      Ending CSW with composite status return
                            information
14    ENDSENSE DS   AL1     Requested return sense information on unit
                            check CSW status

      TYPPUN   EQU  X'80'   VM/370 type code for the punch
      TYPPRT   EQU  X'40'   VM/370 type code for the printer
```

LINK TABLE ENTRY: LINKTABL

LINKTABL describes the status of a single link in the network; collectively, all the links defined for the system are referred to as the link table.

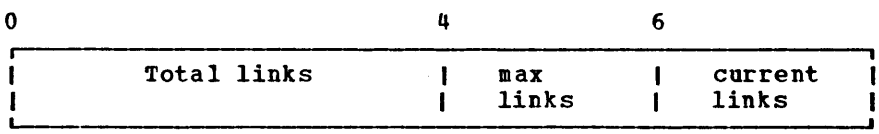
The first link table entry has the locid of this RSCS node in the linkid field. Normal link definitions start in the second link table entry.



0	LINKID	DS	CL8	EBCDIC linkid
8	LDEFTNME	DS	CL4	Default task name
C	LACTTNME	DS	CL4	Active task name
10	LDEFDRVR	DS	CL8	Default driver id
18	LACTDRVR	DS	CL8	Active driver id
20	LDEFLINE	DS	2X	Default virtual line address
22	LACTLINE	DS	2X	Active virtual line address
24	LDRVRVAR	DS	1F	Line driver variable information
28	LDEFCLS1	DS	CL1	Default spool file class 1
29	LDEFCLS2	DS	CL1	Default spool file class 2
2A	LDEFCLS3	DS	CL1	Default spool file class 3
2B	LDEFCLS4	DS	CL1	Default spool file class 4
2C	LACTCLS1	DS	CL1	Active spool file class 1

```
2D      LACTCLS2  DS    CL1     Active spool file class 2
2E      LACTCLS3  DS    CL1     Active spool file class 3
2F      LACTCLS4  DS    CL1     Active spool file class 4
30      LTIMEZON  DS    1X      Time zone displacement west from GMT
31      LSPARE    DS    1X      Spare byte
32      LFLAG     DS    2X      Link table status flag bytes

      Bits defined in LFLAG

        LACTIVE   EQU   X'80'   Link active (line driver task loaded)
        LALERT    EQU   X'40'   AXS ALERT exit set
        LHOLD     EQU   X'20'   Link hold set
        LDRAIN    EQU   X'10'   Link drain in progress
        LCONNECT  EQU   X'08'   Link connected
        LTIMERON  EQU   X'02'   Timer ALERT request outstanding
        LHALT     EQU   X'01'   Link to be forced inactive

34      LTIMER    DS    1F      Active task timer value
38      LINPUTQ   DS    1F      Input file tag queue address
3C      LOUTPUTQ  DS    1F      Output file tag queue address
40      LWORKQ    DS    1F      General string stack address
44      LRESERVD  DS    1H      Count of tag elements reserved
46      LPENDING  DS    1H      Count of unaccepted tags
48      LTAKEN    DS    8X      Count of tag slots in use
4A      LTRNSCNT  DS    1H      Link transaction count
4C      LERRCNT   DS    1H      Error Count
4E      LTOCNT    DS    1H      Timeout count
        LINKLEN   EQU   *-LINKTABL  Length of link table entry
```

An 8-byte header precedes the first entry in the link table (that is, the first link defined by the LINKTABL DSECT). The TLINKS field (TVECTOR0) in SVECTORS points to this header:



where:

total links — is the total number of link table entries generated during RSCS initialization.

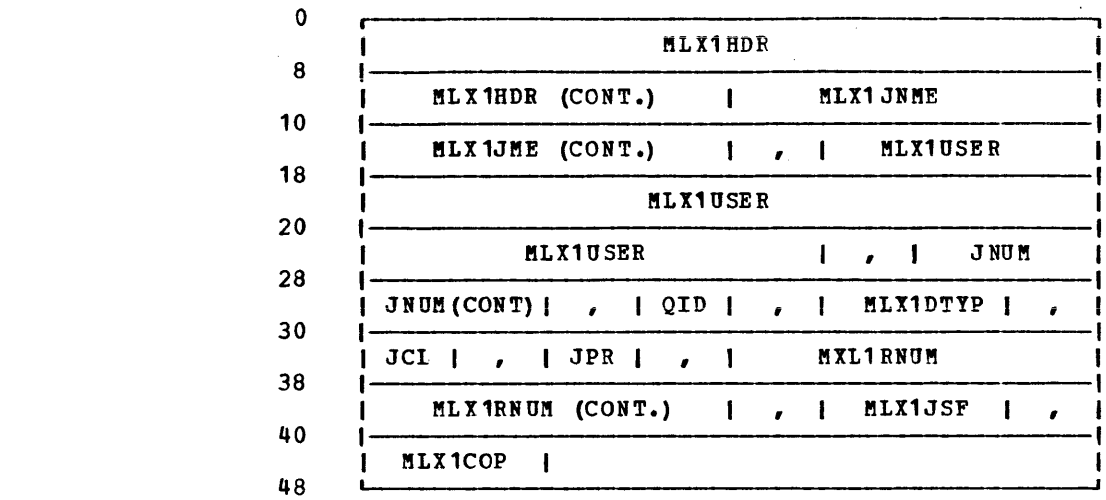
max links — is the maximum number of concurrently active links allowable.

current links — is the number of links active in RSCS at a given time.

MLX RECORDS

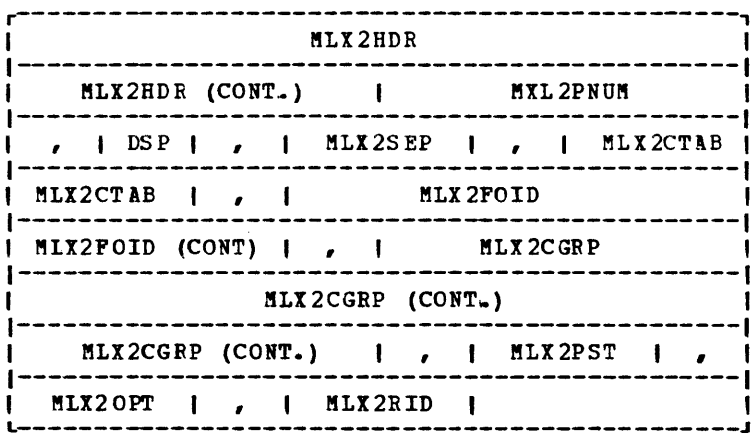
The MLX records are created by VSE/POWER. These three records describe various characteristics of the output file.

MLX Record 1:



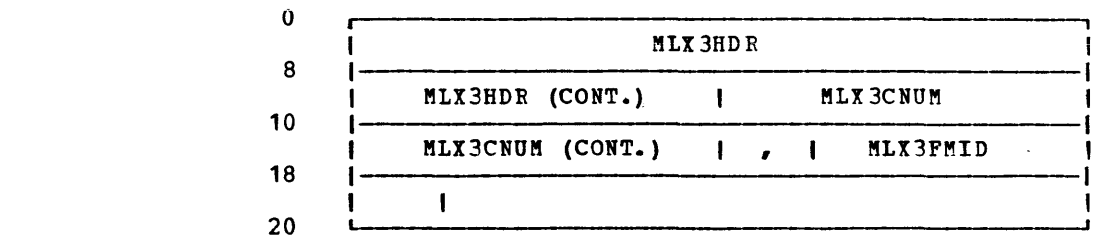

```
 0  MLX1HDR  DS    CL12           MLX RECORD 1 HEADER:    * $$ MLX Q1=
 C  MLX1JNME DS    CL8            JOB NAME
14          DS    CL1            ,DELIMITER
15  MLX1USER DS    CL16           USER INFORMATION
25          DS    CL1            ,DELIMITER
26  MLX1JNUM DS    XL4            JOB NUMBER
2A          DS    CL1            ,DELIMITER
2B  MLX1QID  DS    C              QUEUE RECORD ID
2C          DS    CL1            ,DELIMITER
2D  MLX1DTYP DS    XL2            DEVICE TYPE OR LINE ID
2F          DS    CL1            ,DELIMITER
30  MLX1JCL  DS    CL1            JOB CLASS
31          DS    CL1            ,DELIMITER
32  MLX1JPR  DS    CL1            JOB PRIORITY
33          DS    CL1            ,DELIMITER
34  MLX1RNUM DS    XL8            RECORD COUNT
3C          DS    CL1            ,DELIMITER
3D  MLX1JSF  DS    XL2            JOB SUFFIX NUMBER
3F          DS    CL1            ,DELIMITER
40  MLX1COP  DS    XL2            NUMBER OF COPIES
    MLX1LEN  EQU   *-MLX1REC      LENGTH OF MLX RECORD 1
```

MLX Record 2:



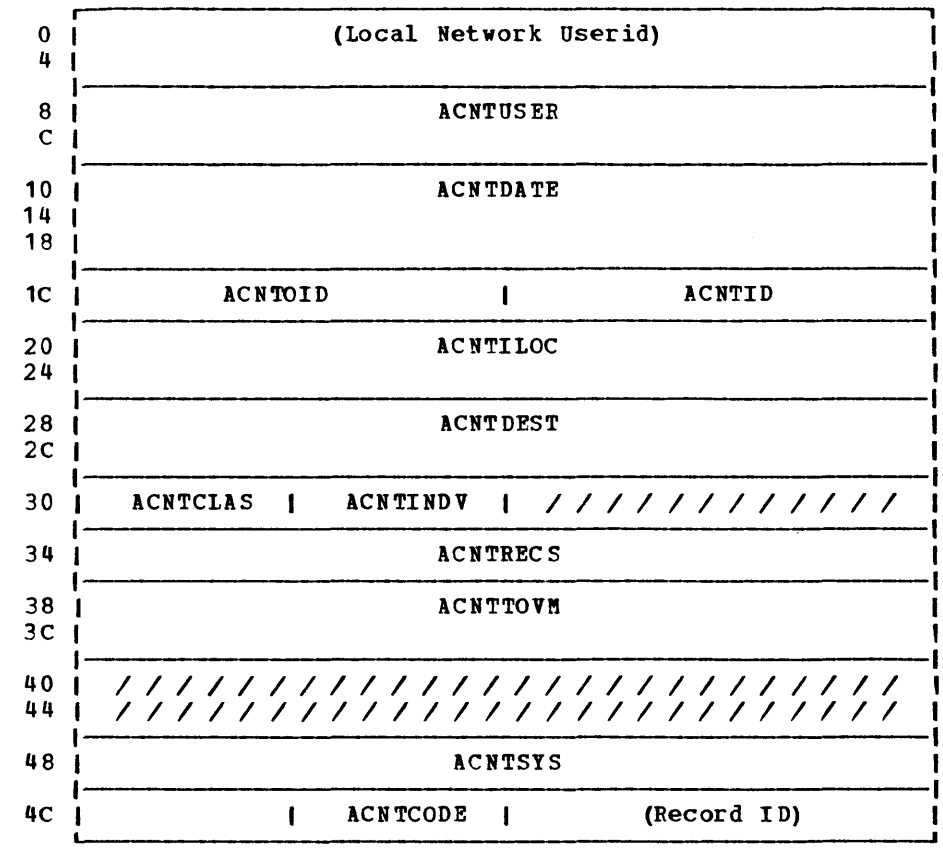

```
 0  MLX2HDR  DS    CL12           MLX RECORD 2 HEADER:   * $$ MLX Q2=
 C  MLX2PNUM DS    XL4            NUMBER OF PAGES
10            DS    CL1            ,DELIMITER
11  MLX2DSP  DS    CL1            DISPOSITION
12            DS    CL1            ,DELIMITER
13  MLX2SEP  DS    XL2            NUMBER OF SEPARATORS
15            DS    CL1            ,DELIMITER
16  MLX2CTAB DS    CL4            COMPACTION TABLE NAME
1A            DS    CL1            ,DELIMITER
1B  MLX2FOID DS    XL8            FORMS OVERLAY ID - 3800
23            DS    CL1            ,DELIMITER
24  MLX2CGRP DS    XL16           COPY GROUPS - 3800
34            DS    CL1            ,DELIMITER
35  MLX2PST  DS    XL2            PAPER STATUS - 3800
37            DS    CL1            ,DELIMITER
38  MLX2OPT  DS    XL2            OPTION BYTE - 3800
40            DS    CL1            ,DELIMITER
41  MLX2RID  DS    XL2            ORIGIN REMOTE ID FOR LIST/PUNCH
42  MLX2LEN  EQU   *-MLX2REC      LENGTH OF MLX RECORD 2
```

MLX Record 3:




```
 0  MLX3HDR  DS    CL12          MLX RECORD 3 HEADER:    * $$ MLX Q3=
 C  MLX3CNUM DS    XL8           LINE OR CARD COUNT
13          DS    CL1           ,  DELIMITER
14  MLX3FMID DS    CL4           FORMS IDENTIFIER
    MLX3LEN  EQU   *-MLX3REC     LENGTH OF MLX RECORD 3
```

NETWORK ACCOUNTING CARD FORMAT



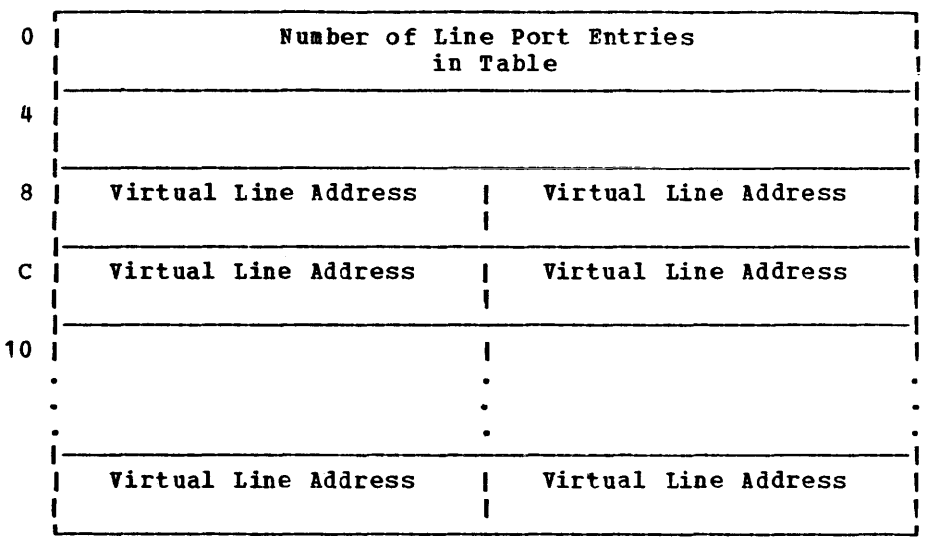
0		DS	CL8	1-8	Local network USERID fixed by CP
8	ACNTUSER	DS	CL8	9-16	Originating location USERID
10	ACNTDATA	DS	CL12	17-28	Date and time of record (MMDDYYHHMMSS)
1C	ACNTOID	DS	CL2	29-30	Origin spool file ID
1E	ACNTID	DS	CL2	31-31	Local spool file ID
20	ACNTILOC	DS	CL8	33-40	Originating location ID
28	ACNTDEST	DS	CL8	41-48	Destination location ID
39	ACNTCLAS	DS	CL1	49	Class
31	ACNTINDV	DS	CL1	50	Origin device type ('8N'=PUN/'4N'=PRT)
32		DS	CL2	51-52	Filler
34	ACNTRECS	DS	CL4	53-56	Number of records in file
38	ACNTTOVM	DS	CL8	57-64	Destination location USERID
40		DS	CL8	65-72	Filler
48	ACNTSYS	DS	CL5	73-77	System ID (Serial + Model)
4D	ACNTCODE	DS	CL1	78	Transmission code (01=SEND/02=RECV)
4E		DS	CL2	79-80	Record identifier ('C0') fixed by CP

PORT TABLE

BUILT BY: DMTIRX at RSCS initialization

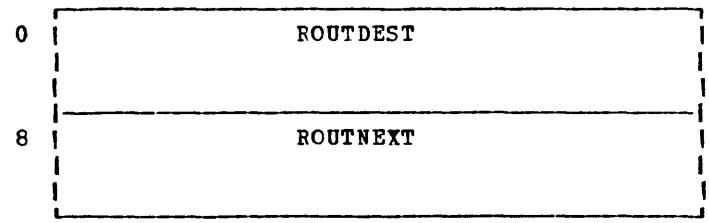
FUNCTION: Record allocation status of switchable line ports available to RSCS

DESCRIPTION: The first doubleword of the table is reserved for control information. Each following halfword contains the virtual device address of a line port which may be dialed, and which is available to RSCS.



OPERATIONAL NOTES: The line port entries are marked "in use" by setting the high-order four bits of such entries to 1s.

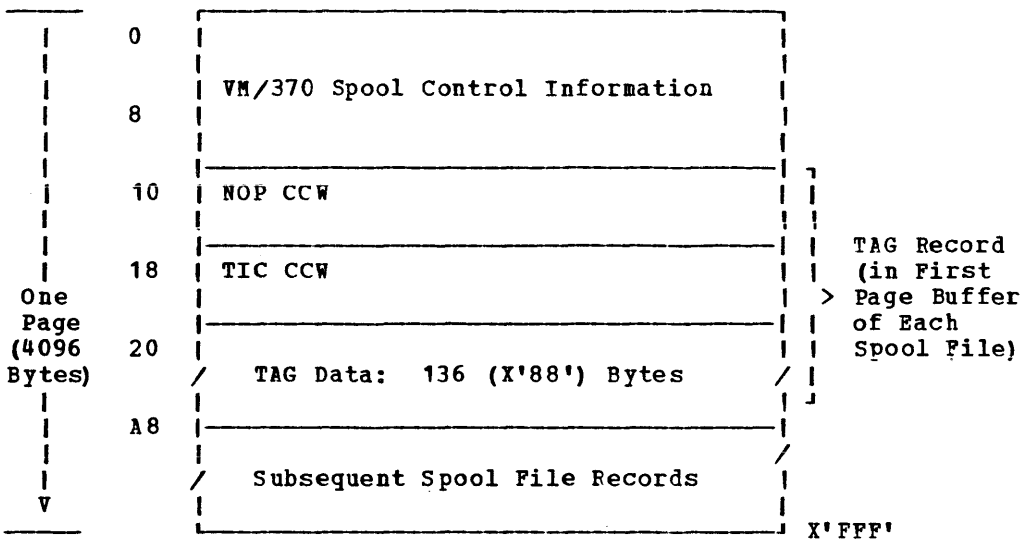
ROUTING TABLE ENTRY




```
0  ROUTDEST  DS  CL8          Final destination ID
8  ROUTNEXT  DS  CL8          LINKID for inderect routing
```

The routing table contains routing information as submitted either in the operator ROUTE commands or in the RSCS DIRECT ROUTE entries. The SVECTORS field TVECTOR1 contains TROUTE, the address of the ROUTE table.

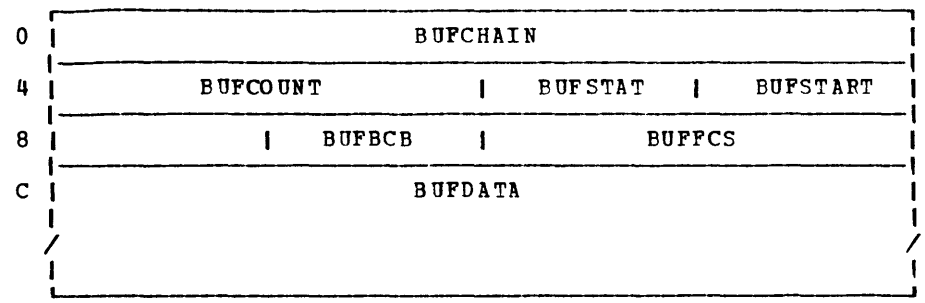
SPOOL PAGE BUFFER FORMAT



When an output file is first spooled to RSCS from a virtual machine user on the same VM/370 system, the tag data is set according to the user specification in the TAG command to CP.

When an RSCS line driver task receives a file on its link, it spools it (to itself) in an output file that has additional RSCS-specific tag data in the tag record. This RSCS-built tag data includes a store-and-forward ('S&F') flag.

TELECOMMUNICATIONS BUFFER



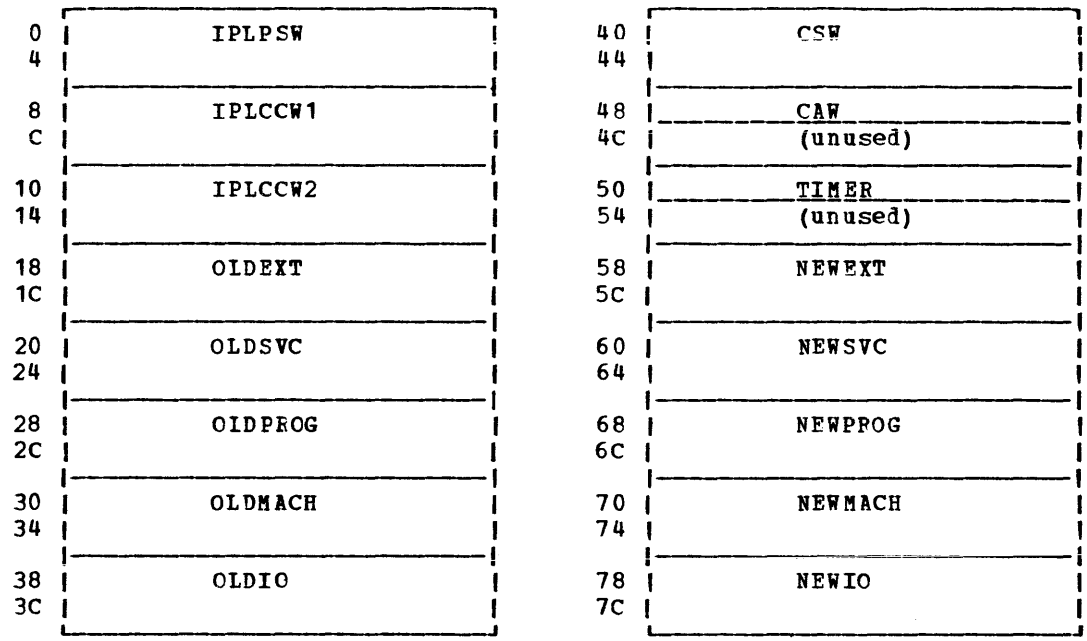

```
   BUFDSECT  DSECT
Q  BUFBEGIN  DS  0F    Beginning of the buffer
0  BUFCHAIN  DC  A(0)  Buffer chain field
4  BUFCOUNT  DS  1H    Count of bytes to transmit
6  BUFSTAT   DS  1C    Buffer status byte

   Bits defined in BUFSTAT
   BUFFAKE   EQU X'01' Dummy buffer indicator
   BUFRESP   EQU X'02' Response only in buffer
   BUFNAK    EQU X'04' NAK response (negative acknowledgement)
                       being sent
   BUFTEXT   EQU X'08' Buffer contains text information
   BUFUCHEK  EQU X'10' Unit check expected

7  BUFSTART  DS  CL2   Transmission control bytes
9  BUFBCB    DS  1C    Block control byte
A  BUFFCS    DS  CL2   Function control sequence
C  BUFDATA   DS  0F    Data portion of buffer
```

SVECTORS: LOW STORAGE DEFINITIONS

SVECTORS defines low storage for the RSCS virtual machine.
It includes two types of storage: machine-defined and RSCS-defined.

Machine-Defined Low Storage

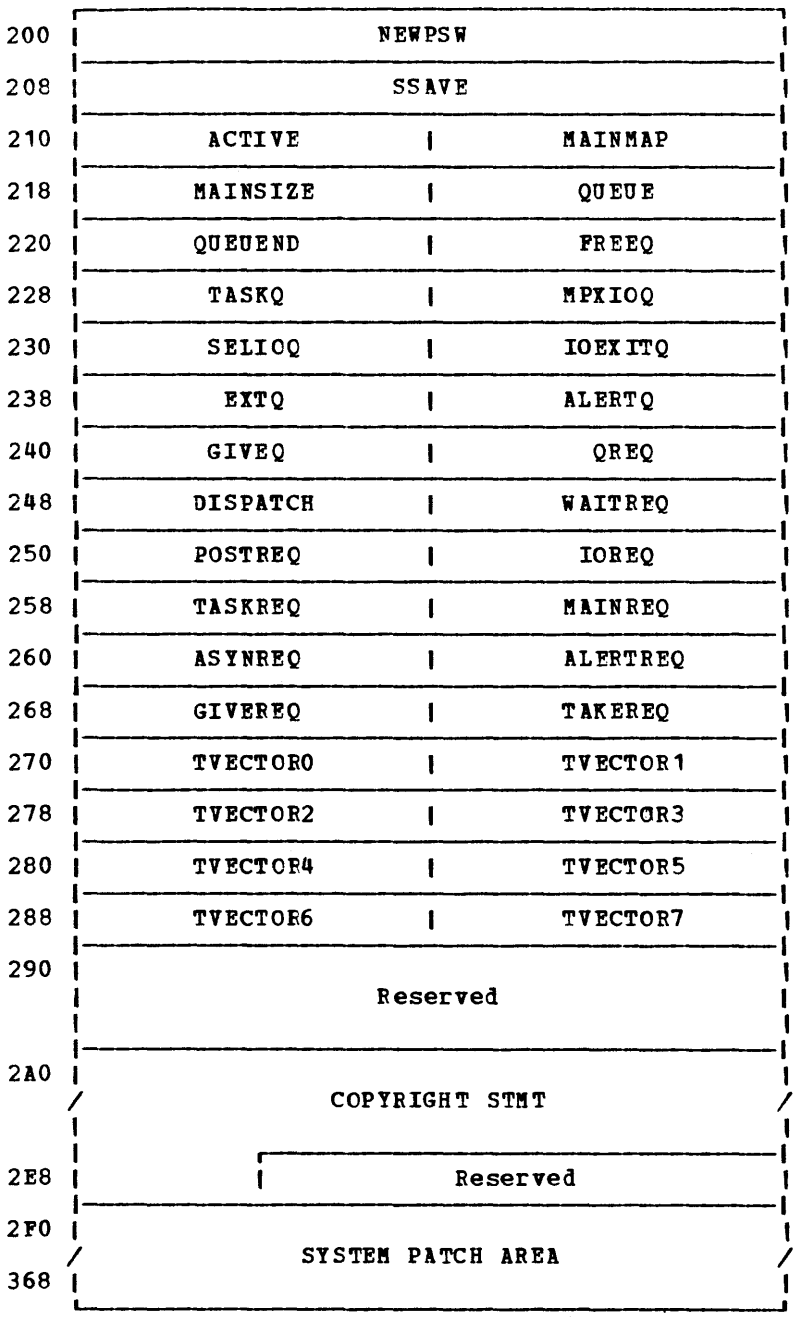
The SVECTORS machine-defined low storage defines machine status data referenced during program execution and required by System/370 architecture.




```
 0        IPLPSW   DS  D X'00040000'     V(DMTINI)
 8        IPLCCW1  DS  D
10        IPLCCW2  DS  D

18        OLDEXT   DS  D                 External interrupt old PSW
20        OLDSVC   DS  D                 Supervisor call old PSW
28        OLDPROG  DS  D                 Program check old PSW
30        OLDMACH  DS  D                 Machine check old PSW
38        OLDIO    DS  D                 Input/output old PSW

40        CSW      DS  D                 Channel status word
48        CAW      DS  D                 Channel address word

4C                 DS  F                 Unused
50        TIMER    DS  F 4X'FF'
54                 DS  F                 Unused

58        NEWEXT   DS  D X'00040000'     V(DMTEXT)
60        NEWSVC   DS  D X'00040000'     V(DMTSVC)
68        NEWPROG  DS  D X'00040000'     V(REXOUCH)
70        NEWMACH  DS  D X'00020000'     A(OLDMACH)
78        NEWIO    DS  D X'00040000'     V(DMTIOMIN)
```

SVECTORS Table Contents

RSCS program storage begins with the SVECTORS table at hex location 200 in supervisor module DMTVEC. The SVECTORS table contains pointers to modules that comprise the supervisor, to supervisor control queues, and to queues of requests for supervisor services.

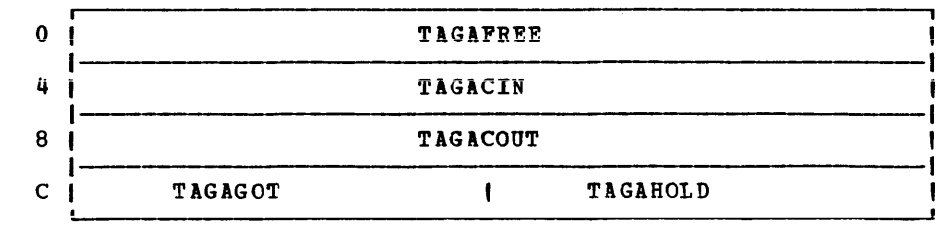


```
                ORG   SVECTORS+X'200'  Leave room for machine extensions
200     NEWPSW  DC    D'0'            Dispatched PSW for last dispatcher
208     SSAVE   DC    2F'0'           General-purpose low storage save
                                      area
210     ACTIVE  DC    X'00'           ID of currently active task
                DC    AL3(0)          Address of task element for last
                                      dispatchee
214     MAINMAP DC    A(0)            Address of start of main storage
                                      allocation map
218     MAINSIZE DC   F'0'            Total number of pages in main
                                      storage
21C     QUEUE   DC    A(0)            Address of start of supervisor
                                      queue
220     QUEUEND DC    A(0)            Address of end of last supervisor
                                      queue element
224     FREEQ   DC    A(0)            Address of start of free elememt
                                      queue
228     TASKQ   DC    A(0)            Address of start of task elememt
                                      queue
22C     MPXIOQ  DC    A(0)            Address of start of multiplexer
                                      I/O queue
230     SELIOQ  DC    A(0)            Address of start of selector I/O
                                      queue
234     IOEXITQ DC    A(0)            Address of start of asynchronous
                                      I/O request element queue
238     EXTQ    DC    A(0)            Address of start of external
                                      request element queue
23C     ALERTQ  DC    A(0)            Address of start of task
                                      asynchronous request element queue
240     GIVEQ   DC    A(0)            Address of start of GIVE request
                                      element queue
244     QREQ    DC    V(DMTQRQ)       Supervisor queue allocation
                                      request entry address
248     DISPATCH DC   V(DMTDSP)       Task dispatcher entry address
24C     WAITREQ DC    V(DMTWAT)       Wait request entry address
250     POSTREQ DC    V(DMTPST)       Post request entry address
254     IOREQ   DC    V(DMTIOMRQ)     I/O request entry address
258     TASKREQ DC    V(DMTASK)       Task management request entry
                                      address
25C     MAINREQ DC    V(DMTSTO)       Main allocation request entry
                                      address
260     ASYNREQ DC    V(DMTASY)       Asynchronous interrupt request
                                      entry address
264     ALERTREQ DC   A(DMTSIG)       Task asynchronous signal request
                                      A(ALERT) entry address
268     GIVEREQ DC    V(DMTGIV)       Task request GIVE request
                                      entry address
26C     TAKEREQ DC    V(DMTAKE)       Task request TAKE request
                                      entry address


270     TVECTOR0 DC   A(0)            Task defined field
274     TVECTOR1 DC   A(0)            Task defined field
278     TVECTOR2 DC   A(0)            Task defined field
27C     TVECTOR3 DC   A(0)            Task defined field
280     TVECTOR4 DC   A(0)            Task defined field
284     TVECTOR5 DC   A(0)            Task defined field
288     TVECTOR6 DC   A(0)            Task defined field
28C     TVECTOR7 DC   A(0)            Task defined field

        TLINKS  EQU   TVECTOR0        Link table address
        TROUTE  EQU   TVECTOR1        Route table address
        TPORTS  EQU   TVECTOR2        Switchable port table address
        TTAGQ   EQU   TVECTOR3        Tag slot queue
        TCOM    EQU   TVECTOR4        Common routine chain
```

```
        TVMID     EQU  TVECTOR5        Pointer to local host virtual
                                       machine user id
290               DS   4F
2A0     COPYRITE  EQU  *               Copyright statement
                  DC   C'5748-XP1 COPYRIGHT IBM CORP 1979 '
                  DC   C'LICENSED MATERIAL-PROGRAM PROPERTY OF IBM'
        CSTMTEND  EQU  *
2EA               DC   CL6' '         Reserved
2F0     SYSPATCH  DS   128X           ** System Patch Area **
```

TAG QUEUE DATA: TAGAREA

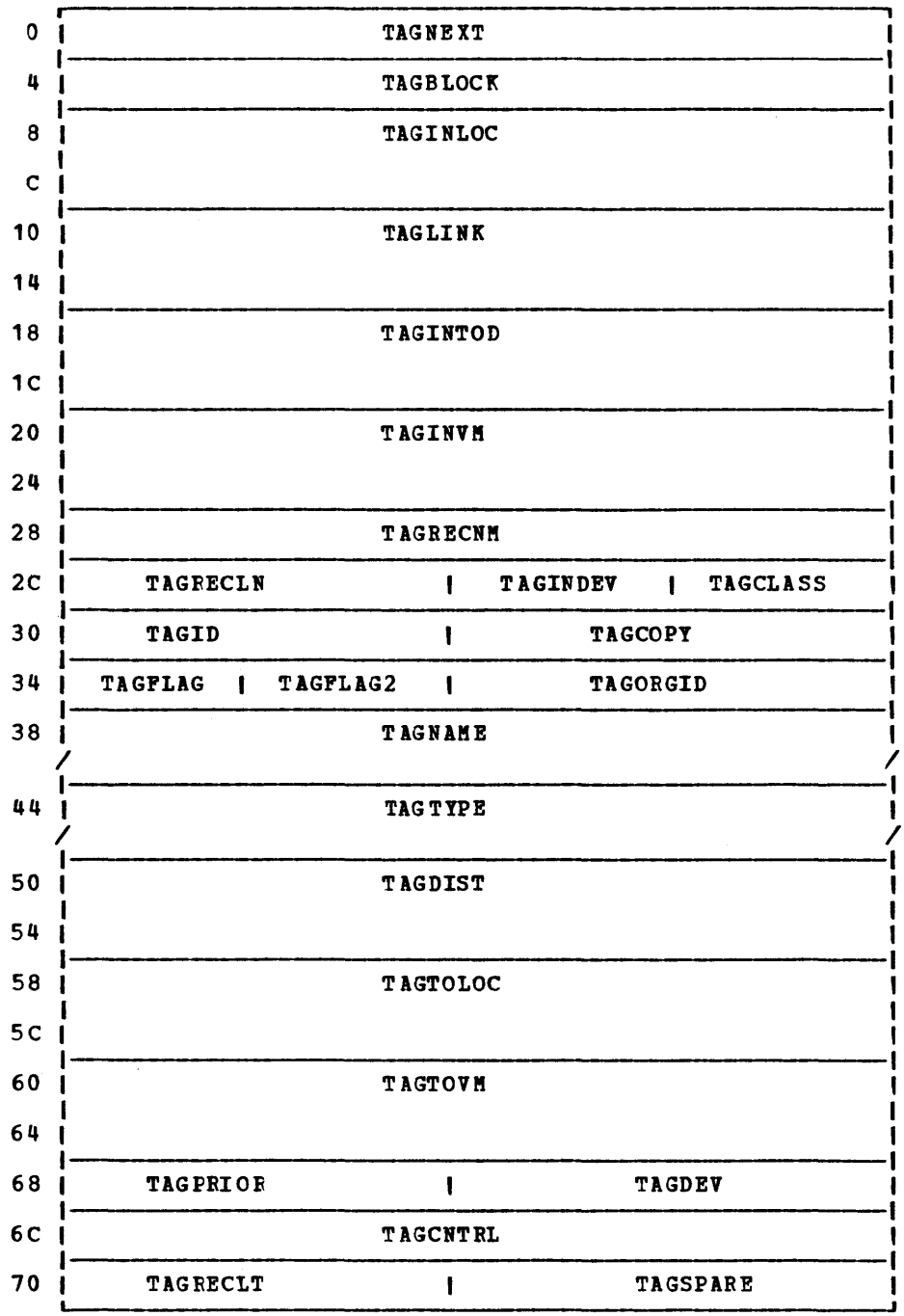
TAGAREA in DMTAXS module contains data about the disposition of the tag queue element pointers and other tag control information. It is pointed to by TTAGQ in SVECTORS.




```
0       TAGAFREE  DC  A(0)     Address of queue of free TAG slots
                               (or elements)
4       TAGACIN   DC  A(0)     Pointer to queue of active input TAGs
8       TAGACOUT  DC  A(0)     Pointer to queue of active output
                               TAGs
C       TAGAGOT   DC  H'0'     Number free slots left
E       TAGAHOLD  DC  H'0'     Number slots to be held
```

TAG QUEUE ELEMENT FOR RSCS SPOOL FILE

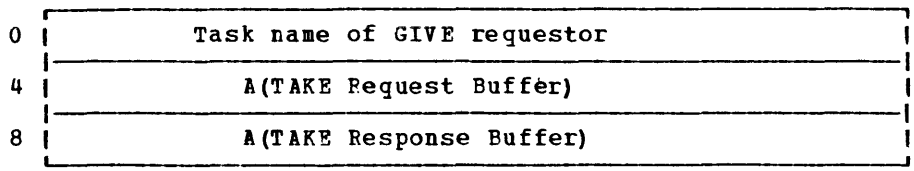
TAG describes a file enqueued for processing by RSCS. Part of the data in this area is built from tag data specified via the CP TAG command and inserted by CP into the spool file block (SFB) at the start of the spool file. RSCS reads the SFB and copies the appropriate data into the tag slot that it constructs for this file. Each tag slot entry in use is enqueued on the input (for transmission) or output (while receiving) queue of the line driver task responsible for the file.



```
 0    TAGNEXT   DS  1F      Address of next active queue entry
 4    TAGBLOCK  DS  1F      Address of associated I/O area
 8    TAGINLOC  DS  CL8     Originating location
10    TAGLINK   DS  CL8     Next location for transmission
18    TAGINTOD  DS  CL8     Time of file origin
20    TAGINVM   DS  CL8     Originating virtual machine
28    TAGRECNM  DS  1F      Number of records in file
2C    TAGRECLN  DS  1H      Maximum file data record length
2E    TAGINDEV  DS  1X      Device code of originating device
2F    TAGCLASS  DS  CL1     File output class
30    TAGID     DS  1H      Current VM/370 Spool ID
32    TAGCOPY   DS  1H      Number of copies required
34    TAGFLAG   DS  1X      VM/370 SFBLOK control flags (SFBFLAG)
35    TAGFLAG2  DS  1X      VM/370 SFBLOK control flags (SFBFLAG)
36    TAGORGID  DS  1H      VM/370 Spool ID at origin location
38    TAGNAME   DS  CL12    Filename
44    TAGTYPE   DS  CL12    Filetype
50    TAGDIST   DS  CL8     File distribution code
58    TAGTOLOC  DS  CL8     Destination location ID
60    TAGTOVM   DS  CL8     Destination virtual machine ID
68    TAGPRIOR  DS  CL2     Transmission priority
6A    TAGDEV    DS  2X      Active file's virtual device address
6C    TAGCNTRL  DS  CL4     Network Control record format
70    TAGRECLT  DS  1F      Number of records left in file
74    TAGSPARE  DS  1F      Spare Fullword

      TAGLEN    EQU *-TAGNEXT   Length (in bytes) of the file TAG
```

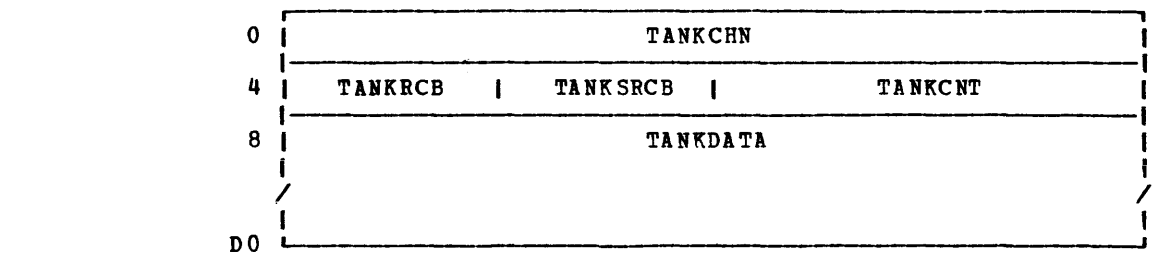
TAKE REQUEST TABLE IN GIVE/TAKE REQUESTED TASK

The format of a TAKE request table is:



TANKS

Unit Record Tank

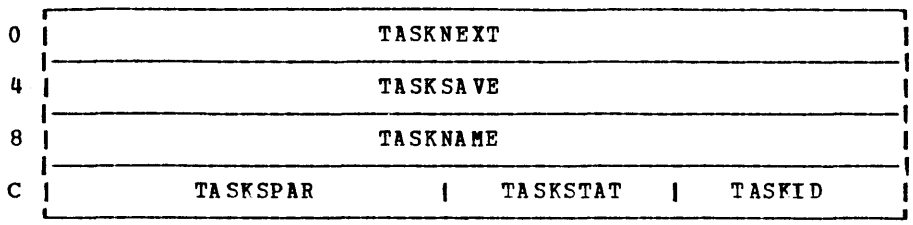


	TANKDSEC	DSECT		
0	TANKCHN	DC	A(0)	Beginning of the buffer
4	TANKRCB	DS	1C	Tank record control byte
5	TANKSRCB	DS	1C	Tank sub-record control byte
6	TANKCNT	DS	1H	Count of data bytes in tank
8	TANKDATA	DS	CL200	Data area in the tank
D0	TANKEND	DS	0F	Force next to word boundary

TASK QUEUE ELEMENT: TASKE

Each task queue element contains status information pertaining to one active task.

The TASKQ field of SVECTORS points to this queue.



0	TASKNEXT	DS	1F	Address of the next element on the task element queue
4	TASKSAVE	DS	1F	Address of this task's Task Save Area (TAREA)
8	TASKNAME	DS	CL4	Task name specified by the task; 4 bytes long
C	TASKSPAR	DS	AL2	Unused
E	TASKSTAT	DS	1X	Status flags associated with the task

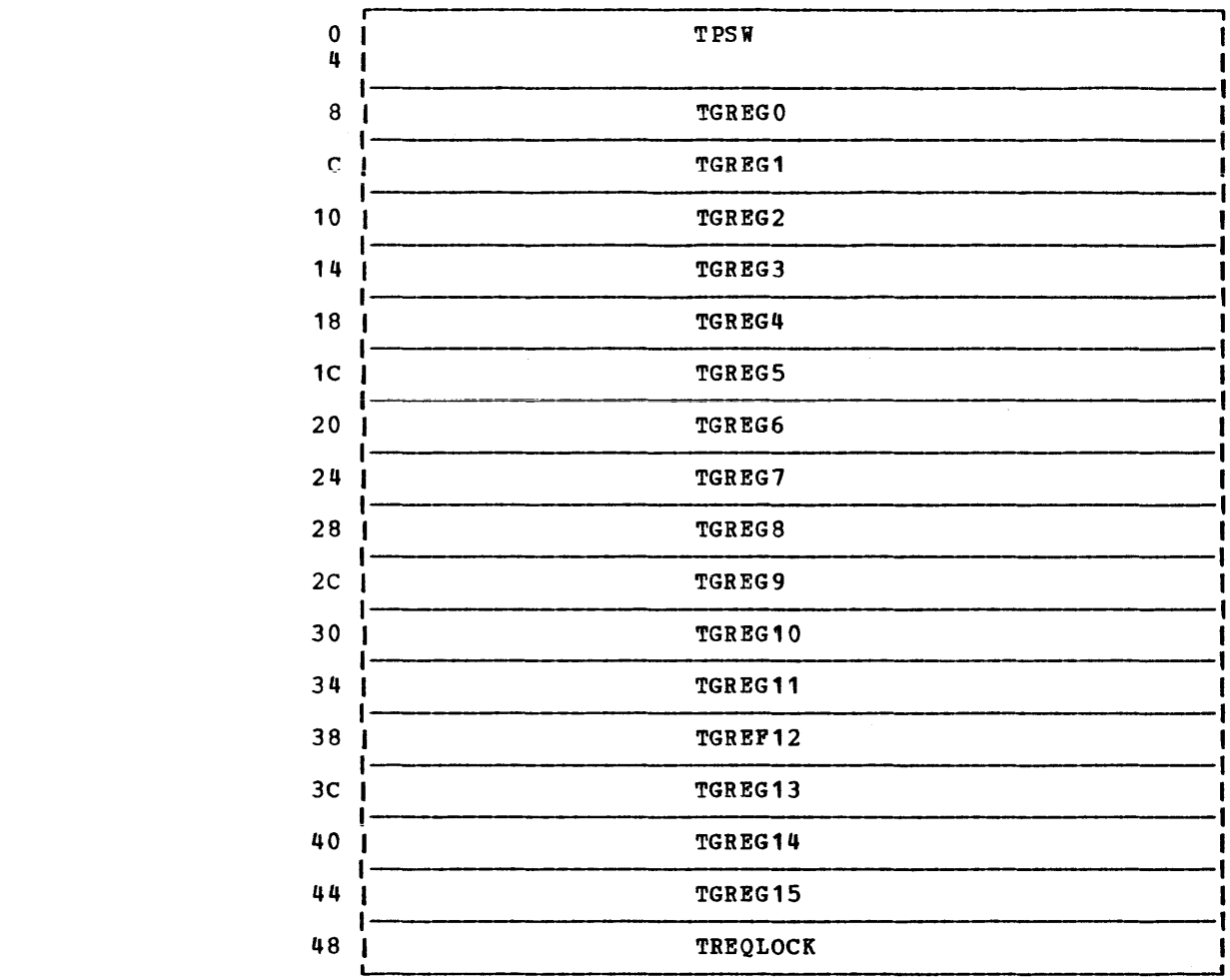
Bits defined in TASKSTAT

WAITING	EQU	X'80'	Flag set to 1 when task is nondispatchable
LOCKLIST	EQU	X'40'	Flag set to 1 while task is waiting for the synch lock list
LIMBO	EQU	X'01'	Flag set to 1 when a task is being terminated

F	TASKID	DS	1X	Number ID for the task; 1 byte is assigned by supervisor when task is made active

TASK SAVE AREA: TAREA

The task save area comprises the first 78 bytes of the storage area defined in each task's storage.




```
 0  TPSW      DS  1D  PSW with which a temporarily interrupted task
                       resumes execution
 8  TGREG0    DS  1F  Save area for general register  0
 C  TGREG1    DS  1F  Save area for general register  1
10  TGREG2    DS  1F  Save area for general register  2
14  TGREG3    DS  1F  Save area for general register  3
18  TGREG4    DS  1F  Save area for general register  4
1C  TGREG5    DS  1F  Save area for general register  5
20  TGREG6    DS  1F  Save area for general register  6
24  TGREG7    DS  1F  Save area for general register  7
28  TGREG8    DS  1F  Save area for general register  8
2C  TGREG9    DS  1F  Save area for general register  9
30  TGREG10   DS  1F  Save area for general register 10
34  TGREG11   DS  1F  Save area for general register 11
38  TGREG12   DS  1F  Save area for general register 12
3C  TGREG13   DS  1F  Save area for general register 13
40  TGREG14   DS  1F  Save area for general register 14
44  TGREG15   DS  1F  Save area for general register 15
48  TREQLOCK  DS  1F  Synchronization lock used to signal
                      whether or not a task has information
```

Request and Alert Elements

INTRODUCTION

The following pages provide information on the format and use of RSCS request and alert elements. These elements are used in task-to-task communication.

The information provided includes:

- The name of the module that builds the element
- The function performed by the element
- A brief description of the element's usage
- The format of the element
- Operational notes to assist in understanding how the element is used

The elements are grouped in this section as follows:

- Request Elements Processed by DMTREX
- Command Alert Elements for Commands Processed by DMTAXS
- Command Alert Elements for Commands Processed by Line Drivers

The function code, in byte 2 of each element, tells the alerted task the kind of alert request. The meaning of the remainder of the element is defined for that function code for that task.

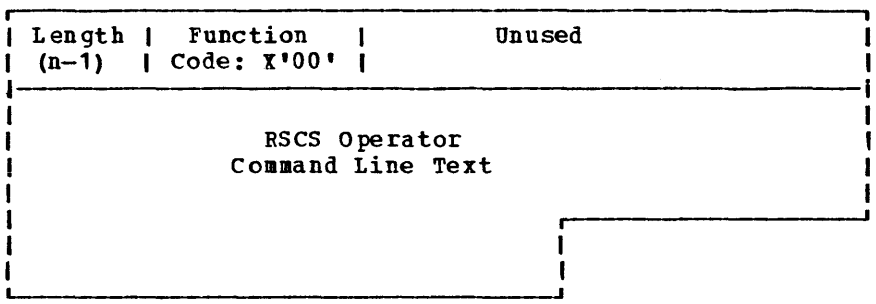
REQUEST ELEMENTS PROCESSED BY DMTREX

Command Request Element

BUILT BY: DMTNPT, DMTSML

FUNCTION: Execute an RSCS operator command

DESCRIPTION: This request element is passed by a line driver via GIVE/TAKE to the REX task in response to a command entry at a remote station.



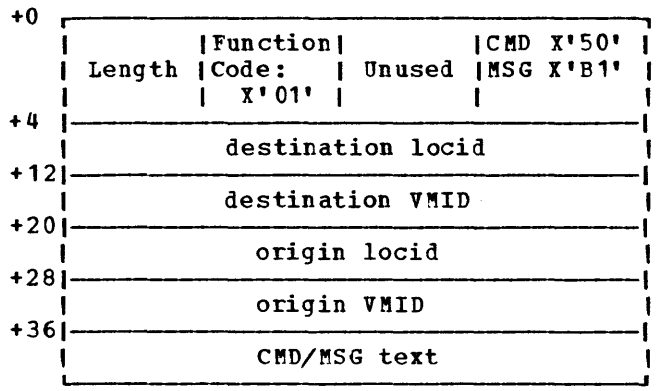
OPERATIONAL NOTES:
No response text is returned. Command responses are distributed via a call to DMTMGX.

Command/Message Routing Request Element

BUILT BY: DMTVMB, DMTVMC, DMTNJI

FUNCTION: Process a command or message received by a line driver.

DESCRIPTION: This request element is passed by a line driver via GIVE/TAKE to the REX task, to process to a command or message being received by the line driver from a remote system.



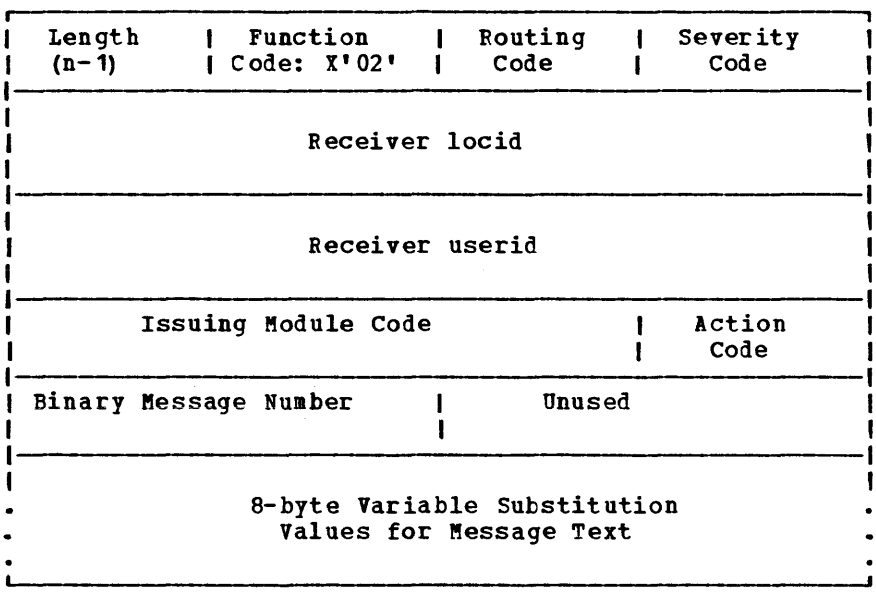
OPERATIONAL NOTES:
No response text is returned.

Message Request Element

BUILT BY: DMTREX, DMTCMX, DMTAXS, DMTNPT, DMTSML, DMTPOW, DMTNJI, DMTVMB, DMTVMC

FUNCTION: Issue an RSCS message

DESCRIPTION: This request element is passed via GIVE/TAKE to the REX task, to specify the construction and distribution of an RSCS message (by DMTMGX).



OPERATIONAL NOTES:

The routing code and severity code from the message definition (in DMTMSG) are used when not supplied in the message request element. If the message is not defined in DMTMSG, it is constructed using the specifications in the message request element, and the "variable substitution values" become the message text, unmodified.

Routing codes:

- X'80' Local RSCS console
- X'40' Remote addressee
- X'20' Local user
- X'10' Local VM/370 operator

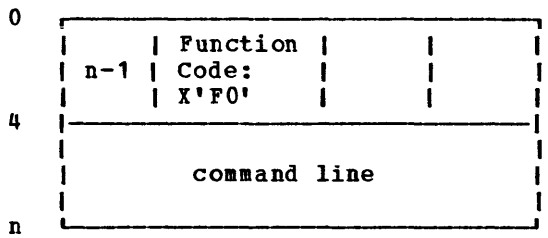
No response text is returned.

Restart Terminate Request Element

BUILT BY: DMTSML, DMTNJI, DMTPOW

FUNCTION: To terminate a line driver task specifying a command to be executed after line driver deactivation.

DESCRIPTION: This request element is passed via GIVE/TAKE to the REX task to terminate the line driver task and execute the specified command.



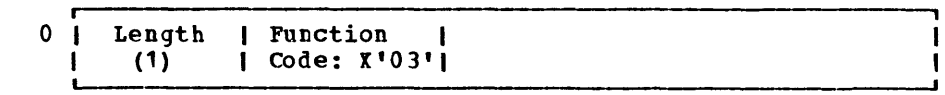
OPERATIONAL NOTES:
There are no error conditions for the restart terminate function, so no response is made. However, line driver tasks must issue a WAIT request following a call to GIVE for terminate because REX may not execute the request immediately.

Terminate Request Element

BUILT BY: DMTNPT, DMTSML, DMTVMB, DMTVMC, DMTNJI, DMTPOW

FUNCTION: Terminate line driver task.

DESCRIPTION: This request element is passed via GIVE/TAKE to the REX task, to terminate line driver operation in response to a DRAIN command.

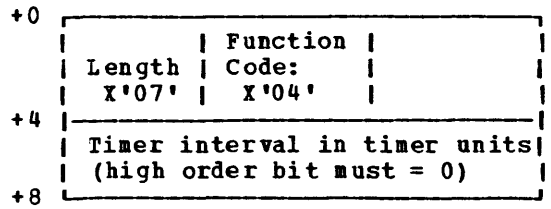


OPERATIONAL NOTES:
There are no error conditions for the terminate function, so no response is made. However, line driver tasks must issue a WAIT request following a call to GIVE for terminate, because REX may not execute the request immediately.

Timer Request Element

BUILT BY: DMTVMC, DMTSML, DMTPOW, DMTNJI

FUNCTION: To set a task internal timer.



OPERATIONAL NOTES:

The use and meaning of the fields are described below:

Response Post Codes:

X'80' - normal
X'81' - active timer interval replaced
X'84' - request format invalid

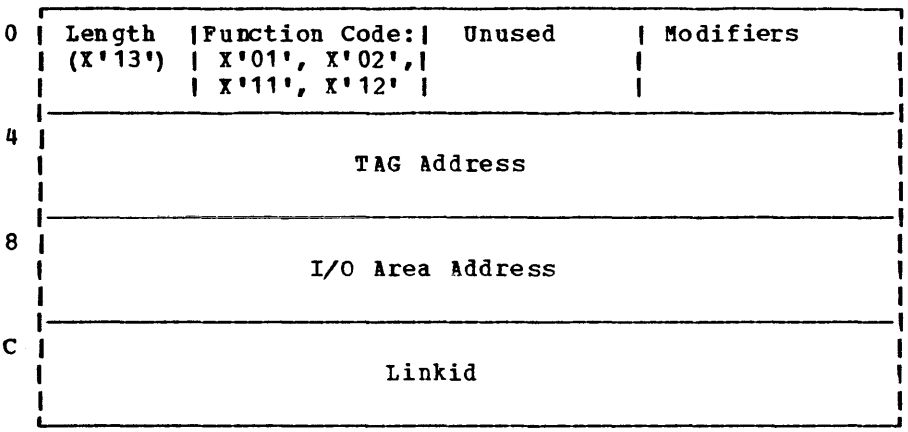
TIMER INTERVAL: Request field specifying, in timer units, the timer value to be set. One unit = 1/300 of a second.

File Request Element

BUILT BY: DMTNPT, DMTSML, DMTPOW, DMTVMB, DMTVMC, DMTNJI

FUNCTION: Initiates or terminates processing of an input or output file.

DESCRIPTION: This request element is passed via GIVE/TAKE to the AXS task by line drivers to effect local spool file access during communications with a remote station.



OPERATIONAL NOTES:

The use and meaning of the various fields depends on the requested function, as described below. Certain fields may be updated during request processing. The (updated) file request element is returned to the requestor as a GIVE response.

Open Input

Function Code: X'01'

Modifiers: Unused.

Tag Address: Response field which points to the opened file's active tag.

I/O Area Address: Response field which points to a virtual page buffer containing the opened file's first VM/370 spool data buffer.

Linkid: Request field which specifies the requesting line driver's linkid.

Response Post Codes:
- X'08' Terminal system error
- X'04' No file available
- X'02' Undefined linkid
- X'01' Previously open file returned

Open Output

Function Code: X'11'

Modifiers: X'80': Do not return possible previously opened file
X'20': Save output file on abnormal termination

Tag Address: Request field which points to a prototype file TAG for the output file, constructed by the calling line driver.

I/O Area Address: Response field which points to a virtual page buffer containing an I/O table, a write CCW, and a buffer for processing the output file.

Linkid: Request field which specifies the requesting line driver's linkid.

Response Post Codes:
X'04' Error, file not opened
X'02' Undefined linkid
X'01' Previously open file returned

Close Input

Function Code: X'02'

Modifiers:
X'80' Do not purge copy or file
X'40' Purge all copies, and purge file
X'20' Re-enqueue file for further processing.

Tag Address: Request field which points to the file's active TAG in DMTSYS, as supplied by open input.

I/O Area Address: Unused

Linkid: Unused.

Response Post Codes:
X'04' Tag not found, close failed

Close Output

Function Code: X'12'

Modifiers: X'40' Purge output file.

Tag Address: Request field which points to a prototype file TAG for the output file, constructed by the calling line driver. This tag is used to update the parameters to be set for the output file.

I/O Area Address: Request field which points to the file's I/O area, as supplied by open output.

Linkid: Unused.

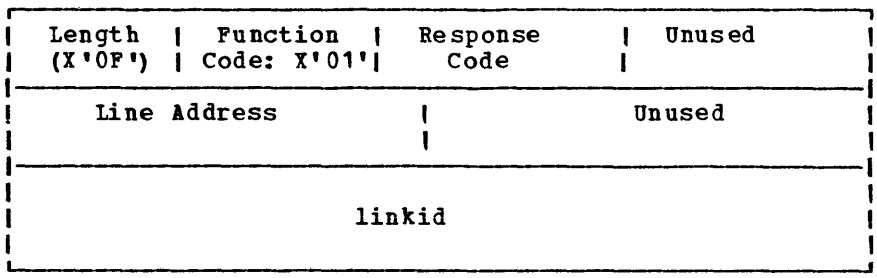
Response Post Codes:
X'04'/I/O area not found, close failed

Line Alert Element

BUILT BY: DMTCMX

FUNCTION: Request line port allocation

DESCRIPTION: This alert element is passed to the LAX task (DMTLAX) to verify and reserve line ports for links being activated in response to a START command.



OPERATIONAL NOTES:

The use and meaning of the fields are described below. Certain fields are updated during processing.

Response Codes:

- X'08' Specified line address not attached (CC=3)
- X'04' Specified line address not valid RSCS port device type
- X'02' Line not available

Line Address: Request field specifying requested line address. Zero specification implies request for allocation of a switchable line from the port table. If successful, the port's line address is returned in this field as a response.

Linkid: Response field specifying the ID of the link which has reserved the particular requested line address (with response code X'02').

COMMAND ALERT ELEMENTS FOR COMMANDS PROCESSED BY DMTAXS

Reorder Alert Element

BUILT BY: DMTCMX, DMTREX

FUNCTION: Execute a file queue reorder.

Length	Function	Response	Modifiers
X'03'	X'01'	Code	

OPERATIONAL NOTES:

Response Codes:
X'00' Element accepted for processing
X'10' Element rejected, busy

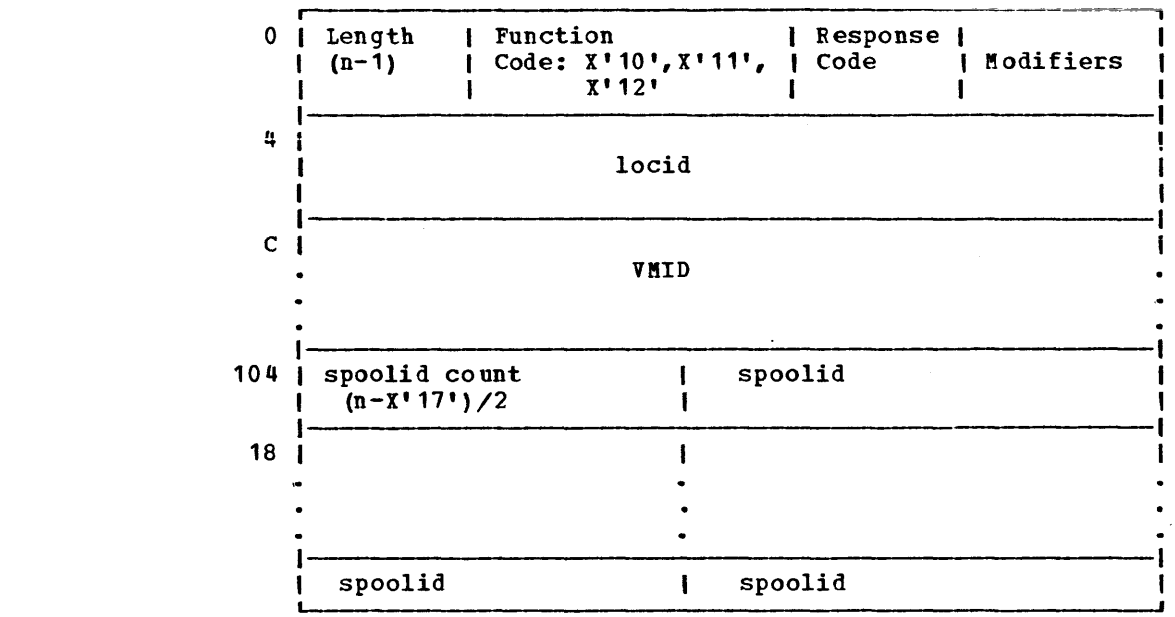
Modifiers: Unused

ORDER, PURGE, and CLOSE Command Alert Element

BUILT BY: DMTCMX

FUNCTION: Execute an AXS command

DESCRIPTION: This alert element is passed to the AXS task (DMTAXS) to request second-level processing of an ORDER, PURGE, or CLOSE command.



OPERATIONAL NOTES:

The linkid field specifies the affected link and the command origin locid. VMID specifies the command origin userid. The spoolid fields are binary halfwords; they specify the files enqueued on the specified link which are to be reordered or purged. The spoolid count field is a binary halfword; it specifies the total number of spoolid fields present. The meanings of the other fields are:

ORDER Command

Function Code: X'10'

Response Codes:
- X'00' Element accepted for processing
- X'10' Element rejected, busy

Modifiers:
- X'80' Response messages go to local RSCS operator
- X'00' Response messages go to specified link/VMID.

PURGE Command

Function Code: X'11'

Response Codes:
  X'00' Element accepted for processing
  X'10' Element rejected, busy

Modifiers:
  X'80' Response messages go to local RSCS operator
  X'40' Purge all files enqueued on the specified link
  X'00' Purge only specified files; response messages go
        to specified link/userid

CLOSE Command

Function Code: X'12'

Response Codes:
  X'00' Element accepted for processing
  X'10' Element rejected, busy

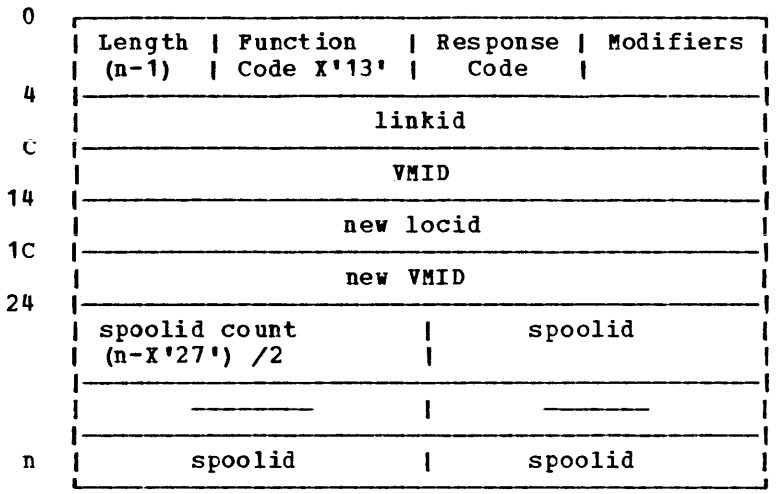
Modifiers:
  X'80' Response messages go to local RSCS operator
  X'40' CLOSE both input and output files on the specified link
  X'20' CLOSE input files on the specified link
  X'10' CLOSE output files on the specified link
  X'00' CLOSE only specified spoolids. Response messages
        go to specified link/userid.

TRANSFER Command Alert Element

BUILT BY: DMTCMX

FUNCTION: Execute an AXS command.

DESCRIPTION: This alert element is passed by a DMTCMX call to ALERTREQ to the AXS task (DMTAXS) to request second-level processing of the TRANSFER command.



OPERATIONAL NOTES:

The linkid field specifies the affected link and command origin locid. VMID specifies the command origin userid. NEW LOCID and NEW VMID are the new destination location and user IDs for the specified spoolids. The spoolid fields are binary halfwords and specify the files enqueued on the specified link which are to be transferred. The SPOOLID COUNT field is a binary halfword and specifies the total number of spoolid fields present. The meanings of the other fields are:

TRANSFER Command

Function Code: X'13'

Response Codes:
- X'00' Element accepted for processing
- X'10' Element rejected, busy

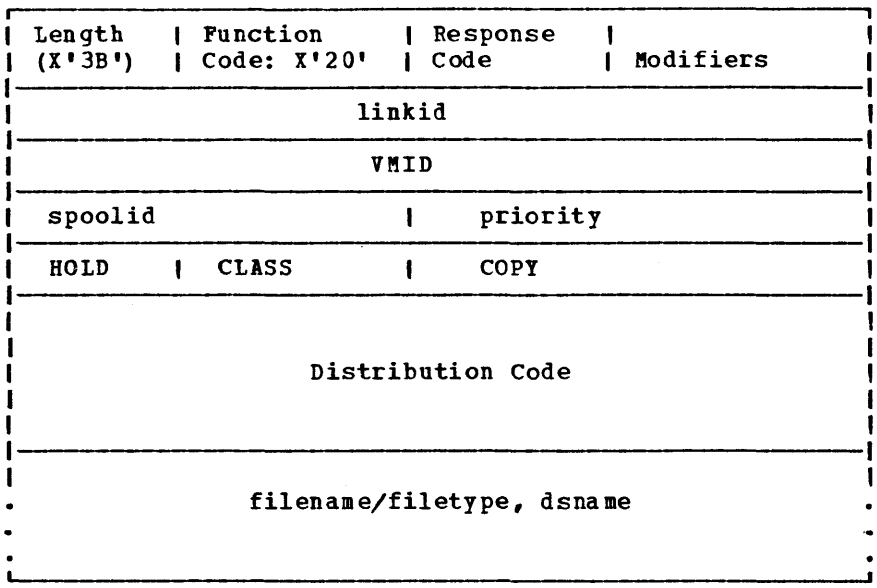
Modifiers:
- X'80' Response messages go to local RSCS operator
- X'00' Response messages go to specified link/VMID

CHANGE Command Alert Element

BUILT BY: DMTCMX

FUNCTION: Execute AXS command

DESCRIPTION: This alert element is passed by a DMTCMX alert to the AXS task (DMTAXS), to request second-level processing of a CHANGE command.



OPERATIONAL NOTES:

The linkid field specifies both the link on which the object inactive file is enqueued and the command origin locid. VMID specifies the command origin userid. The spoolid field is a binary halfword and specifies the object file's VM/370 RSCS identifier.

The following fields are specified only when the corresponding file attribute is to be changed. If the field is not specified, it is set to all 1 bits (X'FF...').

- Priority halfword binary priority 0-99
- HOLD X'7F' - set hold status
  X'3F' - reset hold status (NOHOLD)
- CLASS 1-byte EBCDIC class, A-Z, 0-9
- COPY halfword binary copy count, 1-99
- Distribution code 8-byte EBCDIC spool file distribution code
- Filename/filetype, dsname, 24-byte EBCDIC spool file filename or filetype or dsname

CHANGE Command

Function Code: X'20'

Response Codes:
X'00' Element accepted for processing
X'10' Element rejected, busy

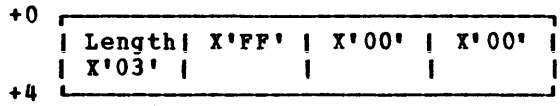
Modifiers:
X'80' Response messages go to local RSCS operator
X'00' Response messages go to specified link

Initialize Acceptor Alert Element

BUILT BY: DMTREX

FUNCTION: To inform the AXS Task that file acceptance may begin.

DESCRIPTION: Before profile execution, DMTAXS does not accept any files into its internal tag slot queue. After profile execution, DMTREX alerts DMTAXS with an initialize acceptor alert element of the following format, that file acceptance is to begin:



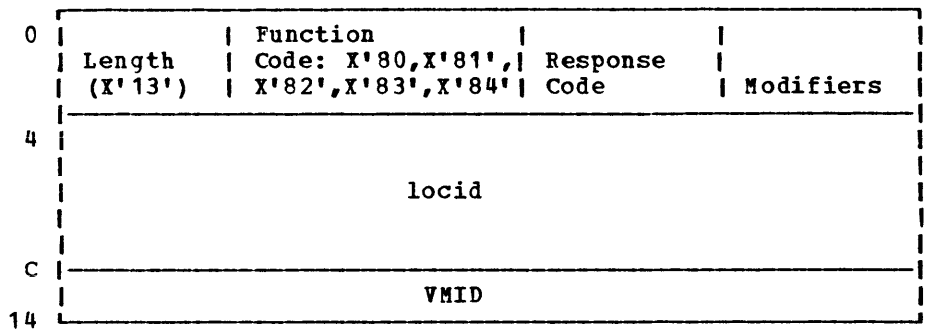
COMMAND ALERT ELEMENTS PROCESSED BY LINE DRIVERS

Line Driver Command (START, DRAIN, FREE, HOLD, TRACE) Alert Element Format

BUILT BY: DMTCMX

FUNCTION: Execute a line driver command

DESCRIPTION: This alert element is passed by a DMTCMX alert request to a line driver task (DMTNPT, DMTVMB, DMTVMC, DMTPOW, DMTNJI, DMTSML) to request second-level processing of a START, DRAIN, FREE, HOLD, or TRACE command.



OPERATIONAL NOTES:

The locid/VMID specifies the location/userid to receive response messages. The meanings of the other fields are:

START Command

Function Code: X'80'

Response Codes:
- X'00' Element accepted for processing
- X'10' Element rejected, busy

Modifiers:
- X'80' Start updated classes
- X'00' Reset DRAIN status

DRAIN Command

Function Code: X'81'

Response Codes:
- X'00' Element accepted for processing
- X'10' Element rejected, busy

Modifiers: Unused

FREE Command

Function Code: X'82'

Response Codes:
  X'00' Element accepted for processing
  X'10' Element rejected, busy

Modifiers: Unused

HOLD Command

Function Code: X'83'

Response Codes:
  X'00' Element accepted for processing
  X'10' Element rejected, busy

Modifiers:
  X'80' HOLD Immediate
  X'00' HOLD after file processing

TRACE Command

Function Code: X'84'

Response Codes:
  X'00' Element accepted for processing
  X'10' Element rejected, busy

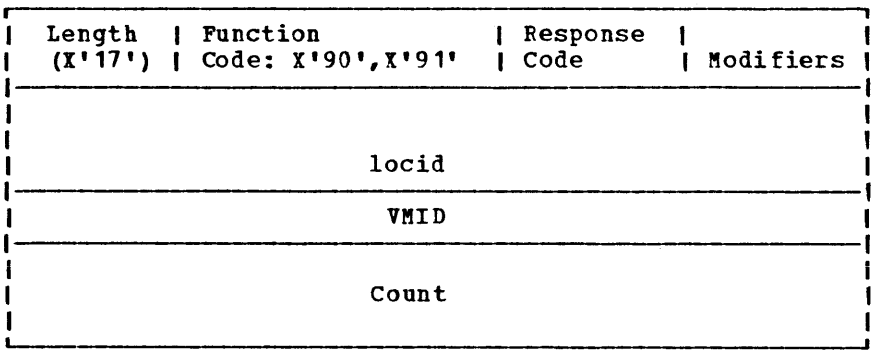
Modifiers:
  X'80' Start sum reporting
  X'40' Stop sum reporting
  X'20' Start line logging
  X'10' Stop line logging

Line Driver Command (BACKSPAC, FWDSPACE) Alert Element Format

BUILT BY: DMTCMX

FUNCTION: Execute a line driver command

DESCRIPTION: This alert element is passed by a DMTCMX alert request to a line second-level driver task (DMTNPT, DMTSML) to request second-level processing of a BACKSPAC or FWDSPACE command.



OPERATIONAL NOTES:

The locid/VMID specifies the location/userid to receive response messages. The count field is a binary fullword, and specifies the number of units to be back spaced or forward spaced. The meanings of the other fields are:

BACKSPAC Command

Function Code: X'90'

Response Codes:
- X'00' Element accepted for processing
- X'10' Element rejected, busy

Modifiers:
- X'80' Backspace count
- X'00' Backspace file (restart)

FWDSPACE Command

Function Code: X'91'

Response Codes:
- X'00' Element accepted for processing
- X'10' Element rejected, busy

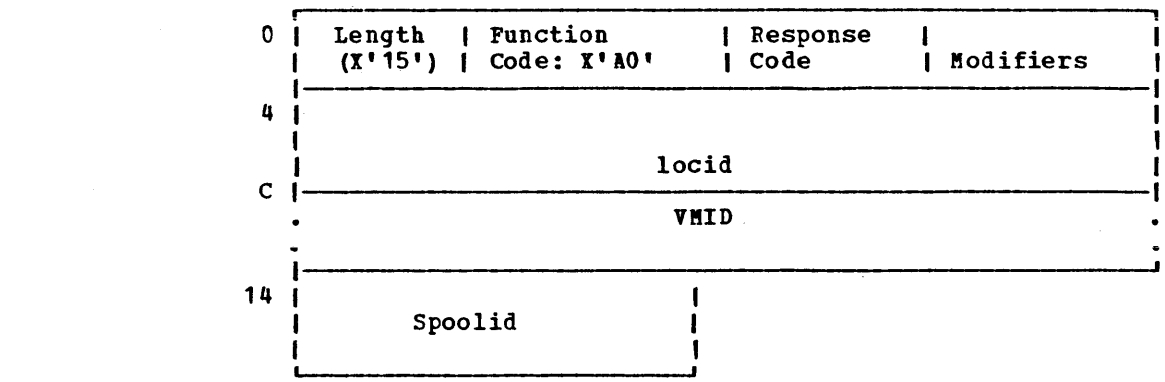
Modifiers: Unused

FLUSH Command Alert Element

BUILT BY: DMTCMX

FUNCTION: Execute a line driver command

DESCRIPTION: This alert element is passed by a DMTCMX alert request to a line driver task (DMTNPT, DMTSML) to request second-level processing of a FLUSH command.



OPERATIONAL NOTES:

The locid/VMID specifies the location/userid to receive response messages. The spoolid field is a binary halfword, and specifies the VM/370 RSCS identifier of the active file to be flushed. The meanings of the other fields are:

FLUSH Command

Function Code: X'A0'

Response Codes:
- X'00' Element accepted for processing
- X'10' Element rejected, busy

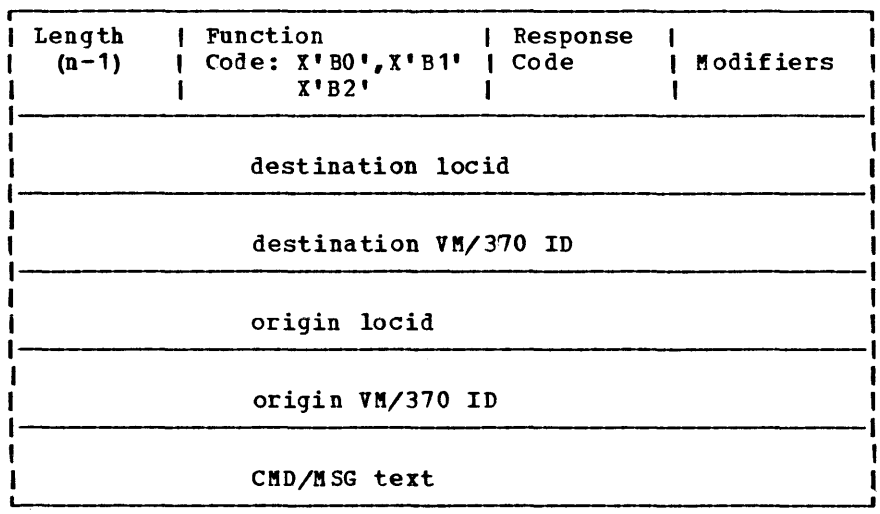
Modifiers:
- X'80' Flush all copies, purge file
- X'40' Flush hold, keep file, do not decrement copy count
- X'00' Flush, decrement copy count, purge file if no copy count remains

Line Driver Command (COMMAND, MSG, MESSAGE) Alert Element

BUILT BY: DMTCMX, DMTMGX

FUNCTION: Execute a line driver command

DESCRIPTION: This alert element is passed by either a DMTCMX or DMTMGX alert request to a line driver task (DMTNPT, DMTPOW, DMTSML) to forward messages, and to request second-level processing of a CMD command.



OPERATIONAL NOTES:

The locid specifies the location to receive the message or command text. Meanings of other fields are:

CMD Command

Function Code: X'B0'

Response Codes: X'00' Element accepted for processing
X'10' Element rejected, busy

Modifiers: Unused

MSG Command

Function Code: X'B1'

Response Codes: X'00' Element accepted for processing
X'10' Element rejected, busy

Modifiers: Unused.

System-Generated Messages

Function Code: X'B2'

Response Codes: X'00' Element accepted for processing
X'80' Element rejected, busy

Modifiers: One-byte binary RSCS severity code

NETWORK CONNECTION CONTROL RECORDS

Initial Signon Control Record and Response Signon Control Record Format

```
NCCRCB    DC  X'F0'           General record control byte
NCCSRCB   DC  X'0'            Sub-record control byte
NCCISRCB  EQU C'1'            Initial SIGNON character
NCCRSRCB  EQU C'J'            Response SIGNON character
NCCIDL    DC  AL1(NCCIL)      Length of logical record
NCCINODE  DC  CL8' '          Node identification
NCCIQUAL  DC  X'0'            Qualifier if shared spool
NCCIEVNT  DC  FL4'0'          Event sequence number
NCCIREST  DC  HL2'0'          Partial node to node resistance
NCCIBFSZ  DC  HL2'0'          Maximum transmission block size
NCCILPAS  DC  CL8' '          Line password
NCCINPAS  DC  CL8' '          Node password
NCCIFLG   DC  X'0'            Feature flags
NCCIFLGM  EQU B'10000000'     Multiple trunk (response)
NCCIL     EQU *-NCCRCB
NCCIEND   DC  X'0'            End RCB
```

Concur/Reset Signon Control Record Format

```
NCCRCB    DC  X'F0'           General record control byte
NCCSNB    DC  X'0'            Sub-record control byte
NCCESRCB  EQU C'K'            Reset SIGNON character
NCCCSRCB  EQU C'L'            Concur SIGNON character
NCCCDL    DC  AL1(NCCCL)      Length of logical record
NCCCEVNT  DC  FL4'0'          Event sequence number
NCCCREST  DC  OHL2'0'         Total node to node resistance
NCCEREST  DC  HL2'0'          Partial node to node resistance
NCCCL     EQU *-NCCRCE
NCCCEND   DC  X'0'            End RCB
```

Add/Subtract Connection Control Record Format

```
NCCRCB    DC  X'F0'           General record control byte
NCCSRCB   DC  X'0'            Sub-record control byte
NCCASRCB  EQU C'M'            Add connection character
NCCSSRCB  EQU C'N'            Subtract connection character
NCCADL    DC  AL1(NCCAL)      Length of logical record
NCCANODA  DC  CL8' '          Lower node identification
NCCAQULA  DC  X'0'            Qualifier if shared spool
NCCANODB  DC  CL8' '          Higher node identification
NCCAQULB  DC  X'0'            Qualifier if shared spool
NCCAEVNT  DC  FL4'0'          Event sequence number
NCCAREST  DC  HL2'0'          Node to node resistance (total)
NCCAL     EQU *-NCCRCB
NCCAEND   DC  X'0'            End RCB
```

NETWORK JOB HEADER RECORD FORMAT: NJHDSECT

```
*         Block Control Information

NJHLEN    DC  AL2(NJHLLEN)      Length of entire block
NJHFLAGS  DC  X'00'             Flags
NJHSEQ    DC  BL.1'0',AL.7(0)   Transmission sequence indicator
NJHLBCI   EQU *-NJHDSECT        Length of block control information

*         General Section

NJHG      DS  0F                Start of general section
NJHGLEN   DC  AL2(NJHGLLEN)     Length of general section
NJHGFLGS  DS  0BL2              Section type flags
NJHGTYPE  DC  AL1(NTYPGEN)        ID for general section
NJHGMOD   DC  AL1 (NJHG$MOD)      Modifier
NJHG$MOD  EQU B'00000000'         Value of modifier

NJHGJID   DC  Y(0)              Job identifier
NJHGJCLS  DC  C'A'              Job class
NJHGMCLS  DC  C'A'              Message class
NJHGFLG1  DC  B'00000000'       Flags
NJHGPRIO  DC  AL1(0)            Selection priority
NJHGORGQ  DC  AL1(0)            Origin node system qualifier
NJHGJCPY  DC  AL1(0)            Job copy count
NJHGLRCT  DC  AL1(0)              Job line count
          DC  XL3'00'             Reserved
NJHGACCT  DC  CL8' '            Networking account number
NJHGJNAM  DC  CL8' '            Job name
NJEGUSID  DC  CL8' '            User ID (TSO, VM/370)
NJHGPASS  DS  CL8               Password
NJHGNPAS  DS  CL8               New password
NJHGETS   DC  FL8'0'            Entry time/date stamp
NJHGORGN  DC  CL8' '            Origin node name
NJHGORGN  DC  CL8' '            Origin remote name
NJHGXEQN  DC  CL8' '            Execution node name
NJHGXEQU  DC  CL8' '            Execution user ID (VM/370)
NJHGPRTN  DC  CL8' '            Default print node name
NJHGPRTR  DC  CL8' '            Default print remote name
NJHGPUNN  DC  CL8' '            Default punch node name
NJHGPUNR  DC  CL8' '            Default punch remote name
NJHGFORM  DC  CL8' '            Job forms
NJHGICRD  DC  F'0'              Input card count
NJHGETIM  DC  F'0'              Estimated execution time
NJHGELIN  DC  F'0'              Estimated output lines
NJHGECRD  DC  F'0'              Estimated output cards
NJHGPRGN  DC  CL20' '           Programmer's name
NJHGROOM  DC  CL8' '            Programmer's room number
NJHGDEPT  DC  CL8' '            Programmer"s department number
NJHGBLDG  DC  CL8' '            Programmer's building number
NJHGNREC  DC  F'0'              Record count on output
NJHGEND   DS  0F                End of general section
NJHGLLEN  EQU *-NJHG            Length of general section
```

```
*         JES2 Subsystem Section

NJH2      DS  0F              Start of JES2 section
NJH2LEN   DC  AL2(NJH2LLEN)   Length of JES2 section
NJH2FLGS  DS  0BL2            Section type flags
NJH2TYPE  DC  AL1(NTYPJES2)     ID for JES2 section
NJH2MOD   DC  AL1(NJH2$MOD)     Modifier
NJH2$MOD  EQU B'00000000'       Valud of modifier

NJH2FLG1  DC  B'00000000'     Flags
          DC  XL3'00'         Reserved
NJH2ACCT  DC  CL4' '          Originator's JES2 account number
NJH2END   DS  0F              End of JES2 section
NJH2LLEN  EQU *-NJH2          Length of JES2 section

NJHLLEN   EQU *-NJHDSECT      Length of entire block

*         Recommended Format for a User Section

NJHU      DS  0F              Start of user section
NJHULEN   DC  AL2(NJHULLEN)   Length of user section
NJHUFLGS  DS  0BL2            Section type flags
NJHUTYPE  DC  AL1(NTYPUSER)     ID for user section --
*                                 Bits 0-1 must be B'11'
*                                 Bits 2-7 can be anything
NJHUMOD   DC  AL1(NJHU$MOD)     Modifier --
NJHU$MOD  EQU B'00000000'         Mod value can be anything

NJHUCODE  DC  CL4' '          SHARE/GUIDE installation code
*                             Place user information fields
*                               between 'NJHUCODE' & 'NJHUEND'
NJHUEND   DS  0F              End of user section
NJHULLEN  EQU *-NJHJ          Length of user section

*         Section Type Flags

NTYPGEN   EQU B'00000000'     General section
NTYPSUB   EQU B'10000000'     Subsystem section
NTYPASP   EQU B'10000001'     ASP   subsystem section
NTYPHASP  EQU B'10000010'     HASP  subsystem section
NTYPJES1  EQU B'10000011'     JES/RES subsystem section
NTYPJES2  EQU B'10000100'     JES2  subsystem section
NTYPJES3  EQU B'10000101'     JES3  subsystem section
NTYPPOWR  EQU B'10000110'     VSE/POWER subsystem section
NTYPVNET  EQU B'10000111'     VM/370 subsystem section

NTYPUSER  EQU B'11000000'     User section

*         General Section, NJHGFLG1

NJHGF1PR  EQU B'10000000'     Do not recompute priority
```

NETWORK JOB TRAILER RECORD FORMAT: NJTDSECT

```
*         Block Control Information

NJTLEN    DC  AL2(NJTLLEN)    Length of entire block
NJTFLAGS  DC  X'00'           Flags
NJTSEQ    DC  BL.3'0',AL.7(0) Transmission sequence indicator
NJTLBCI   EQU *-NJTDSECT      Length of block control information
```

```
*         General Section

NJTG      DS  0F              Start of general section
NJTGLEN   DC  AL2(NJTGLLEN)   Length of general section
NJTGFLGS  DS  0BL2            Section type flags
NJTGTYPE  DC  AL1(NTYPGEN)      ID for general section
NJTGMOD   DC  AL1(NJTG$MOD)     Modifier
NJTG$MOD  EQU B'00000000'       Value of modifier

NJTGFLG1  DC  B'00000000'     Flags
NJTGXCLS  DC  C'A'            Actual execution class
          DC  XL2'0'          Reserved
NJTGSTRT  DC  FL8'0'          Execution start time/date
NJTGSTOP  DC  FL8'0'          Execution stop time/date
NJTGACPU  DC  F'0'            Actual CPU time
NJTGALIN  DC  F'0'            Actual output lines
NJTGACRD  DC  F'0'            Actual output cards
NJTGEXCP  DC  F'0'            Excp count
NJTGIXPR  DC  AL1(0)          Initial XEQ  selection priority
NJTGAXPR  DC  AL1(0)          Actual  XEQ  selection priority
NJTGIOPR  DC  AL1(0)          Initial output selection priority
NJTGAOPR  DC  AL1(0)          Actual output selection priority
NJTGEND   DS  0F              End of general section
NJTGLLEN  EQU *-NJTG          Length of general section
NJTLLEN   EQU *-NJTDSECT      Length of entire block

*         Recommended Format for a User Section

NJTU      DS  0F              Start of user section
NJTULEN   DC  AL2(NJTOLLEN)   Length of user section
NJTUFLGS  DS  0BL2            Section type flags
NJTUTYPE  DC  AL1(NTYPUSER)     ID for user section --
*                                 Bits 0-1 must be B'11'
*                                 Bits 2-7 can by anything
NJTUMOD   DC  AL1(NJTU$MOD)     Modifier --
NJTU$MOD  EQU B'00000000'         Mod value can be anything

NJTUCODE  DC  CL4' '          SHARE/GUIDE installation code
*                             Place user information fields
*                               between 'NJTUCODE' & 'NJTUEND'
NJTUEND   DS  0F              End of user section
NJTULLEN  EQU *-NJTU          Length of user section


NETWORK DATA SET HEADER RECORD FORMAT: NDHDSECT


*         Block Control Information

NDHLEN    DC  AL2(NDHLLEN)    Length of entire block
NDHFLAGS  DC  X'00'           Flags
NDHSEQ    DC  BL.1'0',AL.7(0) Transmission sequence indicator
NDHLBCI   EQU *-NDHDSECT      Length of block control information

*         General Section

NDHG      DS  0F              Start of general section
NDHGLEN   DC  AL2(NDHGLLEN)   Length of general section
NDHGFLGS  DS  0BL2            Section type flags
NDHGTYPE  DC  AL1(NTYPGEN)      ID for general section
NDHGMOD   DC  AL1(NDHG$MOD)     Modifier
NDHG$MOD  EQU B'00000000'       Value of modifier

NDHGNODE  DC  CL8' '          Destination node name
```

```
NDHGRMT   DC  CL8' '          Destination remote name
NDHGPROC  DC  CL8' '          Proc invocation name
NDHGSTEP  DC  CL8' '          Step name
NDHGDD    DC  CL8' '          DD name
NDHGDSNO  DC  H'0'            Data set number
NDHGSEC   DC  AL1(0)          Security level
NDHGCLAS  DC  C'A'            Output class
NDHGNREC  DC  F'0'            Record count
NDHGFLG1  DC  B'00000000'     Flags
NDHGRCFM  DC  B'00000000'     Record format
NDHGLREC  DC  H'0'            Max logical record length
NDHGDSCT  DC  AL1(1)          Data set copy count
NDHGFCBI  DC  AL1(0)          3211 FCB index
          DC  XL2'00'         Reserved
NDHGFORM  DC  CL8' '          Forms ID
NDHGFCB   DC  CL8' '          FCB ID
NDHGUCS   DC  CL8' '          UCS ID
NDHGXWTR  DC  CL8' '          External writer ID
NDHGEND   DS  0F              End of general section
NDHGLLEN  EQU *-NDHG          Length of general section

*         RSCS Subsystem Section

NDHV      DS  0F              Start of RSCS section
NDHVLEN   DC  AL2(NDHVLLEN)   Length of RSCS section
NDHVFLGS  DS  0BL2            Section type flags
NDHVTYPE  DC  AL1(NTYPVNET)     ID for RSCS section
NDHVMOD   DC  AL1(NDHV$MOD)     Modifier
NDHV$MOD  EQU B'00000000'       Value of modifier

NDHVFLG1  DC  B'00000000'     Flags
NDHVCLAS  DC  C'A'            VM/370 spool file class
NDHVIDEV  DC  X'00'           VM/370 origin device type
          DC  XL1'00'         Reserved
NDHVDIST  DC  CL8' '          VM/370 distribution code
NDHVFNAM  DC  CL12' '         VM/370 file name
NDHVFTYP  DC  CL12' '         VM/370 file type
NDHVPRIO  DC  AL2(0)          VM/370 transmission priorty
NDHVEND   DS  0F              End of RSCS section
NDHVLLEN  EQU *-NDHV          Length of RSCS section

NDHLLEN   EQU *-NDHDSECT      Length of entire block

*         3800 Printer Characteristics General Section (Optional)

NDHA      DS  0F              Start of 3800 char section
NDHALEN   DC  Y(NDHALLEN)     Length of 3800 char section
NDHAFLGS  DS  0BL2            Flags and modifier
NDHATYPE  DC  AL1(NTYPGEN)      ID for general section
NDHAMOD   DC  AL1(NDHA$MOD)     Modifier
NDHA$MOD  EQU B'10000000'       Value of modifier (3800 char)

NDHAFLG1  DC  B'00000000'     Flags
NDHAFLCT  DC  AL1(0)          Flash count
NDHATREP  DC  X'00'           Table reference character
          DC  X'00'           Reserved
NDHATAB1  DC  CL8' '          Translate table 1
NDHATAB2  DC  CL8' '          Translate table 2
NDHATAB3  DC  CL8' '          Translate table 3
NDHATAB4  DC  CL8' '          Translate table 4
NDHAFLSH  DC  CL8' '          Flash cartridge ID
NDHAMODF  DC  CL8' '          Copy modification ID
NDHACPYG  DC  XL8'00'         Copy groups
NDHAEND   DS  0F              End of 3800 char section
NDHALLEN  EQU *-NDHA          Length of 3800 char section
```

```
*         3800 Characteristics General Section, NDHAFLG1

NDHAF1J   EQU B'10000000'      'OPTCD=J' specified
NDHAF1BR  EQU B'01000000'      'BURST=YES' specified

*         Record Characteristics Change General Section

NDHC      DS  0F               Start of char change general section
NDHCLEN   DC  AL2(NDHCLLEN)    Length of char change general section
NDHCFLGS  DS  0BL2             Section type flags
NDHCTYPE  DC  AL1(NTYPGEN)       ID for general section
NDHCMOD   DC  AL1(NDHC$MOD)      Modifier
NDHC$MOD  EQU B'01000000'        Value of modifier (char change)

NDHCFLG1  DC  B'00000000'      Flags
NDHCRCFM  DC  B'00000000'      Record format
NDHCLREC  DC  AL2(0)           Maximum logical record length
NDHCEND   DS  0F               End of char change general section
NDHCLLEN  EQU *-NDHC           Length of char change general section

*         Recommended Format for a User Section

NDHU      DS  0F               Start of user section
NDHULEN   DC  AL2(NDHULLEN)    Length of user section
NDHUFLGS  DS  0BL2             Section type flags
NDHUTYPE  DC  AL1(NTYPUSER)      ID for user section --
*                                  Bits 0-1 must be B'11'
*                                  Bits 2-7 can be anything
NDHUMOD   DC  AL1(NDHU$MOD)      Modifier --
NDHU$MOD  EQU B'00000000'          Mod value can be anything

NDHUCODE  DC  CL4' '           SHARE/GUIDE installation code
*                              Place user information fields
*                                between 'NDHUCODE' & 'NDHUEND'
NDHUEND   DS  0F               End of user section
NDHULLEN  EQU *-NDHU           Length of user section

*         General Section, NDHGFLG1

NDHGF1SP  EQU B'10000000'      Spin data set
NDHGF1HD  EQU B'01000000'      Hold data set at destination
NDHGF1LG  EQU B'00100000'      Job log indicator


COMMAND/MESSAGE HEADER FORMATS


NMRDSECT  DSECT
NMRFLAG   DC  X'0'             Flag byte
NMRLEVEL  DC  0X'0'            Importance level (high 4 bits)
NMRPRIO   DC  X'0'             Output priority (low 4 bits)
NMRTYPE   DC  X'0'             Type byte
NMRML     DC  X'0'             Length of message
NMRTO     DC  0CL9' '          To node
NMRTONOD  DC  CL8' '           To node name
NMRTOQUL  DC  X'0'             To node qualifier
NMROUT    DC  XL8'0'           Local output informaton
NMRFM     DC  0CL9' '          From node
NMRFMNOD  DC  CL8' '           From node name
NMRFMQUL  DC  X'0'             From node qualifier
NMRMSG    DC  CL132' '         Message
NMRL      EQU *-NMRDSECT
NMR       EQU NMRDSECT,NMRL    Alias for NMRDSECT with length
```

```
*         Formatted Command Definitions

          ORG  NMRMSG
NMRFNORM  DC   0XL20'0'          Formatted area for normal command
NMRFRTE   DC   0XL36'0'          Formatted area for route command
NMRFOP    DC   X'0'              Op code
NMRFFLG   DC   X'0'              Flags or op code modifier
NMRFJID   DC   XL2'0'            Initial job number
NMRFORGN  DC   CL8' '            Origin node name
NMRFJNAM  DC   CL8' '            Job name
NMRFD     DC   CL8' '            Destination for route command
NMRFR     DC   CL8' '            Remote if not implied by NMRFD

*         NMROUT Format for UCMID Messages

          ORG  NMROUT
NMRUCM    DC   X'0'              MCS console ID
NMRUCMA   DC   X'0'              MCS console area
NMRLINET  DC   XL2'0'            Line type for MLWTO

*         NMROUT Format for Logical Routed Messages

          ORG  NMROUT
NMRDESC   DC   XL2'0'            MCS descriptor codes
NMRROUT   DC   XL2'0'            MCS console routings
NMRDOMID  DC   XL4'0'            MCS DOM ID

*         NMROUT Format for Remote Messages

          ORG  NMROUT
NMRRMT    DC   CL8' '            Remote name 'rnnn    '

*         DMROUT for TSO User Messages

          ORG  NMROUT
NMRUSER   DC   CL8' '            TSO user ID

*         NMRFLAG Definitions

NMRFLAGC  EQU  B'10000000'       NMRMSG contains a command
NMRFLAGW  EQU  B'01000000'       NMROUT has JES2 remote number
NRMFLAGT  EQU  B'00100000'       NMROUT has TSO user ID
NMRFLAGU  EQU  B'00010000'       NMROUT has UCMID information
NMRFLAGR  EQU  B'00001000'       Console is only remote authorized
NMRFLAGJ  EQU  B'00000100'       Console not job authorized
NMRFLAGD  EQU  B'00000010'       Console not device authorized
NMRFLAGS  EQU  B'00000001'       Console not system authorized

*         NMRTYPE Definitions

NMRTYPEX  EQU  B'11110000'       Reserved bits
NMRTYPEF  EQU  2                 Formatted command in nmrmsg

*         NMRFOP Definitions

NMRFOPD   EQU  1                 Display job command
NMRFOPC   EQU  2                 Cancel job command
NMRFOPA   EQU  3                 Release job command
NMRFOPH   EQU  4                 Hold job command
NMRFOPR   EQU  5                 Route job command

*         NMRFFLG Definitions

NMRFFLGO  EQU  X'80'             Cancel or route output
NMRFFLGD  EQU  X'40'             Cancel execution with dump
```

Section 6: Diagnostic Aids

Problem Determination

The complexity and physically dispersed nature of data telecommunication systems can lead to difficulty in diagnosing malfunctions. Because many different problems with the various system components can present external symptoms which appear identical at the operator and user level, it is important to collect as much physical documentation pertaining to the malfunction as possible. Such documentation may include virtual storage dumps of the RSCS virtual machine, the RSCS operator console hard copy output, a log trace of the line I/O activity, and copies of files received in error.

In the event of a definite program error that can be detected by the CPU or channel hardware (e.g., program checks), or by the RSCS system itself, a virtual storage dump is automatically taken by means of a CP DUMP command executed by a DIAGNOSE X'08' instruction. In this case, the RSCS operator console output normally includes a terminal diagnostic error message, as well as information concerning the activity leading up to the error. Any available console output containing such information should be retrieved and attached to the virtual storage dump.

In many cases, minor malfunctions detectable by users, operators, and system programmers do not result in automatic virtual storage dumps. These malfunctions usually leave some evidence in RSCS virtual storage following their occurrence, and a dump of RSCS virtual storage can be useful if taken quickly after the malfunction occurs, before subsequent processing obscures the evidence. When a malfunction is detected, RSCS processing should be suspended by using the RSCS virtual machine console to place the virtual machine in CP console function mode (normally, by use of the ATTENTION key). Having entered CP console function mode, the following CP command should be entered:

```
DUMP 0- * (comment describing symptoms)
```

When CP has confirmed DUMP command completion, the following two CP commands should be entered:

```
CLOSE E
BEGIN
```

The CLOSE command causes the storage dump print file to be queued for real printer output, and the BEGIN command causes RSCS to resume processing from where it was suspended. The character string to the right of the asterisk is printed as entered at the start of the dump listing; it should identify reason for taking the storage dump. All console listings (including the DUMP command) and other documentation should be retrieved and attached to the dump, and any further description of the problem should be written on the dump listing or attached to it.

Suspected telecommunication problems should be documented by means of the RSCS TRACE command, using both the LOG and SUM keywords. A line activity log trace should be activated during the occurrence of the malfunction if possible. This presents no problem if the malfunction is constant or reproduceable at will, but intermittent or unpredictable malfunctions can sometimes be difficult to "catch". Line activity logging is initiated by the RSCS command:

TRACE 'linkid' LOG

and is terminated by the RSCS command:

TRACE 'linkid' NOLOG

The log trace is automatically scheduled for real printer output on termination of logging, which should be done several seconds or minutes after the malfunction has occurred. (Avoid allowing log trace to remain active for long periods, since this can generate a very large printer spool file which may exhaust VM/370 system spooling space.) The log trace printed output should be retrieved, annotated with any descriptive comments, and attached to pertinent console output showing the trace commands and to a virtual storage dump taken immediately following the malfunction, as described above. (A virtual storage dump may be taken while log trace is active without any conflict.)

If it is suspected that spool files are being altered in transmission (resulting in partial files, missing or duplicated records), copies of the original and altered files should be retrieved if possible. When a CMS DISK DUMP file cannot be loaded by a CMS DISK LOAD command due to file format errors, a CMS READCARD command should be issued to obtain an exact copy of the altered file. Other documentation of the problem (console output listings, line log trace, and virtual storage dump) should be obtained as described above, if possible.

The documentation of a malfunction should be analyzed to diagnose the problem and identify the malfunctioning system component, if possible. There is no standard procedure for this kind of analysis. In general, system documentation and program listings can be used to determine normal system behavior which should appear in the log traces, dumps, etc. Any deviations or intermittent behavior would be suspect, and should be investigated. Often such discoveries sufficiently specify a particular malfunction to allow identification of a responsible routine or component which can be easily analyzed and tested in detail.

For example, suppose the diagnostic information indicates apparently normal communication, with a very high transmission exchange rate as indicated by the TRACE linkid SUM command, and no file data transfer. Analysis of a storage dump and log trace could show that a single CMD/MSG element is being repeatedly exchanged on the link. This suggests that a loop exists in the network routing structure in the user-defined RSCS directories at the network locations. In other words, the CMD/MSG element may be addressed to a location that each side of the link has accidentally routed to the other. This frequent source of trouble can be corrected by identifying the erroneously routed location ID from the storage dump or log trace, and issuing a ROUTE locid OFF command on either side of the link.

When problems are determined, any diagnosis along with the problem documentation and proposed correction (if any) should be forwarded to the people responsible for maintenance of the component suspected to be in error. It is most important to include a complete description of the configuration of the system in use, including specific hardware types and models, particular software systems, modification levels, and any local modifications in use.

To analyze a problem documented in an RSCS storage dump, you need to determine the status of the RSCS system tasks at the time of the dump. Lay out the dump with a visible marker such as a colored felt pen.

The locations of the tasks, modules, save areas, queues, tables and indicators all start with the pointers in the SVECTORS table beginning at storage location X'200', and in the common areas addressed by the TVECTORS in SVECTORS.

The Data Area Aids diagrams (in Section 5 of this manual) help you use the SVECTORS and other system control data. Use these diagrams, source listings, and the data area and control block detailed descriptions (also in Section 5 of this manual) to determine such things as active I/O, inactive I/O, tasks awaiting dispatching, tasks that have issued waits on synch locks, etc.

When a line driver task is executing and a dump condition occurs (for example, due to a program check) the name of the line driver is at the top of the dump.

The SVECTORS ACTIVE field, one byte at address X'210', contains the id of the currently active task. If the value in this field is X'00', the dispatcher has found no task in the task table that is ready to run, and the supervisor is either executing or waiting (according to current PSW wait bit) for an interrupt. The address at location X'211'-X'213' points to the task element of the last task to be dispatched before the dump was taken.

If a dump is automatically taken by RSCS, the active register contents at the time of error are stored, beginning with register 0, at location X'100B0'.

The value in register 14 (at location X'10040') in this save area is usually the address that the most recently active task exited from (BALR 14, 15).

Module Message Directory

The following list identifies the module code where the decisions are made to issue RSCS messages.

Message Source Module	Number ID	Generated at Label
DMTAXM101I		TAGPEND
DMTAXM102I		ACCEPEND
DMTAXM103E		ACCEPUR2
DMTAXM104I		CLOOSCN2
DMTAXM105I		CLOIPGE1
DMTAXM106I		FILSTRY
		OPENPOOF
DMTAXM107I		UNPECHEK
DMTAXM108E		OPENRDER
DMTAXM109I		REORD(ER)
		REFILCON
DMTAXM111I		CLOSCAN1
DMTAXM500I		CLOFINIS
DMTAXM501E		CLOSE
DMTAXM502E		CLOFCHEK
DMTAXM520I		CHANGE
DMTAXM521I		CHANHO
DMTAXM522I		CHANNOH
DMTAXM523I		CHANSCAN
DMTAXM524E		CHANGE
		ORDECHEK
		PURGCHEK
		TRANCHEK
DMTAXM525E		CHANGE
		CLOFCHEK
		ORDECHEK
		PURGCHEK
		TRANCHEK
DMTAXM526E		CHANGE
		CLOFCHEK
		ORDECHEK
		PURGCHEK
		TRANCHEK
DMTAXM640I		PURGDONE
DMTAXM645I		TRANDONE
DMTCMX001I		CMXFINXT
DMTCMX003I		CMXM003
DMTCMX004I		CMXM003
DMTCMX005I		CMXM003U
DMTCMX170I		MSGM170
		MSGLENOK
DMTCMX171I		MSGM170
DMTCMX201E		CMXLGOT
		CMXMISS
DMTCMX202E		A1TOCHK

Message Source Module	Number ID	Generated at Label
		A1TRANLK
		DEFLKNEW
		DEFNOLNK
		QYOLINK
		QYCKROUT
		QYINACTV
DMTCMX203E		A1FLKGOT
		A1FSTOW
		CHALKGOT
		FLUASCAN
		L2FLKGOT
		QYOFILE
		QYOFNULL
DMTCMX204E		A1TOCHK
		A1INPUT
		A1OUTPUT
		A1TRAMIS
		CHALKGOT1
		CHALKGT1
		CHANTERM
		CHASCAN
		CKRNGE
		CMXM204
		CP
		CPQIND
		CPQUERY
		CPQLOG
		CPQM204
		CPQNAMES
		CPQUSERS
		EXEC
		FLUMORE
		FORERR
		HT
		L0FOUL
		L0HOLD
		L0TRACE
		L1TERM
		QYROUGO
		QYROUMIS
		QYTOOMCH
		QYOFILE
		QYOLINK
		QYOSYSTM
		R0FORMAT
		REORDER
		ROSCAN
		ROUTCONT
		SHUTDOWN
		STALKGOT
DMTCMX205E		A1TOCHK
		CHACLASS
		CHACOPY
		CHAHOLD
		CHANOHOL
		CHAPRIOR
		FLUKEYWD
		L0TKEYWD
		R0CLASS
		R0KEEP
		R0ZONE
		R0LINE
		R0TASK
		R0TYPE
DMTCMX206E		CHACLASS
		CHACOPY
		CHADIST
		CHANAME
		CHAPRIOR
		LOHOLD
		LOTRACE
		L1FLKGOT
		QUERY
		R0CLASS
		R0CLMULT
		R0KEEP
		R0ZONE
		R0LINE
		R0TASK
		R0TYPE
DMTCMX208E		A1TOCHK
		CPQUSERS
		DISCONN
		MSGLNKGO
		MSGNOUSR
DMTCMX209E		CMXHIT
DMTCMX210E		CMD
		CMDNOLOC
		MSG
		MSGNOLOC
DMTCMX300I		CMXALRDY
DMTCMX301E		CMXALRDY
DMTCMX302E		A1TRAULK
		MSGNOLNK
DMTCMX303E		CMD
		CMXM303
		FORCE
		LOFLKGOT
		L1FLKGOT
		L2FLKGOT
		MSG
DMTCMX304E		CMXALRDY
DMTCMX310E		CMDNOLNK
		QYNOTROU
		MSGNOLNK
DMTCMX320E		CMD
		MSG
DMTCMX540I		DEFDO
DMTCMX541I		DEFDO
DMTCMX542E		DEFINE
		DEFM542
DMTCMX543E		DEFM543
		DEFNEXT
		DEFNOLNK
DMTCMX544E		DEFDO
		DEFM544
DMTCMX550I		DELDELET
DMTCMX551E		DELETE
DMTCMX552E		DELETE
DMTCMX560I		DISCHARG
DMTCMX561E		DISCONN
DMTCMX625I		QYOSY625

Message Source Module	Number ID	Generated at Label
DMTCMX626I		QYOSYNIN
DMTCMX627I		QYOSYPLF
DMTCMX630I		CMXM630
		ROUTGOT
DMTCMX631I		CMXM631
		ROUOFF
DMTCMX632E		CMXM632
		ROUSET
DMTCMX633E		CMXM633
		ROUSET
DMTCMX634I		QYOSYNRT
DMTCMX636I		QYROUGO1
		QYOSYROU
DMTCMX637I		CMXM637
		QYROUMIS
		ROUOFF
DMTCMX651I		QYINACTV
DMTCMX652I		QYM652
		QY1SNOD
DMTCMX653I		QYM653
DMTCMX654I		QY1QUGO
DMTCMX655I		QY1QUEUE
DMTCMX656I		QY1ACTVE
DMTCMX660I		QY2STAT
		QYM660
DMTCMX661I		QYM661
DMTCMX662I		QY2RSS
		QYM662
DMTCMX663I		QYM663
DMTCMX664E		QY2RSS
		QYM664
		QY2STAT
		QY2VNOH
DMTCMX665I		QY1AM665
DMTCMX670I		QYSYCON
DMTCMX671I		QYM671
DMTCMX672I		QYM672
DMTCMX673I		QYM673
DMTCMX674I		QYOSQNXT
		QY1M674
DMTCMX675E		EXECPROC
		EXECM675
DMTCMX676E		EXECPROC
		EXECM676
DMTCMX677E		EXECPROC
		EXECM676
DMTCMX678E		EXEC
		EXECM678
DMTCMX700I		STALNGOT
DMTCMX701E		STACREAT
DMTCMX702E		STACREAT
DMTCMX703E		STACREAT
DMTCMX704E		STACREAT
DMTCMX705E		STACRERR
DMTCMX706E		STACRERR
DMTCMX707E		STACRERR
DMTCMX708E		STACRERR
DMTCMX709E		STACRERR
DMTCMX710E		STAMAXER
DMTCMX750E		STANOTCL
DMTCMX751I		CMXALRDY
		STAM751
DMTINI410I		WRDONE
DMTINI411R		IPLDISK
DMTINI412R		RDORWRT
DMTINI413R		NUCCYLN
DMTINI481E		HEXERR
DMTINI482E		DECERR1
		DECLOOP
		DECPACK
DMTINI483E		RDORWRT
		IPLZERO
DMTINI484E		BADIPLD
DMTINI485E		NUCCYLN
DMTINI498S		WRERROR
DMTINI499T		RDERROR
DMTIRX000I		IRXGOTTN
DMTIRX400I		IRXGOTTN
DMTIRX449I		GENTAGS
DMTIRX450E		GENGOOD
		GENNOT5
		GENPARM
		GENPSCAN
DMTIRX451E		GENLINK
		GENPARM
		GENPORT
		GENROUTE
		GENSPMF
		GENTAGS
DMTIRX452E		GENLOCAL
DMTIRX453E		GENPARM
DMTIRX454E		GENTAGS
DMTIRX455E		GENROUTE
DMTIRX456E		GENLINK
DMTIRX457E		GENPORT
DMTIRX458E		GENPARM
		GENROUTE
DMTIRX461E		GENLOCAL
		GENROUTE
DMTIRX462E		GENLINK
		GENPARM
		GENROUTE
DMTIRX463E		GENLINK
DMTIRX464E		GENLPTRY
		GENPORT
DMTIRX465E		GENLOCAL
		GENLZTRY
DMTIRX466E		GENLTTRY
DMTIRX467E		GENLCTRY
DMTIRX468E		GENLKTRY
DMTIRX469E		GENTAGS
DMTIRX490T		IRXDISCO
DMTIRX491T		IRXDISCO
		GENFINIS
DMTIRX492T		IRXGOTTN
DMTIRX493T		IRXGOTTN
DMTIRX494T		GENFINIS
DMTIRX495T		IRXM495
DMTNCM108E		VMSPGET

Message Source Module	Number ID	Generated at Label
DMTNCM141I		ISIO
DMTNCM142I		RISIO2
		SIGNCTCG
DMTNCM143I		EOJ
DMTNCM146I		RLOC1
DMTNCM147I		RDEOF
DMTNCM190E		VMSP1
DMTNCM511E		SBKFWDN
DMTNCM531I		SETFWD
DMTNCM570I		SETDRAIN
DMTNCM571E		SETDRER1
DMTNCM580I		RDORJECT
		RDRJECT
		SETFLSHG
DMTNCM581E		SETFLERR
DMTNCM590I		SETFREE
DMTNCM591E		SETFRER1
DMTNCM610I		SETHOLD
DMTNCM611I		ALLHLD
		SETHLDIM
DMTNCM612E		SETHLDE1
DMTNCM750E		SETSTRT1
DMTNCM752I		SETSTART
DMTNCM801I		SETTR3
DMTNCM802I		LOGCLOSE
DMTNCM803I		SETTR2
DMTNCM810E		SETTR810
DMTNCM811E		SETTR811
DMTNCM812E		SETTR812
DMTNCM813E		SETTR813
DMTNCM905I		MC70K
DMTNCM914E		MC7A914
DMTNCM915E		MC7A915
DMTNCM916E		MC7A916
DMTNHD144I		OPENSCLP
DMTNHD145I		HOJOBTLP
DMTNHD910E		TAGSNERR
DMTNHD917I		HOJOBGO
DMTNIT204E		INTSNERR
DMTNIT708E		ISTORERR
DMTNIT911E		IBUFFERR
DMTNIT912E		IRESTERR
DMTNIT913E		INTPASER
DMTNPT070E		IOERRPR1
DMTNPT108E		VMSGET
DMTNPT141I		NPTEINIT
DMTNPT142I		NPTEINIT
DMTNPT143I		LINEDIS2
		LINEDROP
DMTNPT144I		PUTOPEN
DMTNPT145I		PUTCLS1
DMTNPT146I		GETGOT2
DMTNPT147I		GETPURGE
DMTNPT149I		TRPRT
DMTNPT190E		VMSP1
DMTNPT510I		GTBKMSG
DMTNPT511E		SBKFWDN
DMTNPT570I		SETDRAIN
DMTNPT571E		SETDRER1
DMTNPT580I		GETFLUSH
DMTNPT581E		SETFLUSH
		GETFLSHE
DMTNPT590I		SETFREE
DMTNPT591E		SETFRER1
DMTNPT600I		GDGODNE
DMTNPT610I		SETHOLD
		GETFILE
DMTNPT611I		SETHLDIM
		GETFILE
DMTNPT612E		SETHLDE1
DMTNPT750E		SETSTRT1
DMTNPT752I		SETSTART
DMTNPT801I		SETTR3
DMTNPT802I		LOGCLOSE
DMTNPT803I		SETTR2
DMTNPT810E		SETTR810
DMTNPT811E		SETTR811
DMTNPT812E		SETTR812
DMTNPT813E		SETTR813
DMTNPT902E		CONFCK1
DMTNPT903E		SPASSE
DMTNPT904E		SGNERR
DMTNPT905I		NPTGETX
DMTNPT907E		ENDSCAN
DMTNPT934E		PUTCLOSE
DMTNPT936E		GETGOT1
DMTPOW070E		IOERRPR1
DMTPOW108E		VMSPGET
DMTPOW141I		ISIO
DMTPOW142I		SIGNOK
DMTPOW143I		EOJ
DMTPOW144I		POPEN
DMTPOW144I		UOPEN
DMTPOW145I		PCLOSE
DMTPOW145I		UCLOSE
DMTPOW146I		RLOC1
DMTPOW147I		RDEOF
DMTPOW170I		MRTCONT2
DMTPOW190E		VMSP1
DMTPOW531I		CMDNSUP
DMTPOW531I		MSGIGN
DMTPOW570I		SETDRAIN
DMTPOW571E		SETDRER1
DMTPOW590I		SETFREE
DMTPOW591E		SETFRER1
DMTPOW610I		SETHOLD
DMTPOW611I		ALLHLD
DMTPOW612E		SETHLDE1
DMTPOW750E		SETSTRT1
DMTPOW752I		SETSTART
DMTPOW801I		SETTR3
DMTPOW802I		LOGCLOSE
DMTPOW803I		SETTR2
DMTPOW810E		SETTR810
DMTPOW811E		SETTR811
DMTPOW812E		SETTR812
DMTPOW813E		SETTR813
DMTPOW902E		MC7ERR

Message Source Module	Number ID	Generated at Label
DMTPOW903E		MC7
DMTPOW905I		MC7B
DMTPOW908E		POWIERR3
DMTPOW913E		POWIERR1
DMTPOW940E		RREJECT
DMTPOW941E		PCMDERR1
DMTPOW942E		PCMDERR2
DMTPOW943E		PCMDERR3
DMTPOW944I		WGET2
DMTPOW945I		DEOJ
DMTPOW946E		POWIERR2
DMTREX000I		REXICGOT
DMTREX002I		TERLHIT
DMTREX080E		TERLHIT
DMTREX090T		REXPTERM
DMTREX091T		REXITERM
DMTREX679I		EXECMSG
DMTRGX170I		RGXMSG
DMTRGX171I		RGXMSG
DMTRGX300I		RGXNTHR1
DMTRGX301E		RGXNTHR1
DMTRGX302E		RGXNTHR1
DMTRGX303E		RGXNTHR1
DMTRGX304E		RGXNTHR1
DMTRGX320E		RGXNTHR1
DMTRGX330E		RGXMSGER
DMTRGX331E		RGXMSGER
DMTRGX332E		RGXMSGER
DMTSML070E		IOERRPRT
DMTSML108E		VMSPGET
DMTSML141I		ISIO
DMTSML142I		SIGNOK
DMTSML143I		EOJ
DMTSML144I		JOUTPUT
		PCONT
		UOUTPUT
DMTSML145I		JCLOSE1
		PCLOSE
		UCLOSE
DMTSML146I		RLOC1
DMTSML147I		RDEOF
DMTSML149I		TRPRT
DMTSML170I		WGET2
DMTSML190E		VMSP1
DMTSML510I		RDBKMSG
DMTSML511E		SBKFWDN
DMTSML530I		SETCMD
DMTSML570I		SETDRAIN
		$USRNPUN
DMTSML571E		SETDRER1
DMTSML580I		RDFLUSH
DMTSML581E		SETFLUSH
		RDFLSHER
DMTSML590I		SETFREE
DMTSML591E		SETFRER1
DMTSML600I		RDGODNE
DMTSML610I		SETHOLD
DMTSML611I		ALLHLD
		SETHLDIM
DMTSML612E		SETHLDE1
DMTSML750E		SETSTRT1
DMTSML752I		SETSTART
DMTSML801I		SETTR3
DMTSML802I		LOGCLOSE
DMTSML803I		SETTR2
DMTSML810E		SETTR810
DMTSML811E		SETTR811
DMTSML812E		SETTR812
DMTSML813E		SETTR813
DMTSML901E		SMLIERR1
		TCTR
DMTSML902E		MC7ERR
DMTSML903E		MC7A
DMTSML905I		MC7B
DMTSML906E		SMLIERR2
		TCTR
DMTSML934E		JCLOSE
DMTSML935E		READCON
DMTVMB141I		VMRENAUT
DMTVMB142I		VMREN10
DMTVMB143I		LINEDROP
		VMRQUIT
DMTVMB144I		PUTOPEN1
DMTVMB145I		PUTDONE
DMTVMB146I		GETGOTR
DMTVMB147I		GETFDEOF
DMTVMB148I		GETGOT1
DMTVMB510I		GETBKFIL
DMTVMB511E		SBKFWDN
DMTVMB570I		SETDRAIN
DMTVMB571E		SETDRER1
DMTVMB580I		GETFLUSH
		RRESET
DMTVMB581E		GETFLSHE
DMTVMB590I		SETFREE
DMTVMB591E		SETFRER1
DMTVMB610I		SETHOLD
DMTVMB611I		GETFILE
		SETHLDIM
DMTVMB612E		SETHLDE1
DMTVMB750E		SETSTRT1
DMTVMB752I		SETSTART
DMTVMB581E		SETFLUSH
DMTVMB801I		SETTR3
DMTVMB802I		LOGCLOSE
DMTVMB803I		SETTR2
DMTVMB810E		SETTR810
DMTVMB811E		SETTR811
DMTVMB812E		SETTR812
DMTVMB813E		SETTR813
DMTVMB905I		PASSM905
DMTVMB914E		PASSM914
DMTVMB918E		PASSM918
DMTVMB919E		PASSM919
DMTVMC108E		MAKEBLOC
DMTVMC141I		CTCEINIT
		VMCEINIT

Message Source Module	Number ID	Generated at Label
DMTVMC142I		CTCWHAT
		RESPCHK
DMTVMC143I		LINEDIS2
DMTVMC144I		PUTNHEAD
DMTVMC145I		PUTCLOSE
DMTVMC146I		GETGOT
DMTVMC147I		GETCONT
DMTVMC511E		SBKFWDN
DMTVMC570I		SETDR1
DMTVMC571E		SETDRER1
DMTVMC581E		SETFLUSH
DMTVMC590I		SETFREE
DMTVMC591E		SETFRER1

Message Source Module	Number ID	Generated at Label
DMTVMC610I		SETHOLD
DMTVMC611I		SETHLDIM
DMTVMC612E		SETHLDE1
DMTVMC750E		SETSTRT1
DMTVMC752I		SETSTART
DMTVMC801I		SETTR3
DMTVMC802I		LOGCLOSE
DMTVMC803I		SETTR2
DMTVMC810E		SETTR810
DMTVMC811E		SETTR811
DMTVMC812E		SETTR812
DMTVMC813E		SETTR813

Trace Log

Each line driver builds and executes channel programs to control its BSC telecommunication adapter. In the event of a malfunction, an analysis of these I/O transactions can often provide a quick problem diagnosis. RSCS will generate a printed sequential log of each channel program executed by a line driver during an interval controlled by the "TRACE linkid LOG" and "TRACE linkid NOLOG" RSCS operator commands.

Each I/O channel program log consists of one or more printed lines. The 'composite' CSW (an ORing of all CSWs presented for the I/O device during execution of the channel program) appears in columns 44-57 on the first line of each transaction entry. If the "unit check" CSW status bit is set on, the telecommunication adapter sense byte appears in columns 59-60 to the right of the composite CSW. (The sense data are always zero when the unit check CSW status bit is zero.) The first CCW in the channel program for the I/O transaction appears in columns 63-78, to the right of the composite CSW and sense byte, if present. If a command chaining or data chaining bit is set on in a CCW, the following printed log line describes the next CCW in the transaction's channel program. In this case, the CSW and sense byte fields are blank unless a TIC CCW follows the preceding CCW. When a TIC is encountered in the channel program, the character string C'<-----TIC' appears in columns 44-53 of the log line, and the CCW which is the object of the TIC appears in the line's normal CCW field.

Each line of the log must contain a non-TIC CCW. Columns 1-42 of each line contain the read or write buffer addressed by the CCW appearing on the line. For an ending CCW, the buffer is truncated according to the CCW count, decremented by the CSW residual count when the ending CCW is a READ. For non-ending CCWs, the entire buffer is logged according to the CCW count. When the buffer is too large to fit within the printed log buffer field, the first 14 and last 6 bytes of the buffer appear, separated by a double dash. When the CSW residual count indicates that no data were read by an ending READ CCW, a double dash appears in the first two columns of the data buffer log entry. The contents of the RSCS log record are as follows:

1-42 The data buffer described by the CCW to the right, truncated according to the CCW count, the CCW residual count and space constraints.

44-57 Bytes 1-7 of the composite ending CSW, on the first line of a transaction log.

59-60 The sense byte, if any.

63-78 A CCW included in the line transaction channel program.

81-120 The graphic EBCDIC representation of the first 40 bytes of the data buffer (DMTPOW line driver only).

The fields of the record are separated by blanks. The following are samples of read and write log records for NPT:

```
2D                                          0797E00C000003       02079BF820000004
1070                                        0798280C0001AC       0107987A60000002
1002E2C9C7D5D6D5404040E3C5E2-404040404003                        02079BF820000200
1061                                        0798280D0001FF       0107987A60000002
37                                                               02079BF820000200
--                                          0797E00E000004   01  02079BF820000004
--                                          0797E00E000004   01  02079BF820000004
```

```
     First 14 bytes          |Last 6 bytes|      CSW      |Sense|    CCW
                             |            |               |Byte |
-----------------------------|            |               |     |
           TP Buffer         |            |               |     |
```

Note: The dashes in record positions 1 and 2 indicate that there was no data transfer for that I/O transaction.

Appendix A: MULTI-LEAVING Description

"MULTI-LEAVING" is the name of a computer-to-computer communication protocol developed for use by the HASP system and used by RSCS. MULTI-LEAVING can be defined as the fully synchronized, pseudo-simultaneous, bidirectional transmission of a variable number of data streams between two or more computers using Binary Synchronous Communications (BSC) facilities.

MULTI-LEAVING in RSCS

This appendix contains an overview of a comprehensive MULTI-LEAVING communications system (as is used in HASP/ASP). The VM/370 support for programmable BSC workstations is consistent with the MULTI-LEAVING design, but it does not use certain features provided in MULTI-LEAVING:

- The transmission of record types other than print, punch, input, console, and control is not supported.
- The only general control record type used is the terminal sign-on control.
- Only SCB count units of 1 are used.
- Column binary cards are not supported.

In addition, the support provided for the MULTI-LEAVING based protocol used by VSE/POWER differs from that provided for HASP-type systems in the following areas:

- A Request-to-Initiate Function transmission is required before all data transmissions.
- No SCB fields are present in the General Control Record type records.
- Carriage control information is not carried in the SRCB.

MULTI-LEAVING Philosophy

The basic element for MULTI-LEAVING transmission is the character string. One or more character strings are formed from the smallest external element of transmission, the physical record. These physical records are input to MULTI-LEAVING and may be any of the classic record types (card images, printed lines, tape records, etc.). For efficiency in transmission, each data record is reduced to a series of character strings of two basic types:

1. A variable-length nonidentical series of characters
2. A variable number of identical characters

Character Strings and the SCB

An eight-bit control field, termed a String Control Byte (SCB), precedes each character string to identify the type and length of the string. Thus, a string as in 1 above is represented by an SCB followed by the nonduplicate characters. A string of consecutive, duplicate, nonblank characters (as in 2 above) can be represented by an SCB and a single character; the SCB indicates the duplication count, and the character following it is the one to be duplicated. In the case of an all-blank character string, only an SCB is required to indicate both the type and the number of blank characters.

A data record to be transmitted is segmented into the optimum number of character strings (to take full advantage of the identical character compression) by the transmitting program. A special SCB is used to indicate the grouping of character strings in the original physical record. The receiving program can then reconstruct the original record for processing.

Transmission Blocks and the RCB

To allow multiple physical records of various types to be grouped together in a single transmission block, an additional eight-bit control field precedes the group of character strings representing the original physical record. This field, the Record Control Byte (RCB), identifies the general type and function of the physical record (input stream, print stream, data set, etc.). A particular RCB type has been designated to allow the passage of control information between the various systems. Also, to provide for simultaneous transmission of similar functions (that is, multiple input streams, etc.), a stream identification code is included in the RCB.

A second eight-bit control field, the Sub-Record Control Byte (SRCB), is also included immediately following the RCB. This field is used to supply additional information concerning the record to the receiving program. For example, in the transmission of data to be printed, the SRCB can be used for carriage control information.

MULTI-LEAVED Data Streams and the FCS

For actual MULTI-LEAVING transmission, a variable number of records may be combined into a variable block size, as indicated previously (that is, RCB, SRCB, SCB1, SCB2, ..., SCBn, RCB, SRCB, SCB1, ..., etc.). MULTI-LEAVING provides for two (or more) computers to exchange transmission blocks, containing multiple data streams as described above, in an interleaved fashion. To allow optimum use of this capability, however, a system must have the capability to control the flow of a particular data stream while continuing normal transmission of all others. This requirement becomes obvious if one considers the case of the simultaneous transmission of two data streams to a system for immediate transcription to physical I/O devices of different speeds (such as two print streams).

To meter the flow of individual data streams, a Function Control Sequence (FCS) is added to each transmission block. The FCS is a sequence of bits, each of which represents a particular transmission stream. The receiver of several data streams can temporarily stop the transmission of a particular stream by setting the corresponding FCS bit

off in the next transmission to the sender of that stream. The stream can later be resumed by setting the bit on.

Transmission Data Integrity and the BCB

Finally, for error detection and correction purposes, a Block Control Byte (BCB) is added as the first character of each block transmitted. The BCB, in additional to control information, contains a module 16 block sequence count. This count is maintained and verified by both the sending and receiving systems to control lost or duplicated transmission blocks.

In addition to the normal binary synchronous text control characters (STX, ETB, etc.) MULTI-LEAVING uses two of the BSC control characters, ACK0 and NAK. ACK0 is used as a "filler" by all systems to maintain communications when data is not available for transmission. NAK is used as the only negative response and indicates that the previous transmission was not successfully received.

A typical MULTI-LEAVING transmission block looks like this:

DLE	BSC Leader (SOH if no transparency feature)
STX	BSC Start-of-Text
BCB	Block Control Byte
FCS	Function Control Sequence
FCS	Function Control Sequence
RCB	Record Control Byte for record 1
SRCB	Sub-Record Control Byte for record 1
SCB	String Control Byte for record 1
DATA	Character String
SCB	String Control Byte for record 1
DATA	Character String
SCB	Terminating SCB for record 1
RCB	RCB for record 2
SRCB	SRCB for record 2
SCB	SCB for record 2
DATA	Character String
SCB	Terminating SCB for record 2
RCB	Transmission Block terminator
DLE	BSC Leader (SYN if no transparency feature)
ETB	BSC Ending Sequence

MULTI-LEAVING Control Specification

This section describes the bit-by-bit definitions of the various MULTI-LEAVING control fields and includes notes concerning their use.

BLOCK CONTROL BYTE (BCB)

Binary	Meaning
r...	Reserved (must be 1)
.xxx	Control information as follows: .000 .... - Normal block .001 .... - Bypass sequence count validation

```
                .010 cccc - Reset expected block sequence count
                            to "cccc"
                .011 .... - Reserved for future use
                .100 .... - Reserved for future use
                .101 .... - Not used
                .110 .... - Not used
                .111 .... - Reserved for future use

.... cccc       Module 16 block sequence count
```

FUNCTION CONTROL SEQUENCE (FCS)

Binary	Meaning
r... .... r...	Reserved (must be 1... .... 1... ....)
.0..	Normal state
.1..	Suspend all stream transmission (WAIT-A-BIT)
..rr .... ..rr	Reserved for future use
.... .1..	Remote console stream identifier
.... 1...	Function stream identifier for: RJE input stream number 1 RJE print stream number 1 NJI/NJE job transmission stream number 1
.... .1..	Function stream identifier for: RJE input stream number 2 RJE print stream number 2 RJE punch stream number 7 NJI/NJE job transmission stream number 2 NJI/NJE SYSOUT transmission stream number 7
.... ..1.	Function stream identifier for: RJE input stream number 3 RJE print stream number 3 RJE punch stream number 6 NJI/NJE job transmission stream number 3 NJI/NJE SYSOUT transmission stream number 6
....1	Function stream identifier for: RJE input stream number 4 RJE print stream number 4 RJE punch stream number 5 NJI/NJE job transmission stream number 4 NJI/NJE SYSOUT transmission stream number 5
.... 1...	Function stream identifier for: RJE input stream number 5 RJE print stream number 5 RJE punch stream number 4 NJI/NJE job transmission stream number 5 NJI/NJE SYSOUT transmission stream number 4
.... .1..	Function stream identifier for: RJE input stream number 6 RJE print stream number 6 RJE punch stream number 3 NJI/NJE job transmission stream number 6 NJI/NJE SYSOUT transmission stream number 3
....1.	Function stream identifier for: RJE input stream number 7 RJE print stream number 7 RJE punch stream number 2 NJI/NJE job transmission stream number 7 NJI/NJE SYSOUT transmission stream number 2
....1	Function stream identifier for: RJE punch stream number 1 NJI/NJE SYSOUT transmission stream number 2

RECORD CONTROL BYTE (RCB)

Binary *	Hex	Meaning
0000 0000	00	End-of-block
iiii iiii	01-8F	Reserved for future use
1001 0000	90	Request to initiate function (SRCB=RCB of function)
1010 0000	A0	Permission to initiate function (SRCB=RCB of function)
1100 0000	C0	Acknowledge of transmission complete (SRCB=RCB of function)
1101 0000	D0	Not used
1110 0000	E0	BCB sequence error
1111 0000	F0	General control record
1001 0001	91	RJE console message
1iii 0001	A1-F1	Reserved for future use
1001 0010	92	RJE operator command
1iii 0010	A2-F2	Reserved for future use
1iii 0011	93-F3	RJE input record
1iii 0100	94-F4	RJE print record
1iii 0101	95-F5	RJE punch record
1iii 0110	96-F6	Data set record
1iii 0111	97-F7	Terminal message routing request
1iii 1000	98-F8	NJI/NJE input record
1iii 1001	99-F9	NJI/NJE SYSOUT record
1001 1010	9A	NJI/NJE operator command/message
1iii 1010	AA-FA	Reserved for future use
1001 1011	9B	Reserved
1iii 1011	AB-FB	Reserved for future use
1iii 1100	9C-FC	Reserved for future use
1iii 1101	9D-FD	Not used
1iii 1110	9E-FE	Not used
1iii 1111	9F-FF	Not used

* i denotes a position whose value (1 or 0) depends on the hexadecimal value within the range in the column labelled "Hex."

SUB-RECORD CONTROL BYTE (SRCB)

RCB	SRCB
00	None
90	RCB of function to be initiated
A0	RCB of function to be initiated
B0	RCB of function to be cancelled
C0	RCB of function which is complete
E0	Expected count (received count is in BCB)
F0	An identification character as follows: A = Initial RJE SIGN-ON B = Final RJE SIGN-OFF C = Print initialization record D = Punch initialization record E = Input initialization record F = Data set transmission initialization G = System configuration status H = Diagnostic control record I = Initial network SIGN-ON J = Response to initial network SIGN-ON K = Reset network SIGN-ON L = Accept (concurrence) network SIGN-ON M = Add network connection N = Delete network connection O-R = Reserved for future use S-Z = Unused
91	1000 0000 (X'80') 0000 0000 (X'00') if VSE/POWER EOF record
92	1000 0000 (X'80') 1110 1001 (X'E9') if VSE/POWER PSTOP LINE command 0000 0000 (X'00') if VSE/POWER EOF record
93-F3	1000 0000 (X'80') 1000 0001 (X'81') if VSE/POWER 0000 0000 (X'00') if VSE/POWER EOF record
94-F4	Carriage control information as follows: 1010 00nn - Space immediately "nn" spaces (not used) 1011 cccc - Skip immediately to channel "cccc" (not used) 1000 00nn - Space "nn" lines after print 1000 1100 - Load printer FCB image 1001 cccc - Skip to channel "cccc" after print 1000 0000 - Print and suppress space 1000 0001 (X'81') if VSE/POWER 0000 0000 (X'00') if VSE/POWER EOF record
95-F5	1000 1111 (X'8F') 0000 0000 (X'00') if VSE/POWER EOF record
96-F6	Undefined
97-F7	Undefined
98-F8	NJI/NJE input control information as follows: 1000 0000 - Normal input record 1100 0000 - Job header

```
             1110 0000 - Data set header
             1101 0000 - Job trailer
             1111 0000 - Data set trailer (not used)

99-F9     NJI/NJE SYSOUT control information as follows:
             10cc 0000 - Carriage control type as follows:
                         1000 0000 - No carriage control
                         1001 000  - Machine carriage control
                         1010 0000 - ASA carriage control
                         1011 0000 - Reserved for future use
             11cc 0000 - Control record as follows:
                         1100 0000 - Job header
                         1110 0000 - Data set header
                         1101 0000 - Job trailer
                         1111 0000 - Data set trailer (not used)
             1000 ss00 - Spanned record control as follows:
                         1000 0000 - Normal record (not spanned)
                         1000 1000 - First segment of spanned record
                         1000 0100 - Middle segment of spanned record
                         1000 1100 - Last segment of spanned record

9A        1000 0000 (X'80')

9B        1000 0000 (X'80')


STRING CONTROL BYTE (SCB)

Binary          Meaning

0000 0000       End-of-record
                At first SCB, this also indicates end-of-file

100b bbbb       "bbbbb" blanks are to be inserted

101d dddd       The single character following this SCB is to be
                duplicated "ddddd" times

11cc cccc       The "cccccc" characters following this SCB are to be
                inserted
```

Appendix B: RSCS Preloader Utility Under CMS

The preloader (part of RSCS Networking) is a utility program that runs under CMS. It collects text files and reformats them into a single text file that can be dynamically loaded by the RSCS loader. It resolves external references and performs preliminary relocation of address constants. Its function is similar to that of a linkage editor, but its output is in standard text file format and does not include multiple CSECTs.

Line drivers and other programs to be loaded as RSCS tasks may be developed as multiple separate assembly modules which externally reference one another; the assembled text files may be merged into a single RSCS loadable text file by the preloader under CMS. The preloader is invoked as an ordinary CMS utility routine:

```
PRELOAD  loadlist  [control]
```

'loadlist' specifies the filename of an EXEC file which must reside on the caller's A-disk. Each record of this file contains the filename, and optional filetype, of a text (object) file to be used as preloader input. 'control' specifies the filename of a CNTRL file which must also reside on the caller's A-disk if it is specified. The format and interpretation of this file are the same as for the VM/370 VMFLOAD utility. If a load list entry includes a filetype, that filetype is used to identify the input file. Otherwise, if a control file is specified, input file identifiers are constructed using the filename from the load list entry and a filetype of the form 'TXT....' The highest control level identifier for which a file can be located on the caller's accessed disks is used. If no filetype is included in a load list entry and no control file is specified, a default filetype of TEXT is used. Input files are located by a scan of all the caller's disks in their access order.

Note: PRELOAD is not to be used to generate the RSCS nucleus.

The preloader output consists of two files, one with a filetype of TEXT, the other of filetype MAP, both with the same filename as that specified for the input load list. If either of these files already exists on the caller's A-disk, the old file is replaced by the new output file.

The output TEXT file is the merged and linked copy of the input files. The first CSECT or private code section in the input becomes the composite (single) output section; its length is the sum of all input section lengths (rounded up to doubleword multiples between sections for proper section alignment). For the output ESD, subsequent CSECTs are made into entries (RLDs), and subsequent private code sections are disregarded. External references are included in the output ESD only if they remain unresolved.

Input TXT records of non-zero length are relocated and written to the output file. The output RLD is a translated and relocated collection of all input RLD records. The output END card does not specify any entry point, section length, or other code. No sorting is done by the preloader. In general, each output ESD, TXT, and RLD entry appears in the same order as the input entry from which it was translated.

ADCON and VCON fields are relocated within their TXT records. ORG statements that cause relocatable constant fields to overlay or to be overlaid may cause results that differ from results obtained with a loader that completes TXT data loading prior to relocating ADCONs and

VCONs.

The output MAP file is a printable record of the preloader processing, similar to a load map. The first line of the map specifies the output text file name, its residence volume label and virtual device address, and the date and time of file creation. The next section of the map is a listing of the control file used, if one was specified. The remainder of the map consists of a sequence of input file sections, one for each input file in processing order.

The first line of a map input section specifies the input file's filename, filetype, filemode, residence volume label and virtual device address, and the file creation date and time. (If the input file was located on a disk accessed as a read-only A-disk extension, the filemode, volume label, and virtual device address of the A-disk are listed.) If the input file data contains invalid records, the preloader writes them in the map sequentially following the input file identification line. The VM/370 VMFASM utility enters such "invalid" records in text files to specify the updates and macro libraries used in assembly. Following these records, the input file's ESD is listed, including control sections and entries with their relocated addresses, duplicate external symbols, and unresolved external references, if any. The first control section in the input specifies the output control section name; the output section length is included on this ESD map entry.

The preloader does not actually load its input modules into storage before generating its output section, but rather interprets, translates, and relocates its input text files on a two-pass record by record basis. This approach requires that for each TXT record of a particular input control section, each RLD entry (one for each ADCON and VCON) which lies within that control section must be scanned to determine if it lies within the TXT record data. As a result, the preloader processing time has a component proportional to both the total number of TXT records and the total number of RLD entries for each input control section. Roughly stated, this means that when a particular input control section grows sufficiently large, the time required to process it becomes proportional not to the input control section size, but to the square of that size. This effect is significant when a large text file previously generated by the preloader is used as preloader input. In this case, much more CPU time may be required to reprocess the preloader output than was required to generate it in the first place, because several smaller control sections have been merged into a single large control section. This kind of program behavior can be expected, and does not indicate a malfunction.

INTERNALS

The preloader begins by reserving a large block of virtual free storage using DMSFREE. This block is used to build a list of control filetypes; a table of input file names, the Input File Table (IFT); a composite External Symbol Table (EST); chains of elements reflecting RLD entries; and a chain of elements reflecting relocatable constant fields which span a boundary between TXT records. MAINTOP internally allocates storage from the top of the storage block for table entries. MAINBOT internally allocates storage from the bottom of the block for chain elements.

INICNTRL reads and interprets the specified control file, and stacks a list of default filetypes at the bottom of storage for use in locating input files. IFTBUILD builds the IFT by reading the specified input load list file, and where no filetype is specified, attempting to locate the appropriate input file using filetypes from the control list in

repeated calls to the CMS STATE function. When the IFT is completely built, the control list storage is released for reuse. The IFT is used by INGET, the input read routine, to define the input sequence. A call to INGET returns the next sequential record from the input and the address of the active IFT entry, automatically switching files on end-of-file.

ESTBUILD calls INGET to read input records, passing control to DOESD when an ESD record is read, and to DORLD when an RLD record is read. Both DOESD and DORLD return control to ESTBUILD upon completion of record processing. When ESTBUILD encounters an END card, the remaining file input is flushed, and processing continues with the next file. All other input records are ignored. When the end of the last input file is reached, ESTBUILD passes control to GENERATE.

DOESD adds an entry to the EST for each ESD record entry encountered. When DOESD generates a new EST entry, a complete scan of the existing table is made for matching name entries. When matches are found, external references are resolved, synonyms are chained (for PRs and CMs), or duplicate name definitions are flagged as error conditions, depending on the types and statuses of the two entries. A common routine, SELECTYP, is used to decode ESD types. SELECTYP accepts an EST entry as input, and returns control at a fixed four-byte multiple displacement for each EST type group (SD-PC; LD; ER-WX; PR-CM; invalid).

DORLD builds an RLD chain element for each RLD record entry encountered. The SD or PC EST entry for the section in which the referenced relocatable constant is positioned is located, and the new RLD element is entered at the end of its position ID chain. The EST entry to which the RLD element is relative is similarly located, and the new RLD element is also entered at the end of its relative ID chain.

GENERATE scans sequentially through the EST, processing each entry. The total output section length is accumulated from SD and PC entry lengths, and output addresses for SD, PC, and LD entries are set to their cumulative displacements. The output ESD IDs are generated and set for unresolved ER and WX entries. (The first SD or PC entry encountered is marked as the 'prime' section entry, and its output ESD ID is generated and set as well.) For each LD entry, the SD or PC entry for the section in which the LD is positioned is located, the LD entry is chained to it, and the LD's relocated address is generated and set using the section's position relocation constant. For each uncompiled PR or CM entry, an output ESD ID is generated and set, the maximum length (and, for PRs, the maximum alignment) is established and set, synonym relative ID RLD chains are merged, and each entry on the synonym chain is marked "compiled". The cumulative output section length is saved for use as the prime (output) control section length.

RELOCATE sequentially scans through the EST, and for each resolved entry the position ID and relative ID RLD chains are relocated by adjusting each position ID element's constant address field, and storing the relative relocation constant in each relative ID element. For each unresolved EST entry (all uncompiled PRs and CMs, and unresolved ERs and WXs), the output ESD ID of the EST entry is stored in each RLD element of its relative ID chain, and each such RLD element is flagged as unresolved.

ESDWRITE starts the output text file generation. An interim file with the output filename and a filetype of PRELOAD is used for output generation. If such a file already exists it is erased, and the new output text file is built. The EST is sequentially scanned, and output ESD records are built and written using the EST information. The prime section entry is completed by setting its length to the cumulative output section length and its address to its displacement (zero), and it is entered in the output as an SD or PC. Other SDs are transformed to

LDs for output, and other PCs are ignored. LDs are simply entered as LDs in the output. Unresolved ERs and WXs and uncompiled PRs and CMs are included in the output, and other EST entries are ignored. When output generation is complete, if a file already exists with the output filename and a filetype of TEXT, it is erased, and the output PRELOAD file is renamed to a filetype of TEXT.

TXTWRITE rereads the entire input sequence from the beginning. When the first record of an input file is read (at TXTGET), MAPFILE is called to generate and write a map file line describing the input file. When a record which does not begin with X'02' is encountered, the record is written unmodified to the map file, and is otherwise ignored. When a TXT record with non-zero data count is read prior to the first END record for a particular file, it is translated, relocated, and written to the output as described below. All other input records (e.g., ESD, RLD, SLC) are ignored. When an input end-of-file is read, TXTWRITE calls MAPEOF, which formats the file's EST entry and writes it to the map file. If MAPEOF encounters duplicate external symbols or unresolved external references, they are included in the ESD map with the appropriate descriptive note, and a preloader status flag is set reflecting the presence of a duplicate or unresolved external symbol.

For each valid TXT record, the EST entry for the section in which the TXT data is positioned is located. The TXT record's position address is relocated using its section's position relocation constant, and its position ID is set to the prime section's output ESD ID. The section's position ID RLD element chain is then scanned for elements specifying relocatable constants within the TXT record's data. Each constant field entirely within the TXT record's data is relocated with the RLD element's relocation constant, provided that the RLD element is not marked "unresolved".

If a relocatable constant field lies only partially within the TXT record's data, the output TXT record is adjusted to exclude the overlap field, and a call is made to TXTFIX. This routine records the partial field data in an 'overlap element' on a chain in storage, along with the address of its associated RLD element. When a previously chained overlap element specifying the same RLD element is encountered, the new overlap data is merged with the old, and the overlap element is marked for output. When the end of the RLD element chain is reached, a test is made to see if an overlap element is ready for output. If so, a new TXT record is generated containing only the relocated constant field which spanned TXT records, and is written to the output file. The relocated (and, perhaps adjusted) TXT record is then written to the output file.

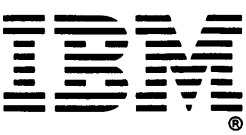
RLDWRITE sequentially scans the entire EST for SD and PC sections. For each such entry, the position ID RLD chain is scanned, and each RLD entry on the chain is used to generate an output RLD record entry. Each RLD position ID is set to the prime section's output ESD ID. For resolved RLD entries, the relative ID is set to the same and the flag is set to address type, and for unresolved entries the output ESD ID contained in the RLD chain element is used as the relative ID. For CXD entries, the relative ID is set to zero. The RLD address field has been relocated by RELOCATE and is used as is.

ENDWRITE writes a simple END record to output. FINIS performs final output file processing (already described), releases the main storage block using DMSFRET, issues a diagnostic message if duplicate or unresolved external symbols were encountered, and returns control to CMS.

Index

LY24-5203-0

IBM VM/370: RSCS Networking Logic (File No. S370-30) Printed in U.S.A. LY24-5203-0



International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N. Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N. Y., U. S. A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N. Y., U. S. A. 10601

IBM Virtual Machine Facility/370: Remote Spooling
Communications Subsystem Networking Logic
Order No. LY24-5203-0

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

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

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

	Yes	*No*
• Does the publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

• What is your occupation? _____

• How do you use this publication:

As an introduction to the subject?	☐	As an instructor in class?	☐
For advanced knowledge of the subject?	☐	As a student in class?	☐
To learn about operating procedures?	☐	As a reference manual?	☐

Your comments:

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Licensed Material — Property of IBM
LY24-5203-0

Reader's Comment Form

Cut or Fold Along Line

IBM VM/370: RSCS Networking Logic (File No. S370-30) Printed in U.S.A. LY24-5203-0

Fold and tape | Please Do Not Staple | Fold and Tape

First Class
Permit 10
Endicott
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department G60
P. O. Box 6
Endicott, New York 13760

Fold | Fold

If you would like a reply, *please print:*

Your Name _____

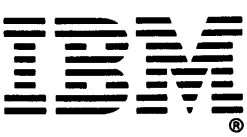
Company Name _____ *Department* _____

Street Address _____

City _____

State _____ *Zip Code* _____

IBM Branch Office serving you _____



International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N. Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N. Y., U. S. A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N. Y., U. S. A. 10601